SPOKEN LANGUAGE CLASSIFICATION FOR AUTOMATED CALL ROUTING

Diploma Thesis

of

GEORGIADOU DESPOINA

Electronic and Computer Engineering

Technical University of Crete

TUC

July 2014

Supervisor: Professor Digalakis Vasilis

ABSTRACT

Nowadays there is a raising interest in providing automated services via natural spoken dialog systems. By natural, we mean that the machine understands and acts upon what people actually say, in contrast to what one would like to say. There are many issues that arise when such systems are targeted for large populations of non-expert users.

Here, we focus on the task of automatically routing telephone calls (call routing) based on a user's spoken response to an open-ended prompt of 'How may I help you?'. We have a database generated from spoken transactions between customers and human agents and we describe methods for automatically understanding such data. Results evaluating call classification from speech are reported for that database. These methods are considered part of Spoken Language Understanding (SLU) in a Spoken Dialog System.

For a start, we provide a brief coverage of Spoken Dialog Systems in order to comprehend the part of Spoken Language Understanding (SLU). More specifically, focusing on the call-routing task, the main established approaches are covered and for each approach, we describe the algorithms used in order to provide the reader with a better view of the state of the art in this area. After that, we introduce the term 'salience' and explain the benefits of using phrases instead of words during the classification procedure. Two classification algorithms are featured which both extract salient phrases from the training dataset and use them in different ways so as to categorize a test call request.

Next, we examine the classification of our test calls depending on their topic using continuous mixture Gaussian distributions (Gaussian Mixture Models). The implementation work starts from the preprocessing of calls in order to appear into a more compact form, followed by the creation stage of dictionary and term weighting of bringing a text version of the classical representation through words in vector format. The following step is the reduction of the dimensions of the vectors using the Singular Value Decomposition (SVD), for the division of numerical linear algebra in tables. Finally, the final vectors are being used for the training of GMM.

In this work, we used an American-English dataset, which contains a high number of pre-processing call requests and classes. The accuracy of our models is compared to the algorithms referred as state-of-the-art and the extracted results as well as comparative tables and figures are offered for a deeper comprehension.

Index Terms: Call Routing, Spoken Dialog System, Spoken Language Understanding (SLU)

ACKNOWLEDGEMENTS

I would like to take some time here and thank the people who made the completion of this work possible.

First of all, I would like to express my sincere thanks to my thesis supervisor, Professor Digalakis Vasilis, for assigning this work to me.

Many thanks also go to Mr Diakoloukas Vasilis who provided his experience and knowledge whenever needed. I deeply appreciate all his guidance and support throughout this work.

Finally, I need to express my deepest gratitude to my best friend for all the patience and mostly for offering me emotional support and encouragement.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

SLU Spoken Language Understanding

HMIHY How May I Help You

ATT American Telephone and Telegraph

IVR Interactive Voice Response

NLU Natural Language Understanding

ASR Automatic Speech Recognition

DM Dialog Management (Dialog Policy)

NLG Natural Language Generation

TTS Text To Speech

NB Naive Bayes

MaxEnt Maximum Entropy

SVM Support Vector Machines

IDF Inverse Document Frequency

TF-IDF Term Frequency - Inverse Document Frequency

SVD Singular Value Decomposition

LSI Latent Semantic Indexing

GMM Gaussian Mixture Model

EM Expectation Maximization

Part I

CHAPTERS

1 INTRODUCTION

In this chapter we introduce Spoken Language Understanding and we focus on one application of it which is the Call Routing Systems. After that, we describe both the objective of this work and how it is organized here.

1.1 SPOKEN LANGUAGE UNDERSTANDING (SLU)

Spoken Language Understanding (SLU) is an emerging field in between speech and language processing, investigating human/machine and human/human communication by leveraging technologies from signal processing, pattern recognition, machine learning and artificial intelligence. SLU aims to extract the meaning of the speech utterances (spoken phrases) and automatically identify the intent of the user as expressed. The term SLU has largely been coined for understanding of human speech directed at machines and it is widely believed that the number of SLU applications will increase in the future since its applications are vast, from voice search in mobile devices to meeting summarization, attracting interest from both commercial and academic sectors. (several elements that are necessary in human-computer dialog are described in [33])

There are three categories of understanding systems [12]. The first one is about artificial intelligence systems that mimic understanding [Eliza, MIT 1966]. The second category contains systems rooted in artificial intelligence, which are successful for very limited domains, using deeper semantics. The third category is systems where understanding is reduced to a mostly statistical language processing problem and it is our main focus in this work.

In the last case, targeted speech understanding tasks are tried to be solved instead of the global machine understanding problem. In targeted speech understanding tasks the problem of understanding a user's utterance (we use the word utterance rather than sentence when referring to spoken language [21]) is reduced to the problem of extracting specific arguments in a given frame-based semantic representation. To do so, known classification methods are applied on the provided training dataset and comparative experiments are performed.

Today there are many targeted speech understanding systems and one them is AT&T's (Appendix A.1) "How May I Help You" system (HMIHY, Gorin, 1997 [4]). HMIHY is a call routing system whose

motivation was to build an alternative to traditional menu-driven IVR call center systems, moving the burden from the users to the system. This uderstanding task is framed as a classification problem.

1.2 CALL ROUTING SYSTEMS

Call routing is the task of getting callers to the right place in the call center, which could be the appropriate live agent or automated service. Natural language call routing lets callers describe the reason for their call in their own words, instead of presenting them with a list of menu options to select from using the telephone touch-tone keypad (details and comparisons can be found in [5]).

1.2.1 *Touch-tone menus*

Touch-tone menus implement call routing by having callers select from a list of options using their touch-tone keypad. If there are more than a few routing destinations, typically several touch-tone menus are arranged in hierarchical layers. Some commercially deployed speech-enabled IVRs replace touch-tone menus with speech-enabled menus, which allow callers to select an option by either speaking a number (≪ For . . ., press or say one≫) or a keyword (≪ Say weather, news, stocks,. . . ≫). Clearly, a speech-enabled IVR that mimics human call routing would be a substantial improvement over existing menu-based systems. Speech-enabled IVRs have made substantial headway at replacing standard touch-tone IVRs in call centers; for example, in airline flight information, banking services, and voice portals.

1.2.2 *Natural language call routing*

Natural language call routing is a speech-enabled IVR that employs an open-ended prompt for call routing. Several years ago speech recognition research developed the technology that makes natural language call routing feasible by combining speech recognition with natural language understanding technology. ≪How may I help you today?≫ is how most customers service representatives (they are mostly called agents) greet callers. This question essentially fulfills the task of directing callers to the right department or agent, which is termed call routing in call center terminology.

1.2.3 *Natural language call routing and Touch-tone menus comparison*

Many people interact with call centers on a daily basis, however, usability of most touch-tone IVRs (Interactive Voice Response systems) is dismal. Callers, having dealt with many IVRs that are difficult to use, dislike touch-tone IVRs and seek agent assistance at the first opportunity. Not surprisingly, more users describe the reason for calling in their own words than make a valid selection from a list of menu options. Clearly, one of the main problems of existing touch-tone systems is that callers either get confused with the menus or are too impatient to listen to all options, and then either do not respond at all or press $\ll 0 \gg$ to reach an operator. While the demise of touch-tone IVRs has been predicted, they are still widespread. Because of high agent costs, call center managers continue to seek automation with IVRs. A study was conducted in a call center of a large telecommunication service provider [5]. Results show that with natural language call routing, more callers respond to the main routing prompt, more callers are routed to a specific destination (instead of defaulting to a general operator who may have to transfer them), and more callers are routed to the correct agent. The survey data show that callers overwhelmingly prefer natural language call routing over standard touch-tone menus. Furthermore, natural language call routing can also deliver significant cost savings to call centers.

1.3 OBJECTIVE OF THIS WORK

The objective of the call routing task is to direct a customer's call to an appropriate destination within a call center or providing some simple information, such as current rates or the area code of a city, on the basis of some kind of interaction with the customer. Our goal is to create an algorithm for call-routing systems with this natural functionality. A caller receives a greeting and makes a request as if talking to a person. The algorithm's job is to recognize and understand what the user wants not in an ontological sense, but just sufficiently to properly direct the call. The necessity of a system like this is obvious. In current systems, such interaction is typically carried out via a touch-tone system with a rigid pre-determined navigational menu. Menu systems can be implemented using a touch-tone system (≪Press 1 if you want x, press 2 if you want y, ...≫), voice labels (≪Please say collect, calling card, ...≫), or a hybrid of the two (≪Press or say 1 if you want x, ... ≫). Each can be useful when the list of options is short and well understood by customers, but for certain tasks designers must resort to unwieldy hierarchical menus that can bore and frustrate users. The primary disadvantages of such navigating menus for

users are the time it takes to listen to all the options, the lack of naturalness and the difficulty of matching their goals to the given options. Since not all the users of call centers are familiar with such navigating menus, it turns out that more often than not, they are unable to cooperate with such systems. These problems are compounded by the necessity of descending a nested hierarchy of choices to zero in on a particular activity. Even requests with simple English phrasings may require users to navigate as many as four or five nested menus with four or five options each. We have developed an alternative to touch-tone menus that allows users to interact with a call router in natural spoken English dialogues just as they would with a human operator. In this call routing system, the prompt to the customer is deliberately general e.g. ≪How may I help you?≫. In contrast to the typical ≪Please say yes or no≫ prompts encountered in current voice dialogue systems, this prompt elicits a wide range of responses. These responses can be very different in length, ranging from single words to long responses that may be syntactically and semantically complex or ambiguous, and that may incorporate a large vocabulary. Here we describe a domain independent, automatically trained natural language call router for directing incoming calls in a call center. Our call router directs customer calls based on their response to an open-ended ≪How may I direct your call?≫ prompt. Routing behavior is trained from a corpus of transcribed and hand-routed calls and then carried out using several techniques. To examine human-human dialogue behavior, we analyzed a set of transcribed telephone calls involving customers interacting with human operators. In a typical dialogue interaction between a caller and a human operator, the operator responds to a caller request by either routing the call to an appropriate destination, or by querying the caller for further information to determine where the call should be routed. In our automatic call router, we develop the first option as well as a new option of sending the call to a human operator in situations where the router recognizes that it is beyond its capabilities to automatically handle the call. An almost similar task is the airline travel information system, ATIS [13]. It is also an intent determination task (as it is usually call in literature), but in this case, slot filling is needed as well.

1.4 ORGANIZATION OF THIS WORK

This work is organized in six chapters:

- Chapter 2:
  - `State of the art`: This chapter provides a brief, comprehensive though coverage of Spoken Language Understanding (SLU). More specifically, focusing on the call-routing

task, the main established approaches are covered. For each approach, we describe the algorithms used in order to provide the reader with a comprehensive view of the state of the art in this area.

- Chapter 3:
  - `Call-type classification`: In this chapter we introduce the term Salience and explain the benefits of using phrases instead of words during the classification procedure. Two classification algorithms are featured. They both extract salient phrases from the training dataset and use them in different ways so as to categorize a test call request.
- Chapter 4:
  - `GMM Classifier`: This chapter shows a different approach to the call-routing task. The classifier we build is GMM/ML and, since it results in good performance, we aim to provide details of each stage of the procedure. Consequently, we firstly describe the turning of the train dataset into vectors, analyzing the specific techniques used. Then, we cover extensively the methods that are applied (SVD, GMM) in order to accomplish our classification task. Much, important information and theory are from [19].
- Chapter 5:
  - `Datasets`: This chapter offers all the necessary statistical information about the dataset we used to implement this work and test the performance of all the algorithms. In particular, we used an American-english database. A class description of the dataset is also included.
- Chapter 6:
  - `Experiments`: This chapter focuses on the experiments we carried out in order to test the performance of all the algorithms we implemented. The extracted results as well as comparative tables and figures are offered for a deeper comprehension.
- Chapter 7:
  - `Conclusion`: This chapter summarizes and features the results and the conclusions from our work. In addition to this, possible future work is proposed.

2 STATE-OF-THE-ART

This chapter provides a brief, comprehensive though coverage of Spoken Language Understanding (SLU). More specifically, focusing on the call-routing task, the main established approaches are covered. For each approach, we describe the algorithms used in order to provide the reader with a comprehensive view of the state of the art in this area.

2.1 SPOKEN DIALOGUE SYSTEMS

A Spoken Dialog System is a dialog system delivered through voice. It denotes a wide range of systems, from weather information systems (for instance MIT Jupiter weather information) to complex problem solving, reasoning, applications. Some applications of spoken language systems are automatic call routing, answering questions about weather or sport, travel planning, tutoring systems, applications within games and many others.

2.1.1 *Architecture*

The objective of a spoken dialogue system is to let a human interact with a computer-based information system using speech as means of interaction. Because a spoken dialogue system can make errors and user requests can be incomplete or ambiguous, a spoken dialogue system must actively request for missing data and confirm information, resulting in a sequence of interactions between a user and a machine. This turn-based sequence of interactions between a user and a machine is called a dialogue. The figure bellow illustrates the architecture of a spoken language system. One example of a dialogue between a human (user) and a machine (system) could be a telephone directory assistance task. In this example, the user can ask for telephone numbers, fax numbers, and email addresses of persons, departments, and companies. After each user input, the system must select an appropriate dialogue action and response to the user in a cooperative way such that finally the user's request can be answered.

Spoken dialog systems divide the complex task of conversing with the user into more specific subtasks handled by specialized components: voice activity detection, speech recognition, natural language understanding, dialog management, natural language generation, and

speech synthesis. These components are usually organized in a pipeline as shown in Figure 1, where each component processes the result of the preceding one and sends its result to the next one. The following sections give a brief overview of each component and the typical issues they face.

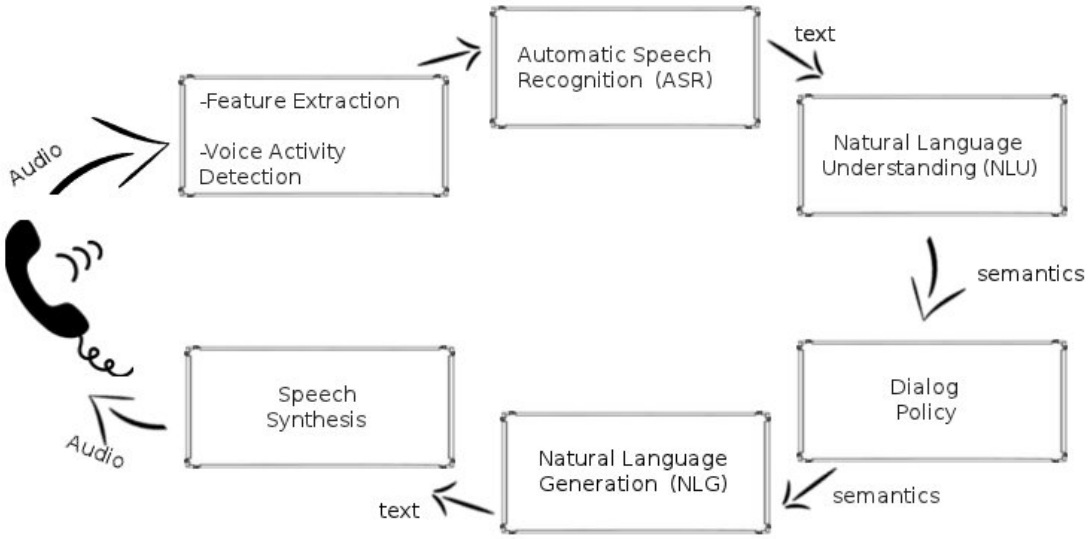


Figure 1: Spoken dialog systems chain

In Figure 1, a user's spoken utterance is taken and transformed into a textual hypothesis of the utterance. This hypothesis is parsed and a semantic representation of the utterance is generated, normally without looking at the dialogue context. This representation is then handled by the dialogue policy and generates a response. The language generation component then generates a surface representation of the utterance, often in some textual form, and passes it to a text-to-speech synthesis which generates the audio output to the user.

2.1.2 *Components*

Spoken dialog systems divide the complex task of conversing with the user into more specific subtasks handled by specialized components. These components are usually organized in a pipeline as

shown in the above figure, where each component processes the result of the preceding one and sends its result to the next one. These may be the most important components of most dialogue systems [32]:

1. Feature Extraction and Voice Activity Detection
2. Speech Recognition
3. Natural Language Understanding
4. Dialog Policy
5. Natural Language Generation
6. Speech Synthesis

The following sections give a brief overview of each component and the typical issues they face.

2.1.2.1 *Feature Extraction and Voice Activity Detection*

Voice activity detection (VAD) is the problem of detecting in the incoming audio signal when the user speaks and when he/she does not. Features from the audio signal are extracted and classified as speech or non-speech. This apparently easy problem can in fact be extremely hard to solve accurately in noisy conditions and has been the focus of much research, particularly in the signal processing community.

2.1.2.2 *Speech Recognition*

The automatic speech recognition module (ASR) takes the speech audio data segmented by the VAD and generates its word-level transcription. In addition, the generated hypothesis is sometimes annotated at the word- or utterance-level with confidence scores. ASR engines rely on three models an acoustic model, which describes the mapping between audio data and phonemes, a lexicon, which describes the mapping between phoneme sequences and words, and a language model, which describes the possible (or likely) sequences of words in a language. The acoustic model needs to be trained on a corpus of transcribed utterances. The lexicon can be either trained as a set of letter-to-sound rules from a corpus of words and their pronunciation, or, more often, it can be written by hand. The language model can be either a hand-written grammar, or a statistical language model trained on a corpus of in-domain data. Most ASR engines are designed to process full utterances, where the definition of an utterance depends on the provided grammar or the corpus on which the statistical LM was built, but usually corresponds to a phrase or sentence. However, they usually perform recognition incrementally, as the user is speaking, and therefore can provide partial recognition results at any time.

2.1.2.3 *Natural Language Understanding*

The natural language understanding module (NLU) takes the sequence of words output by the ASR and generates a semantic representation of it. NLU can be performed by providing a hand-written grammar that captures semantic relationships (either directly from the words, or via a syntactic analysis). Another approach to NLU is to train a statistical parser on a corpus of sentences annotated with their semantic representation. Most NLU modules assume that they are given a full utterance. Again the definition of an utterance varies and depends on the grammar or corpus on which the module was built.

2.1.2.4 *Dialog Policy*

The dialog policy or dialog manager (DM) takes the semantic representation of the user input generated by the NLU and outputs the semantic representation of the system's response. While there are many approaches to dialog management, the DM generally performs (at least) the following three tasks; interpreting user input in the current dialog context, updating the dialog context based on user input, generating relevant system responses. The DM also exploits knowledge about the domain and task at hand, which are usually provided by some back end module such as a database or an expert system.

2.1.2.5 *Natural Language Generation*

The natural language generation module (NLG) takes the semantic representation of the system response and outputs a natural language expression of it. Simple and common approaches to NLG include canned text when there is little variation in system prompts and templates. More advanced approaches have been proposed, either based on linguistic concepts such as discourse structure [Wilcock and Jokinen, 2003] or using statistical mapping between the semantic representation and the surface form [Oh and Rudnicky, 2000]. The NLG can optionally annotate the surface form with mark up tags destined to help speech synthesis using a speech synthesis mark up language such as SSML or JSAPI's. Such tags can indicate prosodic patterns such as rising or falling pitch, pauses, or emphasis.

2.1.2.6 *Speech Synthesis*

The speech synthesis, or text-to-speech module (TTS) takes the natural language output of the NLG (potentially augmented with mark up tags) and generates an audio waveform corresponding to its spoken version. The simplest way to perform TTS, and also the one that leads to the highest naturalness, is to use pre-recorded prompts. As for canned-text NLG, this can be used when there is little variation in the system prompts so that they can all be covered by a voice talent. General speech synthesis allows the system to say any text, even

potentially unplanned ones (which is necessary when the system retrieves variable information from, say, a web site). The synthesized utterance is played back to the user through the output audio device.

2.2 ALGORITHMS FOR CALL ROUTING

In this chapter we describe the state-of-the-art techniques which are applied in text classification and specifically in call-routing tasks. These techniques are useful benchmarks for the evaluation of the algorithms which are about to be proposed in this work. We focus on five widely used classification algorithms; n-gram Classifier, Naive Bayes (NB), Maximum Entropy (MaxEnt), Latent Semantic Intexing (LSI), Boosting and Support Vector machines (SVM). To fix notation, we denote a speech utterance with A, the word string that gave rise to it with $W = w_1 ... w_n$ and the class of the utterance with C(A). The word vocabulary is denoted with V and the class vocabulary with C. The corpus, split in training and test data, T and E, respectively, consists of samples s containing: utterance A, transcription W and utterance class C(A).

2.2.1 *n-gram Classifier*

Assume one builds an n-gram model $P\langle w_i \mid w_{i-1}, ..., w_{i-n+1}, C\rangle$ for each class C by pooling all the training transcriptions that have been labeled with class C.

In a one-pass scenario the decoder search for the most likely path will find:

$$(C(A), W) = argmax_{C,W} \log P\langle A \mid W\rangle + \log P\langle W \mid C\rangle + \log P(C) \quad (1)$$

In a two-pass scenario one builds a pooled n-gram language model $P\langle w_i \mid w_{i-1}, w_{i-n+1}\rangle$ from all the training transcriptions in addition to the class specific language models $P\langle . \mid C\rangle$. Each test utterance is then assigned a class by doing text classification on the 1-best recognition output using the pooled language model:

$$(C(A)) = argmax_C \log P\langle W \mid C\rangle + \log P(C) \quad (2)$$

$$W = argmax_W \log P\langle A \mid W\rangle + \log P(W) \quad (3)$$

Smoothing is a very important issue for this classifier. For estimating the n-gram models the recursive deleted interpolation scheme [18] between relative frequency estimates at different orders is used. For more details about N-gram models see [21][11].

2.2.2 *Naive Bayes (NB)*

One extremely popular supervised learning method we introduce is the multinomial Naive Bayes or multinomial NB model, a probabilistic learning method [9]. For any given event (W,C) in the training or test data, one constructs a binary valued feature vector listing the values each feature takes at this particular point:

$$\underline{f}(W) = (f_1(W), ..., f_F(W)) \tag{4}$$

Let $F = \{f_k, k = 1...F\}$ be the set of features chosen for building a particular model P(W,C). They are binary valued indicator functions $f(W) : V* \to \{0, 1\}$. For convenience we denote $\overline{f_i(W)} = 1 - f_i(W)$. We have explored using features of the form $f_w(W) = 1 \Leftrightarrow w \in W$ (1-gram features) or $f_{w_i,w_{i-1},...,w_{i-N+1}}(W) = 1 \Leftrightarrow (w_i, w_{i-1}, ..., w_{i-N+1}) \in W$ (n-gram features).

Assuming a NB model for the feature vector (see [25]) and the predicted variable (the utterance class), their joint probability is calculated as

$$P(\underline{f}(W), C) = \theta_C \prod_{k=1}^{F} \theta_{kC}^{f_k(W)} \overline{\theta}_{kC}^{\overline{f_k}(W)} \tag{5}$$

where θ_C and θ_{kC} are properly normalized. The class for a given utterance is assigned in two passes.

2.2.2.1 *Maximum Likelihood Parameter Estimation*

The parameters θ_C are estimated using maximum likelihood from the training data (relative frequencies). The parameters θ_{kC} are estimated using MAP smoothing:

$$\theta_{kC} = \frac{C(C, f_k) + \epsilon \cdot 1//2}{C(C) + \epsilon} \tag{6}$$

2.2.2.2 *Conditional Maximum Likelihood Parameter Estimation*

Another option for training the parameters of the model that is expected to be better correlated with the CER is to maximize the conditional likelihood of the training data

$$\sum \tilde{P}(W, C) \log P\langle C \mid W; \underline{\theta}\rangle \tag{7}$$

where P denotes the empirical distribution over the training set.

We have used the rational function growth transform (RFGT) algorithm described in [14] for estimating the parameters of the model

under the conditional maximum likelihood (CML) criterion. Due to the limited amount of space we do not go into the details of the estimation procedure.

It can be easily shown that eq. (5) can be rewritten as a log-linear model of the type that arises in ME probability modeling. Moreover, under the CML estimation criterion the same objective function is maximized for both NB and ME models.

2.2.3 *Maximum Entropy (MaxEnt)*

The MaxEnt method is a flexible statistical modeling framework that has been used in widely in many areas of natural language processing [6][10], one of which is call routing [26]. The MaxEnt allows the combination of multiple overlapping information sources [27],[6]. As described in [2][3], a ME classifier selects a conditional distribution $P(C \mid W)$ with maximum conditional entropy $H(C \mid W)$ from a family of distributions which satisfy the set of constraints:

$$\sum_{W,C} \tilde{P}(W,C) \cdot f_k(W,C) = \sum_{W,C} \tilde{P}(W) \cdot P\langle C \mid W\rangle \cdot f_k(W,C),\ \forall\, k\ =\ \overline{1,F} \tag{8}$$

where P denotes the empirical distribution over the training set.

Smoothing is extremely important for improving the classification accuracy. As shown in [7] ME models can be smoothed using a Gaussian prior on the feature weights and $\underline{\lambda}*$ can be selected using the maximum a posteriori (MAP) criterion. A modified version of improved iterative scaling (IIS) (as presented in [2]) can be used to find $\underline{\lambda}*$ under MAP:

$$\underline{\lambda}\ =\ \mathrm{argmax}_{\underline{\lambda}} \sum_{W,C} \tilde{P}(W,C) \cdot \log P\langle C \mid W;\underline{\lambda}\rangle - \frac{1}{2(T)} \cdot \sum_{k=0}^{F} \frac{\lambda_k^2}{\sigma_k^2} \tag{9}$$

$$P\langle C \mid W;\underline{\lambda}\rangle\ =\ Z(W;\underline{\lambda})^{-1} \cdot \exp\left(\sum_{k=0}^{F} \lambda_{kC} f_k(W,C)\right) \tag{10}$$

where σ_k^2 represent the variance parameters of the Gaussian prior and $|T|$ is the size of training set.

2.2.4 *Boosting*

Boosting is an iterative method for improving the accuracy of any given learning algorithm. The premise of Boosting is to produce a very accurate prediction rule by combining moderately inaccurate (weak) rules. The algorithm operates by learning a weak rule at each iteration so as to minimize the training error rate. A specific implementation of the Boosting is AdaBoost is described in [30]. Boosting has been applied to a number of natural language processing tasks in the past and call routing is one of them [26].

2.2.5 *Support Vector Machines (SVM)*

The final classification technique used for the call routing problem is Support Vector Machines [26]. SVMs are derived from the theory of structural risk minimization [34][9]. SVMs learn the boundaries between samples of the classes by mapping these sample points into a higher dimensional space. In the high dimensional space a hyperplane separating these regions is found by maximizing the margin between closest sample points belonging to competing classes. Much of the flexibility and classification power of SVM's resides in the choice of kernel. Some of the commonly used kernels are linear, polynomial and radial basis functions.

2.2.6 *Vector-based Classifier*

In vector-based information retrieval, the database contains a large collection of documents, each of which is represented as a vector in n-dimensional space. Given a query, a query vector is computed and compared to the existing document vectors, and those documents whose vectors are similar to the query vector are returned.

We apply this technique to call routing by treating each destination as a document, and representing the destination as a vector in n-dimensional space. Given a caller request, an n-dimensional request vector is computed. The similarity between the request vector and each destination vector is then computed and those destinations which are close to the request vector are then selected as the candidate destinations. In order to carry out call routing with the aforementioned vector representation, three issues must be addressed. First, we must determine the vector representation for each destination within the call center. Once computed, these destination vectors should remain constant as long as the organization of the call center remains unchanged. Second, we must determine how a caller request will be mapped to the same vector space for comparison with the destina-

tion vectors. Finally, we must decide how the similarity between the request vector and each destination vector will be measured.

This vector approach to call-routing is based on forming a matrix w using the transcriptions of the queries available to train the system. We assume that each of these has been labeled by an expert with the correct route. The rows of w correspond to different words (or sequences of words) in the vocabulary, and the columns to either different routes or different queries. To route a new query, it is first represented as an additional column vector of w and then matched to the other column vectors in w. Note that this approach ignores word order in queries.

2.2.6.1 *Cosine Similarity*

General information about Cosine Similarity

Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0^o is 1, and it is less than 1 for any other angle. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a Cosine similarity of 1, two vectors at 90^o have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1].

Note that these bounds apply for any number of dimensions, and Cosine similarity is most commonly used in high-dimensional positive spaces. For example, in Information Retrieval and text mining, each term is notionally assigned a different dimension and a document is characterized by a vector where the value of each dimension corresponds to the number of times that term appears in the document. Cosine similarity then gives a useful measure of how similar two documents are likely to be in terms of their subject matter. The technique is also used to measure cohesion within clusters in the field of data mining. It is important to note, however, that this is not a proper distance metric as it does not have the triangle inequality property and it violates the coincidence axiom.

For text matching, the attribute vectors A and B in cos(A,B) are usually the term frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison. In the case of Information Retrieval, the cosine similarity

of two documents will range from 0 to 1, since the term frequencies (tf-idf weights) cannot be negative. The angle between two term frequency vectors cannot be greater than 90^o.

Cosine Similarity for Call Classification

Once the pseudo-document vector representing the caller request is computed, we measure the similarity between the pseudo-document vector and each document vector in V,S. There are a number of ways one may measure the similarity between two vectors, such as using the cosine score between the vectors, the Euclidean distance between the vectors, the Manhattan distance between the vectors, etc.

We follow the standard technique adopted in the information retrieval community and select the cosine score as the basis for our similarity measure. The cosine score between two n-dimensional vectors x and y is given as follows [19]:

$$\cos(x, y) = \frac{xy^T}{\sqrt{\left(\sum_{1 \leqslant i \leqslant n} x^2\right)\left(\sum_{1 \leqslant i \leqslant n} y_i^2\right)}} \tag{11}$$

Using cosine reduces the contribution of each vector to its angle by normalizing for length. Thus the key in maximizing cosine between two vectors is to have them point in the same direction. However, although the raw vector cosine scores give some indication of the closeness of a request to a destination, we noted that the absolute value of such closeness does not translate directly into the likelihood for correct routing. Instead, some destinations may require a higher cosine value, i.e., a closer degree of similarity, than others in order for a request to be correctly associated with those destinations. Thus, in order to select a unique threshold for candidate destination selection that can be appropriately applied across destinations, we transform the cosine score for each destination using a sigmoid function specifically fitted for that destination to obtain a confidence score that represents the router's confidence that the call should be routed to that destination.

From each call in the training data, we generate, for each destination, a cosine value/routing value pair, where the cosine value is that between the destination vector and the request vector, and the routing value is 1 if the call was routed to that destination in the training data and 0 otherwise. Thus, for each destination, we have a set of cosine value/routing value pairs equal to the number of calls in the training data. The subset of these value pairs whose routing value is 1 will be

equal to the number of calls routed to that destination in the training set. Then, for each destination, we used the least squared error method in fitting a sigmoid function, $1/\left(1+e^{-(ax+b)}\right)$ to the set of cosine/routing value pairs. Assuming d_a and d_b are the coefficients of the fitted sigmoid function for destination , we have the following confidence function for a destination d and cosine value x:

$$Conf(d_a, d_b, x) = 1/\left(1+e^{-(d_a x+d_b)}\right) \tag{12}$$

Thus the score given a request and a destination, where d is the vector corresponding to destination, and r is the vector corresponding to the request is.

2.2.6.2 *Latent Semantic Indexing (LSI)*

A known technique for Text Classification [28] is Latent semantic indexing (LSI) [10][9][31][28]. LSI is an indexing and retrieval method that uses a mathematical technique called singular value decomposition (see Chapter 4) to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings. A key feature of LSI is its ability to extract the conceptual content of a body of text by establishing associations between those terms that occur in similar contexts.

After the approximation of a term-document matrix W by one of lower rank using the SVD, the low-rank approximation to W yields a new representation for each document in the collection. We will cast queries into this low-rank representation as well, enabling us to compute query-document similarity scores in this low-rank representation. This process is known as latent semantic indexing (generally abbreviated LSI).

The representation of documents by LSI is economical; each document and term need be represented only by something on the order of 50 to 150 values. The storage requirements for a large document collection can be reduced because much of the redundancy in the characterization of documents by terms is removed in the representation. Offsetting the storage advantage is the fact that the only way documents can be retrieved is by an exhaustive comparison of a query vector against all stored document vectors. Since search algorithms in high dimensional space are not very efficient on serial computers, this may detract from the desirability of the method for very large collections.

The latent semantic indexing method that we have discussed, and in particular the singular-value decomposition technique, are capable of improving the way in which we deal with the problem of multiple terms referring to the same object. They replace individual terms as the descriptors of documents by independent "artificial concepts" that can be specified by any one of several terms (or documents) or combinations thereof. In this way relevant documents that do not contain the terms of the query, or whose contained terms are qualified by other terms in the query or document but not both, can be properly characterized and identified. The method yields a retrieval scheme in which documents are ordered continuously by similarity to the query, so that a threshold can be set depending on the desires and resources of the user and service.

2.3 NOVELTY OF THIS WORK

In this work we tried to find a way to classify our data using a simple algorithm (simpler than MaxEnt or SVM) with good results as well. Firstly, we will experiment with the feature extraction. We will try to extract the features using a combination of 'mutual information' and 'salience', which are described in Chapter 3. All relevant works which deal with the call routing task use salience for feature extraction, but then they use grammars for automated parsing. In order to avoid a rule-based classifier, we will describe a less complex classification technique, creating a simple decision tree. Furthermore, we will try to build a GMM classifier and test how well this method can be applied to our task.

3

CALL-TYPE CLASSIFICATION

This chapter introduces the term 'salience' and explains the benefits of using phrases instead of words during the classification procedure. Two classification algorithms are featured. They both extract salient phrases from the training dataset and use them in different ways so as to categorize a test call request.

3.1 DATA PROCESSING - PHRASES PREFERRED INSTEAD OF WORDS

Traditionally, standard n-gram language models for speech recognition implicitly assume words as the basic lexical unit. However, the motivation for choosing optimal longer units for language modeling is threefold. First, not all languages have a predefined unity, such as the word (e.g. the Chinese language). Second, many word tuples (phrases) are recurrent in the language and can be thought as a single lexical entry (e.g. by and large, I would like to, United States of America, etc..). Third, the conditional probability $P\langle w_i \mid w_{i-n+1}, ..., w_{i-1}\rangle$ can benefit greatly by using variable length units to capture long spanning dependencies, for any given order n of the model. The information in the following subsections can be found in [4].

3.1.1 *Phrase*

An arbitrary continuous word sequence in the training transcriptions is called a phrase. All phrases can be obtained by decomposing the transcriptions into n-tuple word sequences. Namely, each phrase is a substring of a sentence. The number of words in a phrase is constrained to three or less in this experiment.

3.1.2 *Fragment*

The phrases having higher frequency than some threshold are selected as candidates. The candidate phrases are regarded as units for generating the Fragments. Each Fragment is acquired via the clustering of candidate phrases based on their similarity and is represented as a conventional finite-state machine.

3.1.3 *Salience*

Given the representation meaning via network association vectors, then a semantic distortion measure can be introduced between such vectors. A null word for a device is one whose network associations are all zero. The salience of a word for that device can then be defined via its distortion from the null word, thus providing a "norm" on the network association vectors. We now focus our attention on a single-layer information-theoretic network and quantify these intuitions. In this case,the network associations a word comprise a of vector of mutual informations between that word and the various machine actions. There are many possible distance measures that one could explore, but it is advantageous to exploit the information-theoretic nature of the vectors. Given a word, u, denote its network association vector as $\langle I(u, c_k)\rangle$ where $\langle ...\rangle$ denotes a vector whose k^{th} component is $I(u, c_k)$.

We define a semantic distortion measure $d_m(u_1, u_2)$ between two words u_1 and u_2 via first computing the difference between these vectors, then converting that difference into a scalar via projection onto some vector $\bar{u}$. Denote $\bar{w}_1 = \langle I(u_1, c_k)\rangle$ and $\bar{w}_2 = \langle I(u_2, c_k)\rangle$ then define

$$d_m(u_1, u_2) = (\bar{w}_1 - \bar{w}_2) \cdot \bar{u} \tag{13}$$

In particular, if we select $\bar{u}$ to be the vector of a posteriori probabilities $\langle P\langle c_k \mid u_1\rangle\rangle$, then

$$d_m(u_1, u_2) = \langle I(u_1, c_k) - I(u_2, c_k)\rangle \cdot \langle P\langle c_k \mid u_1\rangle\rangle \tag{14}$$

In addition to its geometric interpretation the scalar projection of a vector-difference, this distortion also has an information-theoretic interpretation. It can be easily seen that eq. (14) is equivalent to the Kullback-Leibler distance (a.k.a. relative entropy) between the a posteriori distributions $\langle P\langle c_k \mid u_1\rangle\rangle$ and $\langle P\langle c_k \mid u_2\rangle\rangle$ (Cover Thomas,1991). That is, the semantic distortion between the two words is equivalent to the distance between the distributions that they induce the network's on periphery.

Recall that a null word, u_{null}, is one whose association vector is all zeros. The salience of a word for a given device is defined as its semantic distortion from:

$$sal(u) = d_m(u, u_{null}) = \sum_{k=1}^{k} P\langle c_k \mid u\rangle I(u, c_k) \tag{15}$$

It was shown by Blachman that this is the unique non-negative measure of how much information a value of one random variable provides about a second one (Blachman,1968). That is, denoting random the variable of machine actions by C, then sal(u) is a measure of how much information the word u provides about C. Thus, salience provides an information-theoretic measure of how meaningful a word is for a particular device.

3.1.4 *Salient phrase fragments*

We need to find the most important and meaningful phrase fragments in the training data. We search the space of phrase fragments, guided by two criteria as referred in [15]. First, within the language channel, a word pair v_1v_2 is considered as a candidate unit if it has high mutual information:

$$I(v_1, v_2) = \log_2[P\langle v_2 \mid v_1\rangle / P(v_2)] \tag{16}$$

This measure can be composed to recursively construct longer units, by computing I(f,v) where f is a word-pair or larger fragment. We then introduce an additional between-channel criterion, which is that a fragment should have high information content for the call-action channel. Where f is a fragment and c_k is the set of call-actions, denote its salience by

$$I(v_1, v_2) = \log_2[P\langle v_2 \mid v_1\rangle / P(v_2)] \tag{17}$$

We perform a breadth-first search on the set of phrases, up to length four (an arbitrary cut-off point), pruning it by these two criteria: one defined wholly within the language channel, the other defined via the fragment's extra-linguistic associations. The peak of the a posteriori distribution $P(c_k \mid f)$ is denoted P_{max}. When Pmax is between 0.5 and 0.9, then the fragment is only moderately indicative of that call-type. When P_{max} is low (<0.5), then it is a background

fragment not associated with any particular call-type (for theoretical properties and more details see [16]).

Consequently, we sum up the procedure of the selection of the salient phrase fragments for a specific class ck in the following steps:

1. Find all possible phrase fragments f from the sentences belong to c_k
2. For each f, compute the information $I(f, c)$
3. If $I(f, c)$ is high, it is considered as a candidate
4. For each candidate we compute the a posteriori distribution $P(c_k \mid f)$:
   - If $P(c_k \mid f) < 0.5$, then f is not associated with c_k
   - If $0.5 \leqslant P(c_k \mid f) \leqslant 0.9$, then f is moderately indicative of c_k
   - If $P(c_k \mid f) > 0.9$, then f is indicative of c_k

3.1.5 *Statistical and rule-based approaches for spoken language understanding*

Now, after defining the term 'salient phrase fragment' we need to decide on how to use these fragments. Our choices are to apply either statistical pattern recognition or grammar-based parsing [36].

Statistical classifiers are very robust, can be easily trained and require little supervision during learning but they often suffer from poor generalization when data is insufficient. Grammar-based robust parsers are expressive and portable, can model the language in granularity, and are easy to modify by hand to adapt to new language usages. Although grammars are learnable, they often require more supervision. While they can yield an accurate and detailed analysis when a spoken utterance is covered by the grammar, they are less robust for those sentences not covered, even with robust understanding techniques. Because of this, the statistical classifiers are often used for broad and shallow understanding, and robust parsers are frequently used for narrow and deep understanding in a specific domain, where grammars can be crafted carefully to cover as many usages in the domain as possible.

In this work, we are interested in creating a robust classifier, that can be easily trained. Moreover, the goal is to manage to build an

algorithm which requires no, or at least little supervision. As a consequence, we did not prefer to make a rule-based classifier for sake of simplicity and robustness (more information about rule-based classifiers using grammar fragment acquisition in [22]).

3.2 CALL-TYPE CLASSIFICATION

By understanding spoken language, we mean, here, that we need to classify and automatically route users' calls, based on the meaning of the user's response. To achieve this we implement automatic algorithms for selecting phrases from a training corpus in order to enhance the prediction power of the standard word n-gram. From the speech recognizer output we recognize and exploit automatically-acquired salient phrase fragments to make a call-type classification. The typical approaches to the problem of topic classification are word and concept spotting.

Although these techniques work quite well for small applications, they do not scale up to large tasks and are limited in scope. On the other hand, we view this problem as understanding speech by taking into account the information conveyed by the whole utterance, with the ultimate goal of building automatically trained language models integrating both recognition and understanding.

The extraction and application of salient phrase fragments allows us to classify incoming calls using units that are larger (and usually less ambiguous) than individual words but without the need to parse an entire sentence. Here are two different ways to use the detected salient phrase fragments in order to classify an utterance.

3.2.1 *Peak-of-fragments classifier*

As referred in [20], we need to infer a call-type from the fragments detected within the speech. More generally, we would like to assign a ranking to the call-types. A simple method for doing this involves finding, for each call type, the highest posterior probability attributed to it by any detected fragment: $s(k) = maxP(c_k \in C \mid f_i \in F_t)$, k = 1,..., K where { fi } are the detected fragments for the utterance. The rank-1 decision for this utterance is then the call type.

3.2.2 *Tree classifier*

This classification proposal is a different version of the Neural network classifier in [20]. Typically an utterance contains several detected fragments and it is advantageous to combine the evidence from these when arriving at a classification. One method for doing this is as

follows. The detected fragments form a lattice which is parsed, for each call type separately, to find the path generating the highest cumulative score, summing the posterior probability for that call-type along the path. For a start, we need to read all training data. Using the labeled data we extract the salient phase fragments. We choose these fragments to consist of two, three or four words each. Of course not all fragments of this word-length can possibly be considered as salient. We only keep these ones which result in a probability of a specific class given the fragment, which is higher that a specific threshold. Unavoidably, some salient phase fragments are subfragments of other salient phase fragments of greater number of words. For each class, different groups of salient phase fragments are evaluated, despite the fact that this does not mean that a specific salient phase fragment cannot be included in the group of salient phase fragments of more than one class. The only different thing about these salient phase fragments, which exist in several class groups, is that they will obviously result in different probability of a specific class given this fragment. The next step is to check each test call request thoroughly, for the existence of at least one salient phase fragment. We need to find a way to estimate the probability of each class given this sentence, which in our case is the probability of each class given the found salient phase fragments in the test sentence. We suppose that the probability of a class of which no salient phase fragments appear in our test request, is zero. So we focus our attention on estimating the probability of those classes whose salient phase fragment exist in the sentence we examine. The problem we confront is that we need to prevent our algorithm from taking two or more inappropriate salient phase fragments into account. Two or more salient phase fragments are considered to be inappropriate when the one 'veils' an other. By 'veil', we mean that two or more phrases share the same words in the sentence. For instance, let the test request be "I wanna make a collect call" and the salient phrase fragments <wanna make>, <make a collect>, <make a collect call> and <a collect call>. We cannot just choose to use the 'bigger' one, meaning the fragment which consists of the greater number of words, since a 'smaller' fragment could be more salient for some class. Moreover, a 'smaller' one may finally be a more salient when combined with an other one. A more detailed example follows to give a clearer explanation. Supposing the sentence "I wanna make a collect call" and the salient phrase fragments detected for a specific class <wanna make>, <make a collect>, <make a collect call>, <a collect call> we can finally consider that the initial sentence can be represented by either the fragment <make a collect call> or the combination of the fragments <wanna make> and <a collect call>. From these two alternatives the more preferable option is the one which leads in higher cumulative score, summing the posterior probability for that class given the specific fragment multiplied

by the salience of this fragment. Let this score be called the weight of the option. Unfortunately, things tend to be more complicated considering that this procedure needs to be done for 'bigger' sentences, where more salient phrase fragments may be detected and ,of course, due to the fact that we have many classes, the complexity is that we have to compute a score for each possible combination of the fragments, for each class separately. To confront this complex problem, we create a tree. The tree is a structure which is really helpful in order to organize the important information we extract from a sentence. Owing to the fact that the important information defer from one class to another, we create a single tree for each class. The idea is simple. For each class the test request is scanned and the salient phrase fragments are extracted. From these fragments, all possible combinations are taken into account in order to create a path. Each path is a branch of the tree. Let's see an example. Let the sentence be 'a b c d e f g' and we suppose that there are two classes, class 1 and class 2. For class 1 we have found that the salient phrase fragments existing in our sentence are <a b>, <a b c>, <b c>, <e f>, <e f g> and for class 2 <a b c>, <b c>, <e f g>. The trees that these fragments give for each class are depicted in the following figure.

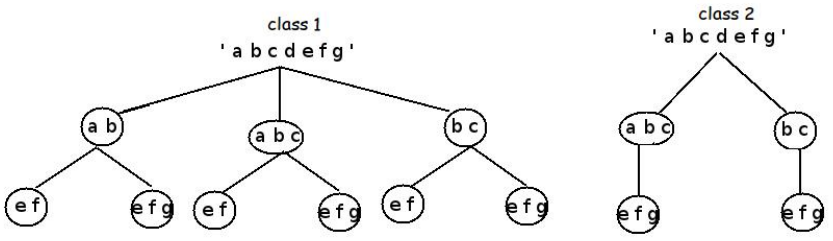


Figure 2: Example tree representation of a phrase

As it can easily be observed, the diverse salient fragments detected for each class result in a totally different tree. The main concept is that for each tree, we calculate the weight of each path which leads from the root to a leaf. Consequently, the tree constructed for class 1 includes six paths, whereas the tree constructed for class 2 has two paths. This means that the tree for class 1 has six weights, one for each path, and the tree for class 2 has two weights. For each class we only choose the path which has the highest weight and let's call it's weight, 'top score of the tree' (tst). Now, we have tst1 for the top score of the tree about class 1 and respectively tst2 for class 2. So, the final classification is achieved by comparing tst 1 and tst2. Supposing tst2 is higher that tst1, we end up to conclude that the most possible classification decision that can be made is class 2. The deduction of this example is that the class with the most salient phrase fragments detected in the test sentence is not necessarily the prevalent

one. However, in this example we assumed the existence of salient phrases.

One crux which can be faced is the possible absence of any salient phrase fragment in the sentence. In view to extract some phrase fragments which can be used to structure a tree for each class, the following idea is implemented. For each possible substring of the sentence, a similar fragment is searched in the list of the salient phrase fragments of each class. In order to exploit the similarity between fragments, we could use a Levenshtein string distance measure, in which the insertion, deletion and substitution penalties are weighted by the salience of the respective words. This has the effect of penalizing salient errors (such as substitution of 'collect' for 'credit') more than non-salient errors (such as substitution of 'this' for 'the'). However, fragments that are similar as strings can have different semantics, e.g. the fragments 'need a credit' and 'a credit card' indicate a billing credit request and credit card payment respectively. It would be undesirable for these to enter the same cluster. In assessing this we must allow for the variability attributable to small samples. We therefore use a measure of semantic distortion which is described in [20] (see also Appendix A.4 for the evaluation of semantic distortion). This is a weighted mean-square error between the estimated posterior distributions. We chose not to use the Kullback-Leiber measure because it is important to take the small sample sizes into account, and using the mean-square measure enables this. Note however that this measure does not obey the triangle inequality and so is not a true measure of distance. The evaluation of the semantic distortion in terms of actual occurrence counts is given in the Appendix. Finally, after computing the semantic distortion between a substring of the sentence and each salient phrase fragment of each class, the substring is substituted by the salient phrase which is semantically closer to the substring. So this salient phrase and the substring give the lowest semantic distortion. After all the possible substitutions of the substrings, a tree for each class is doable and the classification procedure is the same described above. In case of absence of similar phrases to substitute at least one substring of the sentence, the test request is classified in class 'other'.

Table 1: Steps of the Tree Classifier

Steps	Tree Classifier
Step 1	extract salient fragments from training data
Step 2	check each test call request for at least one salient phase fragment
Step 3	estimate the probability of each class given these salient phase fragments (by creating a tree for each class)
Step 4	for each tree, we calculate the weight of each path which leads from the root to a leaf
Step 5	we only choose the path which has the highest weight
Step 6	the class we choose is the one with the highest weighted path in her tree (if we can't build any trees, our decision is class 'other')

4

GMM CLASSIFIER

This chapter shows a different approach to the call-routing task. The classifier we build is GMM/ML and, since it results in good performance, we aim to provide details of each stage of the procedure. Consequently, we firstly describe the turning of the train dataset into matrices, analyzing the specific techniques used. Then, we cover extensively the methods that are applied (SVD, GMM) in order to accomplish our classification task. Many important information and theory are from [19].

4.1 TERM-DOCUMENT MATRIX CONSTRUCTION

First we need to construct an mxn term - document frequency matrix W, where m is the number of terms, n is the number of destinations, and $W[t, d]$ an element represents the number of times the term t occurred in calls to destination d. This number indicates the degree of association between term and destination, and our underlying assumption is that if a term occurred frequently in calls to a destination in our training corpus, then occurrence of that term in a caller's request indicates that the call should be routed to that destination.

4.1.1 *Term weighting*

The count W[t,d] we mention above is not suitable for direct use in routing an input query. By using the raw frequency counts as the elements of the matrix, more weight is given to terms that occurred more often in the training corpus than to those that occurred less frequently. For instance, a term such as 'call', which occurs frequently in calls to multiple destinations will have greater frequency counts than say, the term 'credit'. As a result, when the two vectors are combined, as will be done in the routing process, the term vector for 'call' contributes more to the combined vector than that for 'credit'. Consequently there is a great need to normalize the matrix W.

4.1.2 *Normalization techniques*

Various techniques for weighting the elements of w have been described [10]. Most of these techniques replace W[t,d] by the product of two weightings: one that takes account of the large variation in the number of occurrences of each term by applying some form of compression or normalization and another that accounts for the fact that terms that occur in only a few documents are more likely to be useful for routing purposes than terms that occur in many documents.

Some weighting schemes are Binary, Normal, GfIdf, Idf, Entropy. In this work the weighting schemes we used are Inverse document frequency and Entropy.

4.1.2.1 *Inverse-document frequency (IDF) weighting scheme*

As mentioned above, when two vectors are combined a term vector for a frequent term contributes more to the combined vector than that of a less frequent term. In order to balance the contribution of each term, the term-document frequency matrix is normalized so that each term vector is of unit length. Let B be the result of the normalizing the term-document frequency matrix, whose elements are given as follows:

$$B_{t,d} = \frac{W[t,d]}{\left(\sum_{1 \leqslant e \leqslant n} W[t,e]^2\right)^{1/2}} \tag{18}$$

It is a logical assumption that a term which only occurs in a few documents is more important in routing than a term which occurs in many documents. For instance, the term 'directory', which occurred mostly in calls requesting for directory services, should be more important in discriminating among destinations than 'call', which occurred in many destinations. Thus, we adopted the inverse-document frequency (IDF) weighting scheme (Sparck Jones, 1972) whereby a term is weighted inversely to the number of documents in which it occurs. This score is given by:

$$IDF(t) = \log_2 \frac{n}{d(t)} \tag{19}$$

where t is a term, n is the number of documents in the corpus, and d(t) is the number of documents containing the term t.

If t only occurred in one document, $IDF(t) = \log_2 n$

if t occurred in every document, $IDF(t) = \log_2 1 = 0$

Thus, using this weighting scheme, terms that occur in every document will be eliminated. We weight the matrix B by multiplying each row t by IDF(t) to arrive at the matrix C:

$$C_{t,d} = IDF(t) \cdot B_{t,d} \tag{20}$$

4.1.2.2 *Entropy weighting scheme*

One of our assumptions is that documents with low term entropy (high repetition of few words) are less useful to the searcher than documents with high term entropy (all words equally likely). This rests on the idea that the information conveyed in a document increases as the uncertainty of the content increases. When normalizing a column by entropy we therefore replace each raw frequency count in the column vector by:

$$e_j = 1 + \frac{\sum_{i=1}^{n} p_{ij} \log(p_{ij})}{\log n} \tag{21}$$

where

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{n} f_{ij}} \tag{22}$$

and f_{ij} is the frequency count of term i in document j.

Empirical studies with LSI report that the Log Entropy weighting functions l_{ij} work well, in practice, with many data sets, which is given by:

$$l_{ij} = e_j \log(1 + f_{ij}) \tag{23}$$

Once the term-document matrix is normalized, we are ready to provide a uniform representation of term and document vectors and to reduce the dimensionality of the document vectors by applying the singular-value decomposition method.

4.2 SINGULAR VALUE DECOMPOSITION (SVD)

A fundamental deficiency of current information retrieval methods is that the words searchers use often are not the same as those by which the information they seek has been indexed. There are actually two sides to the issue; we will call them broadly synonymy and polysemy.

We use synonymy in a very general sense to describe the fact that there are many ways to refer to the same object. Users in different contexts, or with different needs, knowledge, or linguistic habits will describe the same information using different terms. Indeed, we have found that the degree of variability in descriptive term usage is much greater than is commonly suspected. For example, two people choose the same main key word for a single well-known object less than 20% of the time. Comparably poor agreement has been reported in studies of interindexer consistency and in the generation of search terms. The prevalence of synonyms tends to decrease the "recall" performance of retrieval systems.

By polysemy we refer to the general fact that most words have more than one distinct meaning (homography). In different contexts or when used by different people the same term (e.g. "chip") takes on varying referential significance. Thus the use of a term in a search query does not necessarily mean that a document containing or labeled by the same term is of interest. Polysemy is one factor underlying poor "precision".

4.2.1 *SVD Basis*

We used the technique of singular value decomposition (SVD) [10][9] to find the lowest error representation of the matrix W in a compact sub-space. This can be seen as "smoothing" the term or document vectors [8]. The essential steps in the process are as follows:

- step 1: For SVD application we construct a term/document matrix w.
- step 2: SVD is applied to w so that $w = USV^T$ where U and V are orthonormal matrices (dimensions MxN and NxN respectively) and S is diagonal matrix of eigenvalues.
- step 3: S is inspected and the dimensions corresponding to only the top R eigenvalues are retained: the other dimensions are discarded. If R$\ll$N, this means that w is represented in a much reduced dimensionality.
- step 4: A query is pre-processed into a vector q.
- step 5: Then q is projected into the reduced subspace to become the vector $q' = q^T US^{-1}$
- step 6: The vector q′ can then be matched (using an appropriate criterion) in the subspace to the training-set vectors.

Table 2: Basic steps in SVD process

Steps	SVD application
Step 1	Construct a term/document matrix w
Step 2	SVD is applied to w so that $w = USV^T$
Step 3	The top R eigenvalues are retained,the other are discarded
Step 4	A query is pre-processed into a vector q.
Step 5	q is projected into the reduced subspace and $q' = q^T US^{-1}$
Step 6	The vector q′ can be matched in the subspace to the training-set vectors.

4.2.2 *SVD Analysis*

Using the method SVD we can find an approximation [8] $\overline{W}$ such that

$$W \simeq \overline{W} = USV^T$$

where $S = diag(s_1, ..., s_s)$
which contains s singular values sorted in descending order. U and

V are orthonormal matrices, which are called left and right singular matrices, respectively.

We have two important properties following the definition of $\overline{W}$:

1. $rank(\overline{W}) = s$
2. $\min_{r(A)=s} \| W - A \|_F^2 = \| W - \overline{W} \|_F^2 = s_{s+1}^2 + ... + s_m^2$, where $m = rank(W)$

In the decomposition, rows of matrix V could be viewed as a set of basis vectors of the vocabulary, and each of the words in our vocabulary could be obtained by some linear combination of them. Since US serves as the coefficients of the linear combinations, we could use the rows of US as the word vectors in the s-dimensional space.

In detail, for $W \simeq \overline{W} = USV^T$ the matrices U, V, and S must all be of full rank. The beauty of an SVD, however, is that it allows a simple strategy for optimal approximate fit using smaller matrices. If the singular values in S, are ordered by size, the first k largest may be kept and the remaining smaller ones set to zero. The product of the resulting matrices is a matrix W which is only approximately equal to W, and is of rank k. It can be shown that the new matrix X is the matrix of rank k, which is closest in the least squares sense to W. Since zeros were introduced into S', the representation can be simplified by deleting the zero rows and columns of S, to obtain a new diagonal matrix S', and then deleting the corresponding columns of U, and V, to obtain U' and V' respectively.

The result is a reduced model:

$$W_{reduced} = U'S'V'^{T}$$

which is the rank-k model with the best possible least-squares-fit to W. It is this reduced model, presented in Figure below, that we use to approximate our data.

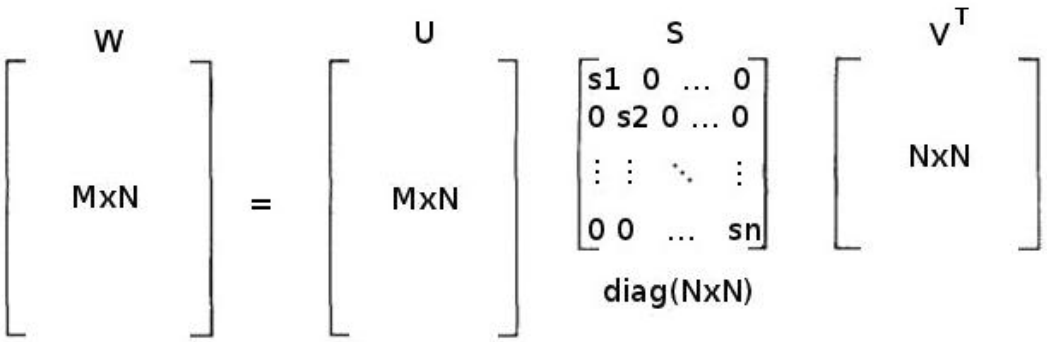


Figure 3: Singular Value Decomposition (SVD)

4.3 GAUSSIAN MIXTURE MODEL (GMM)

A Gaussian mixture model [23][24] is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

We used the scikit-learn tool in python which implements different classes to estimate Gaussian mixture models, that correspond to different estimation strategies. The class sklearn.mixture.GMM allows for easy evaluation of, sampling from, and maximum-likelihood estimation of the parameters of a GMM distribution. With this class we create an object GMM which implements the expectation maximization (EM) algorithm for fitting mixture-of-Gaussian models. It can also draw confidence ellipsoids for multivariate models, and compute the Bayesian Information Criterion to assess the number of clusters in the data. A GMM.fit method is provided that learns a Gaussian Mixture Model from train data. Given test data, it can assign to each sample the class of the Gaussian it mostly probably belong to using the GMM.predict method.

We have already mentioned the use expectation maximization (EM) algorithm. Expectation-maximization is a well-fundamented statisti-

cal algorithm to get around this problem by an iterative process. First one assumes random components (randomly centered on data points, learned from k-means, or even just normally distributed around the origin) and computes for each point a probability of being generated by each component of the model. Then, one tweaks the parameters to maximize the likelihood of the data given those assignments. Repeating this process is guaranteed to always converge to a local optimum.

4.3.1 *GMM Model Formulation*

A Gaussian distribution is unimodal. Sometimes, the data is multimodal. In such cases, one might want to use a more flexible distribution for modeling. One possibility is to use the Gaussian Mixture Model (GMM). A multivariate normal distribution or multivariate Gaussian distribution is a generalization of the one-dimensional Gaussian distribution into multiple dimensions [35].

$$N\langle x \mid \mu, \Sigma\rangle = \frac{1}{(2\pi)^{D/2}} \frac{1}{\mid \Sigma \mid^{1/2}} \exp\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\} \quad (24)$$

A mixture model can be regarded as a type of unsupervised learning or clustering. Mixture models provide a method of describing more complex propability distributions, by combining several probability distributions. Mixture models can also be used to cluster data. The Gaussian mixture distribution is given by the following equation

$$p(x) = \sum_{k=1}^{K} \pi_k N(x \mid \mu_k, \Sigma_k)$$

where

$\pi = \{\pi_j\}$ are the mixing coefficients that satisfy $\pi_j \leqslant 0$ and $\sum \pi_j = 1$

$\mu = \{\mu_j\}$ are mean parameters of the respective Gaussian distributions

$\Sigma = \{\Sigma_j\}$ are variance parameters of the respective Gaussian distributions

We will often use the shorthand θ to denote the set of parameters $\{\pi, \mu, \Sigma\}$. Given iid data $X = \{x_1, ..., x_m\}$ the log-likelihood of this model can be written as:

$$\log p\langle X \mid \theta \rangle = \sum_{i=1}^{m} \log \sum_{j=1}^{k} \pi_j N\langle x_i \mid \mu_j, \Sigma_j \rangle \tag{25}$$

Here the log acts on the sum and hence does not cancel the exp. Consequently, the log likelihood is a nasty function to optimize, and we cannot hope to get a nice closed form solution. Nevertheless, one can take derivatives of the log likelihood with respect to μ, Σ and π and set them to zero to obtain first order stationary conditions. For ease of notation define:

$$\gamma_{ij} = \frac{\pi_i N\langle x_i \mid \mu_j, \Sigma_j \rangle}{\sum_{j'} \pi_{j'} N\langle x_i \mid \mu_{j'}, \Sigma_{j'} \rangle} \tag{26}$$

Furthermore, assume that all the Σ_j are non-singular. Now we can take derivatives of $\log\{p\langle X \mid \theta \rangle\}$ with respect to μ_j and set them to zero. This yields:

$$0 = -\sum_{i=1}^{m} \gamma_{i,j} \Sigma_j^{-1} (x_i - \mu_j) \tag{27}$$

Multiplying by Σ_j and rearranging yields:

$$\mu_j = \frac{1}{m_j} \sum_{i=1}^{m} \gamma_{ij} x_i \tag{28}$$

where we define $m_j = \sum_{i=1}^{m} \gamma_{ij}$

Similarly by setting the derivative with respect to Σ_j to zero we obtain:

$$\Sigma_j = \frac{1}{m_j} \sum_{i=1}^{m} \gamma_{ij} \left(x_i - \mu_j\right) \left(x_i - \mu_j\right)^T \tag{29}$$

Finally, by using Lagrange multipliers to enforce the sum to one constraint we can maximize the log likelihood with respect to π by maximizing:

$$\log p\langle X \mid \theta\rangle \lambda \left(\sum_j \pi_j - 1 \right) \tag{30}$$

which gives

$$0 = \sum_{i=1}^{m} \frac{N\langle x_i \mid \mu_j, \Sigma_j\rangle}{\sum_{j'} \pi_{j'} N\langle x_i \mid \mu_{j'}, \Sigma_{j'}\rangle} + \lambda \tag{31}$$

If we multiply both sides by π_j and sum over j then we obtain $\lambda = -m$. Using this to eliminate λ, one obtains:

$$\pi_j = \frac{m_j}{m} \tag{32}$$

Although (eq.28), (eq.29), and (eq.32) do not give a closed-form solution to the problem of maximizing the log-likelihood (eq.25), they are at least intuitively appealing. Let us view γ_{ij} as the fraction of data point x_i "belonging" by the j-th Gaussian distribution. Under this interpretation, (eq.28) and (eq.29) are exactly the same as $\mu = \frac{1}{m} \sum_{i=1}^{m} x_i$ and $\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)(x_i - \mu)^T$ for data $X = \{x_1, ..., x_m\}$, except for the fact that the each data point xi only contributes a factor γ_{ij} towards computing μ_j and Σ_j.

Of course, the problem is that γ_{ij} depends on μ, Σ, and π, while μ, Σ, and π in turn depend on γ_{ij}. One way to break this chicken and egg situation is via the following simple iterative scheme:

- In Step 1 (which we will later call the E-Step) we use the current value of μ, Σ, and π to estimate γ_{ij} via (eq.26).
- In Step 2 (which we will later call the M-Step) we use the current value of γ_{ij} to estimate μ, Σ, and π via (eq.28), (eq.29), and (eq.32) respectively. Iterate until convergence.

Later we will show that this simple iterative scheme has close connections to a more general algorithm called the Expectation Maximization (EM) algorithm. Now we show that K-Means can be viewed as a limiting case of the above iterative scheme.

4.3.2 *Expectation Maximization (EM)*

We want to find a set of K multivariate Gaussian distributions, given a set of samples, that represent observed samples in a good way. The number of clusters, K, is given, so the parameters that are to be found are the means and covariances of the distributions.

An acknowledged and efficient method for finding the parameters of a GMM is to use Expectation Maximization (EM). The EM algorithm is an iterative refinement algorithm used for finding maximum likelihood estimates of parameters in probabilistic models. The likelihood is a measure for how good the data fits a given model, and is a function of the parameters of the statistical model.

If we assume that all the samples in the dataset are independent, then we can write the likelihood as,

$$\ell = \prod_n p\left(x_n\right) \tag{33}$$

and as we are using a mixture of gaussians as model, we can write

$$p\left(x_n\right) = \sum_k N\langle x_n \mid \mu_k, \Sigma_k\rangle p\left(k\right) \tag{34}$$

In contrast to the K-means algorithm, the EM algorithm for Gaussian Mixture does not assign each sample to only one cluster. Instead, it assigns each sample a set of weights representing the sample's probability of membership to each cluster. This can be expressed with a conditional probability $p\langle k \mid n\rangle$ given a sample n gives the probability of the sample being drawn a certain cluster k.

A responsibility matrix R with elements p_{nk} is used to hold the probabilities

$$pnk = p\langle k \mid n\rangle = \frac{p\langle x_n \mid k\rangle p\left(k\right)}{p\left(x_n\right)} = \frac{N\langle x_n \mid \mu_k, \Sigma_k\rangle p\left(k\right)}{p\left(x_n\right)} \tag{35}$$

Given the data and the model parameters μ_k and Σ_k we now can calculate the likeliness ℓ and the probabilities p_{nk}. This is the expectation (E) step in EM-algorithm.

In the maximization (M) step we estimate the mean, co-variances, and mixing coefficients $p(k)$. As each point has a probability of belonging to a cluster p_{nk} we have to weight each sample's contribution to the parameter with that factor.

The following equations are used to estimate the new set of model parameters:

$$\overline{\mu_k} = \frac{\sum_n p_{nk} x_n}{\sum_n p_{kn}} \tag{36}$$

$$\overline{\Sigma_k} = \frac{\sum_n (x_n - \overline{\mu_k}) \otimes (x_n - \overline{\mu_k})}{\sum_n p_{nk}} \tag{37}$$

$$\overline{p}(k) = \frac{1}{N} \sum_n p_{nk} \tag{38}$$

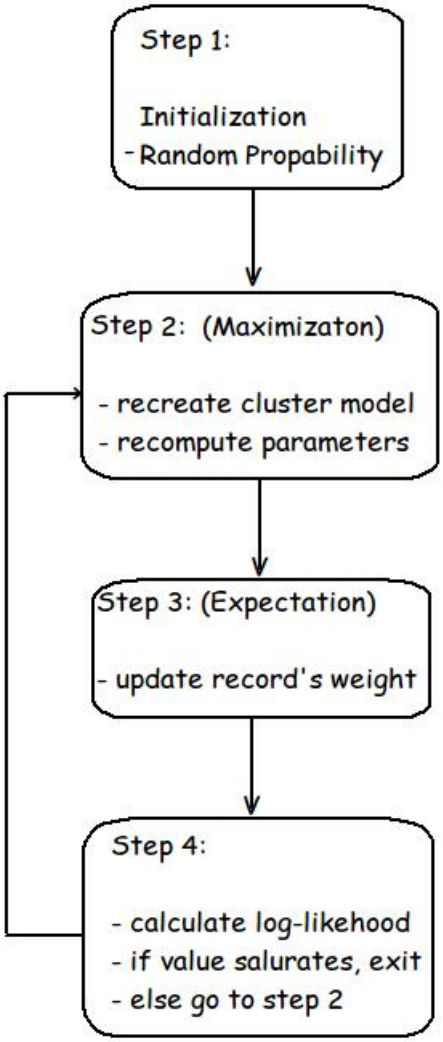


Figure 4: Expectation Maximization Algorithm (EM)

4.3.3 *GMM-Covariance matrix*

The covariance matrix, Equation 37, is normally a full matrix and when it's dimension is nxn the EM algorithm has to estimate nxn parameters, something that seems to be quite time-consuming and has high complexity.

Instead of this, there are three alternative forms of the covariance matrix which are also more appropriate than the full covariance matrix when we dial with sparse data. These extra three covariance types (except full) are:

- diagonal: all the elements of the covariance matrix apart from the diagonal are zero. Consequently, in a nxn matrix, only n parameters have to be estimated. In this way, we do not take the correlation among the variables into account.
- spherical: all the elements of the covariance matrix rather than the diagonal are zero. Moreover, all the diagonal elements are the same. Consequently, in a nxn matrix, only one parameter has to be estimated.
- tied: when there is a high number of components in a Gaussian mixture distribution we need more training data to estimate the parameters. One way to tackle this is to use the same Gaussian distributions to represent all the states of all the models, with only the mixture weights being state-specific. This type of model is called tied mixture (Bellegarda Nahamoo, 1990) because the parameters are tied across the different states.

5 DATASETS

This chapter offers all the necessary statistical information about the dataset we used to implement this work and test the performance of all the algorithms. In particular, we used an American-english database. A class description of the dataset is also included.

5.1 CORPUS ANALYSIS

To examine human-human dialogue behavior, we analyzed the set of transcribed telephone calls involving customers interacting with human operators.

Our data is a set of 6.522 training and 849 test transcribed telephone calls from an american company known as AT & T (American Telephone & Telegraph). We analyzed these calls along two dimensions: the semantics of caller requests and the dialogue actions for operator responses. In our corpus, all callers respond in English to an initial open-ended prompt of "How may I help you?". We classified user responses to this prompt into 14 specific classes based on the action requested by the caller and a 15th class called other, in which a call request should be classified in case none of the 14 classes is a match.

Table 3: Datasets Information

Dataset Name	Number of Training data	Number of Test data	Number of Classes
ATT	6.522	849	15

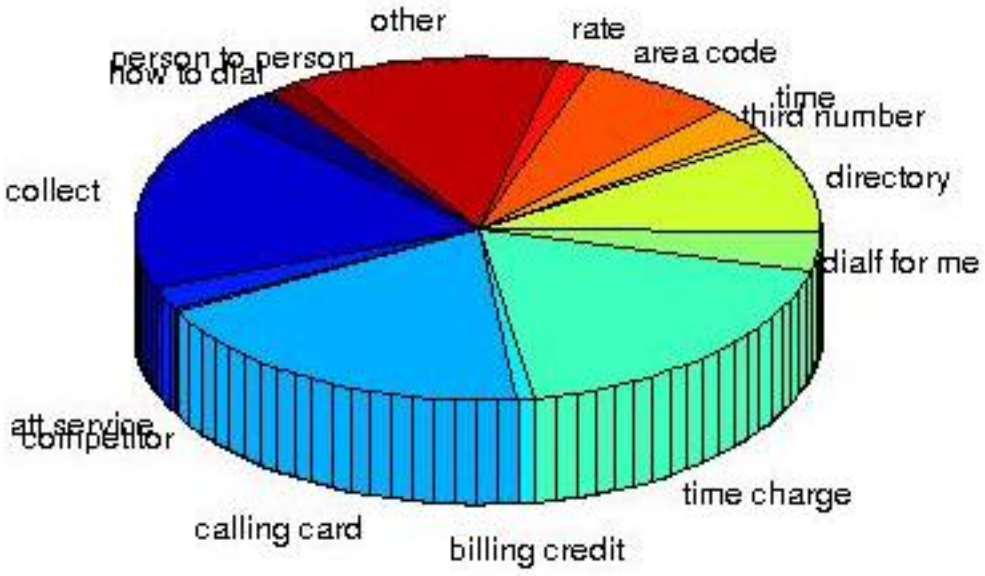


Figure 5: Pie representation of the classes for the ATT train data

5.1.1 *AT&T Dataset Classes*

Here, a brief description of each class is provided.

- Collect: this class includes all the calls of users whose request is to be connected to a phone number by the phone center and not pay for this call. In detail, a collect call is the procedure that a person calls another person and this connection is actually established only when the receiver accepts both the call and the charge of it. This mostly happens from payphones and it serves people who, having no money, want to make a call.

  Example 5.1.1.1 *I'd like to make a collect call.*

- Calling Card: in this class we classify calls which are related to the use of a calling or credit card. Specifically, users may wish to use a payphone without using any of their real money. So they have the ability to use a calling or credit card such as visa or american express in order to place their call. To do so, the callers have to request from the calling center to do that by telling their personal card number.

  Example 5.1.1.2 *I wanna place a call and charge it to my calling card number*

- Dial For Me: this is an operation offered by the phone center in order to assist users who cannot dial the phone number they

are calling to. It is really helpful for people with problems such as reduced vision or blindness, for instance, or for people who dial with a technical malfunction of their phone and thus are unable to dial themselves.

Example 5.1.1.3 *can you dial a call for me?*

- Directory: this operation serves people who call due to their need to get some directory assistance. In telecommunications, directory assistance or directory inquiries is a phone service used to find out a specific telephone number and/or address of a residence, business, or government entity.

  Example 5.1.1.4 *yes please I need information for a telephone number in Columbia south California*

- Third number:a third number call or third party call is an operator assisted telephone call that can be billed to the party other than the calling and called party. The operator calls the third number for the party to accept the charges before the call can proceed.

  Example 5.1.1.5 *yeah I like to bill this to a third number please*

- Time: this class includes all calls from users who simply requests the time. The user has the ability to ask and learn the time not only in the area where the call is placed from, but also the time in other places.

  Example 5.1.1.6 *yes could you tell me what's the time*

- Area Code:this class corresponds to the operation of providing code information by the phone center. This operation serves callers who need to know either an area code (the prefix routing code for a region) or a country code (country codes are necessary only when dialing telephone number in other countries and they are dialed before the national telephone number; by convention, international telephone numbers are indicated by prefixing the country code with a plus sign (+), which is meant to indicate that the subscriber must dial the international dialing prefix in the country from which the call is placed).

  Example 5.1.1.7 *could you give me the country code for the ukraine please*

- Billing Credit: the call requests which belong to this class are made by users who for some reason (they misdialed or they were connected to a wrong number) they want a refund. So, a customer can call and ask for a billing credit expecting that the money he/she was charged will be returned to him/her.

Example 5.1.1.8 *yes hi I called this number which was wrong I wanna get credit*

- Time Charge: the call charges are variable and are used to pay for the cost of the equipment to route a call from the caller's exchange to the recipient's exchange. These call charges can be calculated on a fixed per call basis, a variable basis depending on the time or distance of the call, or a combination of the two. Call charges can even vary at different times of the day. So, calls found in this class ask for information like this described above.

Example 5.1.1.9 *time and charges please*

- Competitor:in call requests of this class, users ask to be connected to another operator such as nynex (NYNEX Corporation was a telephone company that served five New England states as well as most of New York state, except the Rochester area, from 1984 through 1997) or MCI (MCI, is an American telecommunication corporation, currently a subsidiary of Verizon Communications, with its main office in Ashburn, Virginia) as shown in the following example.

Example 5.1.1.10 *yes can you connect me with a nynex operator please*

- Rate: in this class we find calls in which a user wants to find out what is the rate for a specific call they want to place.

Example 5.1.1.11 *yes I will be placing an international phone call I need to know the rate*

- ATT Service: in this class two types of call requests are classified. The first one includes calls by people who use the at&t telephone network or use other services such as at&t cable television and have a problem. The second one includes some other services provided by at&t. One such example is the service of International operator. International operator services (IOS) allow travelers to place an international call using a live telephone operator who speaks their language and accepts all forms of payment for connecting the call from anywhere in the world, to anywhere in the world. This specialty form of operator services uses collect call billing as well as credit card billing to allow callers to place calls without a calling card or without the correct change for the pay telephone.

Example 5.1.1.12 *yes may I be connected with customer service please*

- How To Dial: this class consists of users' calls who request to be guided on how to dial and make a specific type of call.

  Example 5.1.1.13 *I need to figure out how to make an international call*

- Person To Person: a person-to-person call is an operator-assisted call in which the calling party wants to speak to a specific party and not simply to anyone who answers. The caller is not charged for the call unless the requested party can be reached. This method was popular when telephone calls were relatively expensive.

  Example 5.1.1.14 *yes I'd like to make a person to person to edna zallis*

- Other: this class includes all these requests that are general, unobvious or meaningless, like

  Example 5.1.1.15 *hello*

  Example 5.1.1.16 *no I was just hanging up*

Table 4: AT&T Train Dataset Classes

Class Name	Number of Training data
collect	1141
calling card	1243
dial for me	1211
directory	260
third number	601
time	38
area code	185
billing credit	477
time charge	46
competitor	25
rate	112
att service	113
how to dial	148
person to person	112
other	807

Table 5: AT&T Test Dataset Classes

Class Name	Number of Test data
collect	115
calling card	173
dial for me	142
directory	44
third number	76
time	4
area code	21
billing credit	56
time charge	7
competitor	2
rate	13
att service	28
how to dial	18
person to person	21
other	129

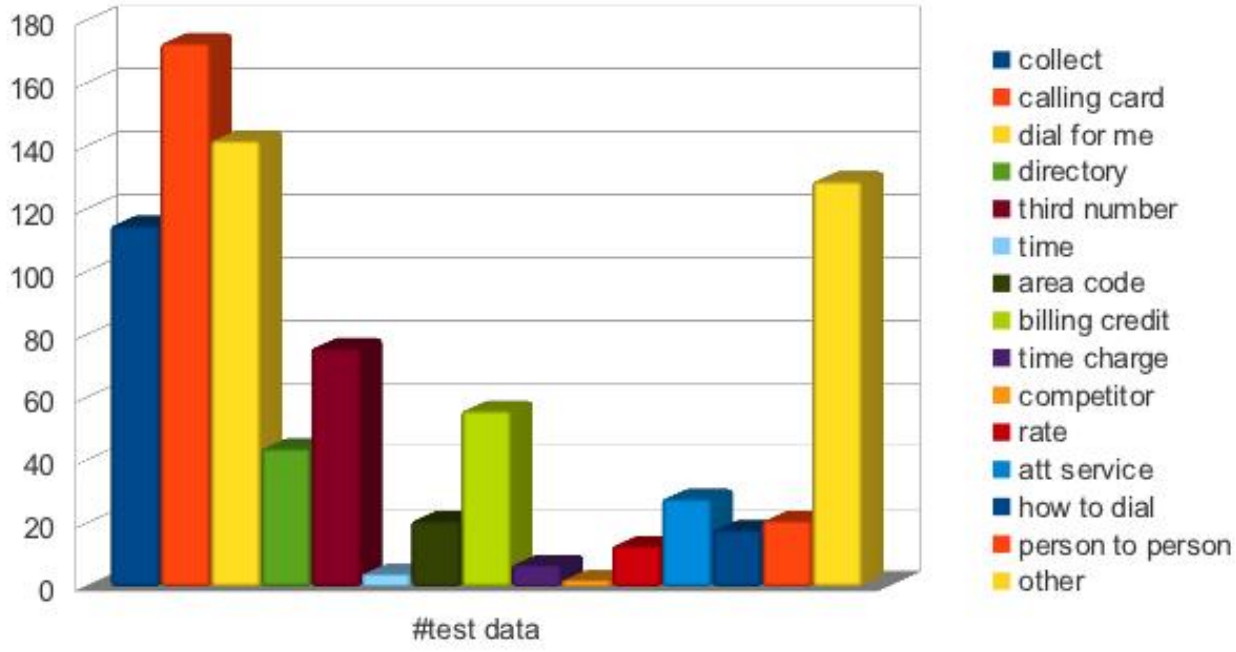


Figure 6: Number of test data in ATT dataset

5.2 PROBLEMS CONCERNING THE CLASSES

In our ATT dataset we can find some call requests which are from fluent speakers and even users who are familiar with these services. As a result, in these call requests, the intention of the user is really obvious. One such example is "collect call please". The problem is that in most cases, the users do not speak american-english fluently or they are not sure on how to ask what they want and thus there are many data samples which do not make sense or at least it is difficult even for a human to comprehend the purpose of that call. Finally, there are some situations which seem to be difficult to classify, due to the fact that the speaker demands the time, for instance, and then they change their minds and ask for something else.

One of the classes which seems to have the most difficult in understanding calls is dial for me. We can hopefully meet some simple requests like "can you dial a call for me?" in which the user clearly declares what is wanted to be done. However there are some more demanding and tough for classification cases such as "I'm having trouble dialing a number here from ayer brothers". Here user's intention is not really obvious, although as humans we can understand that since the caller cannot dial the number, he or she needs to get the phone center to dial the number.

One additional problem we dealt with (especially for the ATT database), is that as it can obviously be seen from Figure 5 (on page 46) and the Table 4 (on page 49), for some of the classes there are not sufficient training data. This is a factor which will definitely play an important role on the efficiency of our classification algorithm.

6 EXPERIMENTS

This chapter focuses on the experiments we carried out in order to test the performance of all the algorithms we implemented. The extracted results as well as comparative tables and figures are offered for a deeper comprehension.

6.1 EVALUATION MEASURES

The target of a classifier is to classify each test data in the correct class. So, to examine how successful our algorithm is, we calculate the accuracy of the classifier. As accuracy, we consider the number of the test data which are classified correctly over the total number of the test data. The formula which describes the accuracy of a classifier is:

$$Accuracy = \frac{\#Correctly\ classified\ test\ data}{\#Total\ test\ data} \cdot 100\ (\%) \quad (39)$$

When we want to evaluate our classifier per class, there are some further measures we can use (Dan Jurasky-NLP course). These measures are precision and recall. First, we define the terms tp, fp, tn, fn:

- tp (true positive): the number of the data which are classified correctly
- fp (false positive): the number of the data which are not classified correctly
- tn (true negative): the number of the data which should not be classified in a class and they weren't
- fn (false negative): the number of the data which should not be classified in a class but they were

Now we can define precision (Prec) and recall (Rec) as:

$$\text{Prec} = \frac{tp}{tp+fp}$$

$$\text{Rec} = \frac{tp}{tp+fn}$$

Precision can be seen as a measure of exactness or quality, whereas recall is a measure of completeness or quantity. In simple terms, high precision means that an algorithm returned substantially more relevant results than irrelevant, while high recall means that an algorithm returned most of the relevant results [1].

Since we have more than two classes we need to find a way to combine the precision and recall we get from each class into one measure, since it is often useful to have a single measure. There are two ways to combine multiple performance measures into one quantity:

- Macroaveraging: compute performance for each class, then average.
- Microaveraging: collect decisions for all classes, compute contingency table, evaluate.

When having evaluated the macroaveraged or microaveraged precision and recall, the combination of these two measures is the traditional F-measure or balanced F-score (F1 score), which is the harmonic mean of precision and recall:

$$F_1 = \frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}$$

6.2 BASELINE

In order to evaluate the results of our classifier, we need to use the results of the relevant algorithms as described in Chapter 2 and compare them with the results of this work in a following section. The baseline algorithms we focus on are Naive Bayes (NB) and Maximum Entropy (MaxEnt) owing to their wide use. For classifying our data with these algorithms we use the Standford Classifier (see Appendix A).

All the compared algorithms are applied on the same train and test dataset. Here is the accuracy of the mentioned baseline techniques applied to our two databases (more detailed results per class and per database can be found in Figure 7 and Figure 8):

Table 6: Baseline Results (%)

Datase Name	NB	MaxEnt
ATT	18.14	74.2

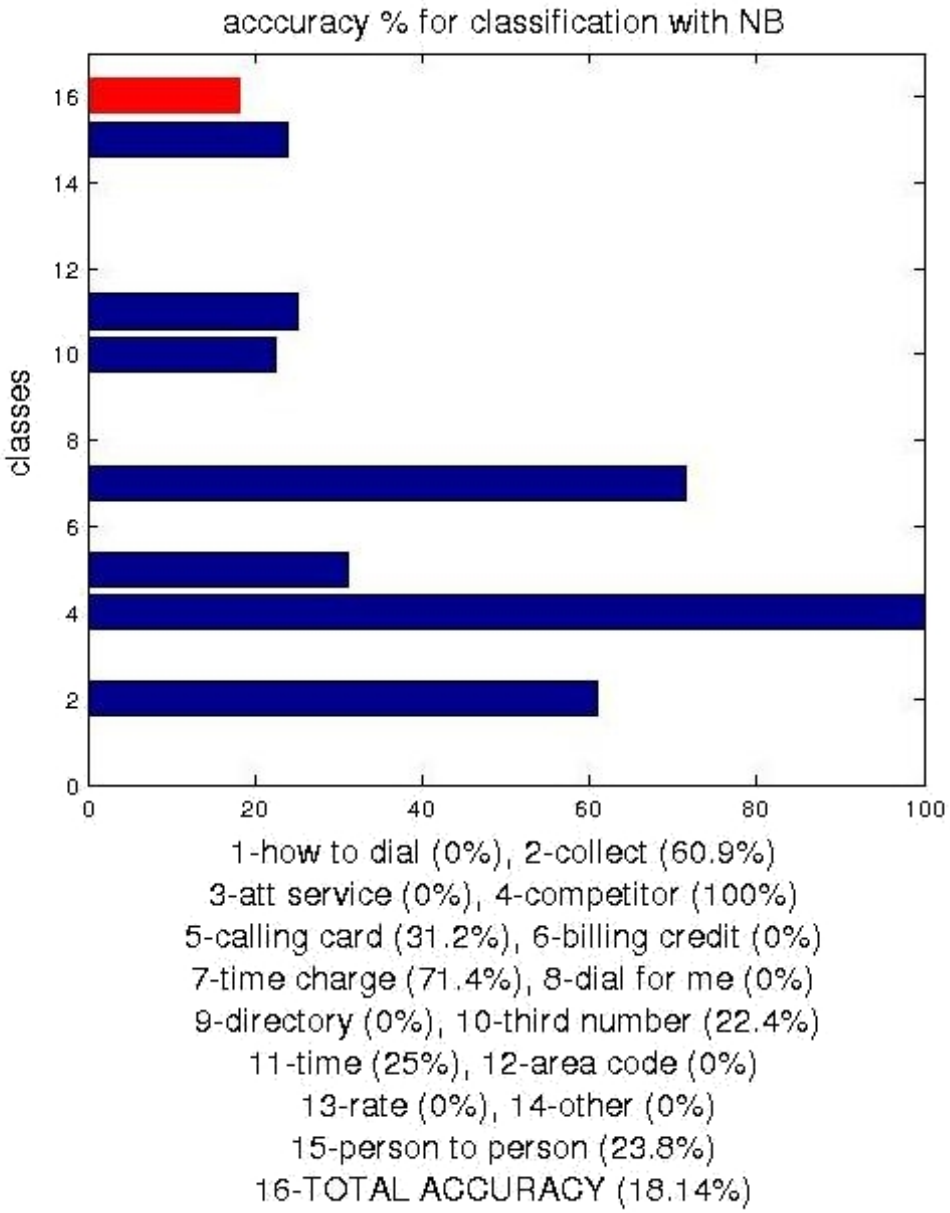


Figure 7: NB baseline results for ATT dataset

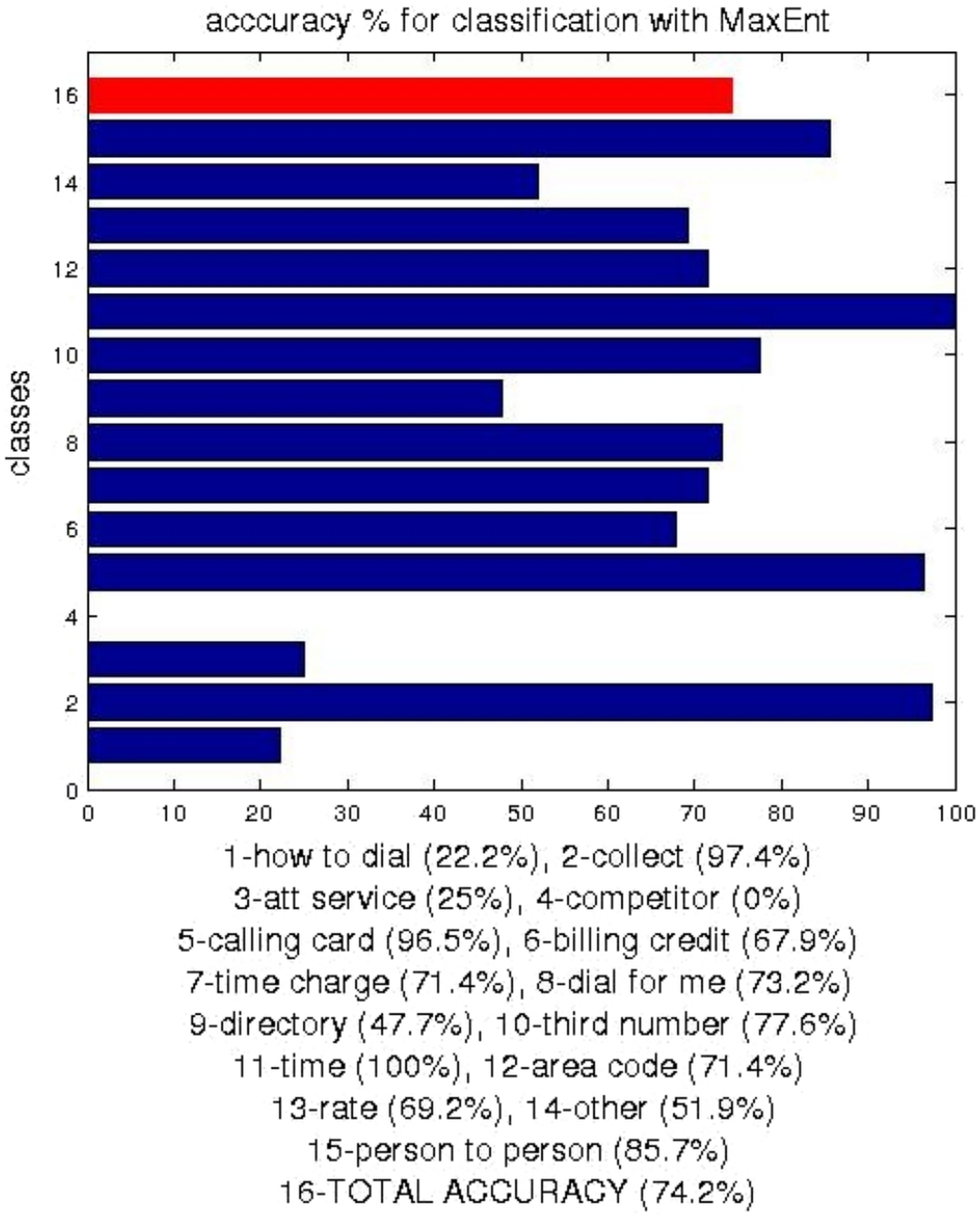


Figure 8: MaxEnt baseline results for ATT dataset

6.3 DATA PROCESSING

We are looking for a way to make our GMM algorithm give the best possible results, it is essential that we try different parameters. When classifying our data using vector-based techniques such as SVD and GMM, there are many stages where the choice of the most appropriate parameters seem to be necessary.

The first parameter we examined on our datasets is the normalization techniques for weighting the elements of the term-document matrix w, which is constructed in the beginning of the procedure. As mentioned, in this work we used Inverse document frequency tf-idf and logEntropy.

The second parameter for which is important to find an optimum value is the dimension of the representation of the matrix w, which is built using the technique of singular value decomposition (SVD). When using SVD there is no specific number of dimensions to reduce the initial term-document matrix. In order to find the best dimension we ran several tests for a wide range of dimensions. In this work we applied the SVD using the gensim tool and specifically the models.LsiModel class due to it's ability to deal effectively with sparse matrices (more information and Python code can be found in Appendix A).

Finally, when we apply the GMM technique there some parameters we need to examined in order to get the best possible results. These parameters are the number of gaussian mixture components in each GMM, as well as the type of covariance (tied, full, diagonal, spherical). We need to compare GMMs with spherical, diagonal, full, and tied covariance matrices since, although one would expect full covariance to perform best in general, it is prone to overfitting on small datasets and does not generalize well to held out test data.

In the following flowchart (Figure 9) we can see all the stages of our classifier which implements SVD and GMM as analyzed in Chapter 4. After that, we will discussed in detail which parameters we examine in each stage and what are the results of this.

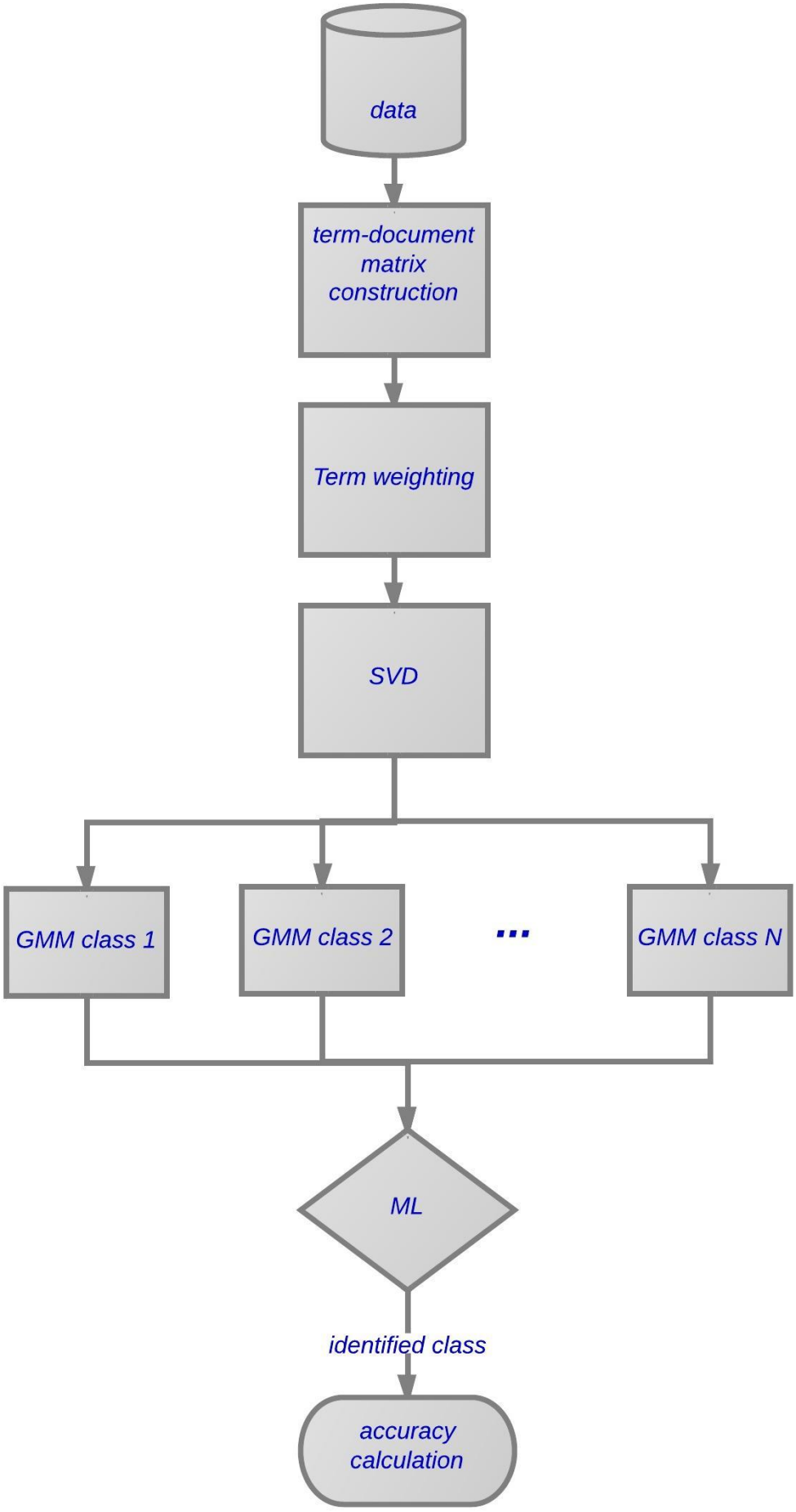


Figure 9: Flowchart with the stages of our classifier

6.4 RESULTS OF THE EXPERIMENTS

In this section all the results of all the experiments are shown. All the experiments were conducted in order to maximize the GMM algorithm's accuracy and to do so, we use the high-level programming language Python. More details on the numerical results of the experiments as well as part of the Python code can be found in Appendix A.

6.4.1 *tf-idf and entropy weighting*

As is can be seen in Figure 9 after creating the term-document matrix we have to normalize it, since not all of the features are equally important and meaningful. Thus, the first parameter we examined on our datasets is the normalization techniques for weighting the elements of the term-document matrix w and here we compare are tf-idf and logEntropy. The results can be seen in the following figure.

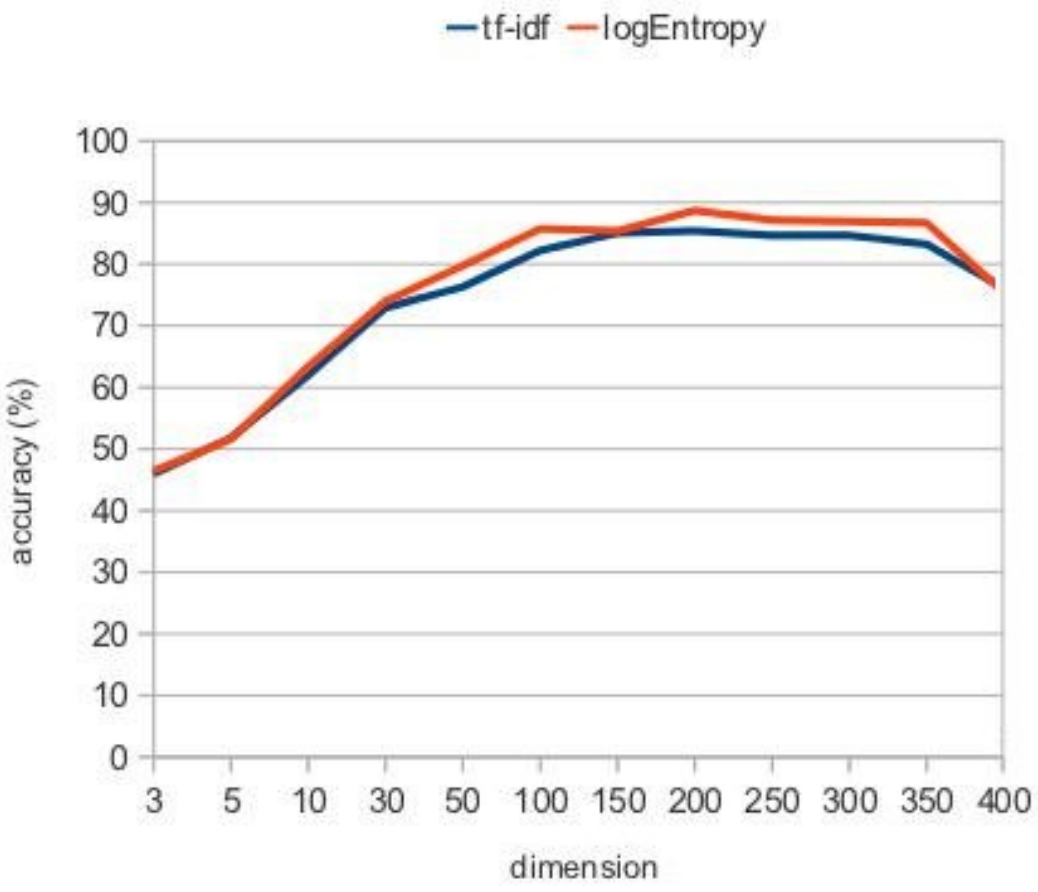


Figure 10: Comparison of the accuracy for tfidf and logEntropy weighting

In Figure 10 we can see that both curves peak in dimension 200. Furthermore, it is undeniable the fact that logEntropy weighting results in higher accuracy for the most part of the dimension range. The difference is not that big. The highest divergence is 3.5% for dimension=100.

6.4.2 *Covariance Type*

This experiment was carried out so that we can compare the results of our GMM algorithm for different covariance type. The four distinct types are spherical, tied, diag and full. We depict the accuracy of the algorithm for each type separately as well as the average accuracy of all the types for each dimension of the svd-reduced matrix we test for a better comparison. We ca see the results in the following figure.

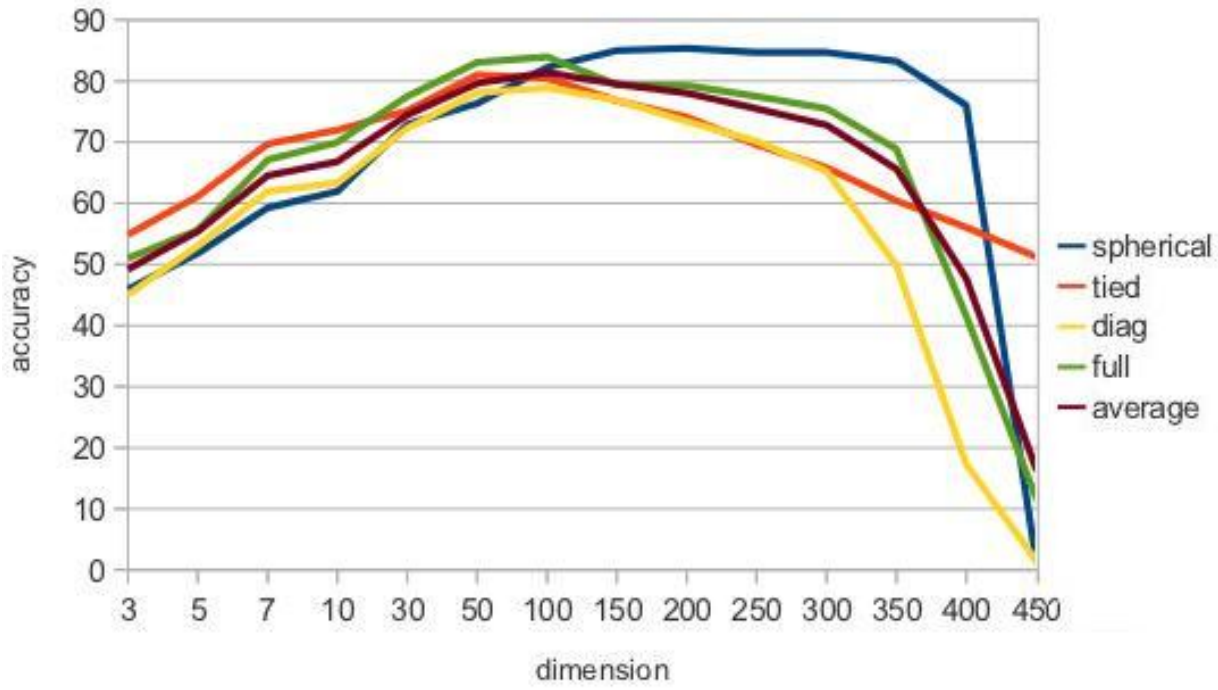


Figure 11: Accuracy of our classifier for different covariance types

It can obviously be seen in Figure 11 that the greatest result is given under spherical covariance for dimension 200. However, for dimensions smaller than 100 it seems that spherical covariance results in the worst accuracy of all. The second greatest result is given under full covariance for dimension 100 which also seems to be higher that the average accuracy for all dimensions lower than 350. On the other hand, tied covariance gives the best average accuracy for all the

dimensions we tested here. The diagonal covariance gives the worst results, being under the average for all the dimensions.

6.4.3 *Number of GMM components*

The next experiment was conducted in order compare the results of our algorithm for different number of Gaussians in each GMM. The selection of the number of components is an important issue as with too many components, the mixture may over-fit the data, while a mixture with too few components may not be flexible enough to approximate the true underlying model. Out of the four covariance types (spherical, tied, diag and full) we used the one which results in the best accuracy. So did we for the dimension. The following figure shows the accuracy of the algorithm for each gaussian number.

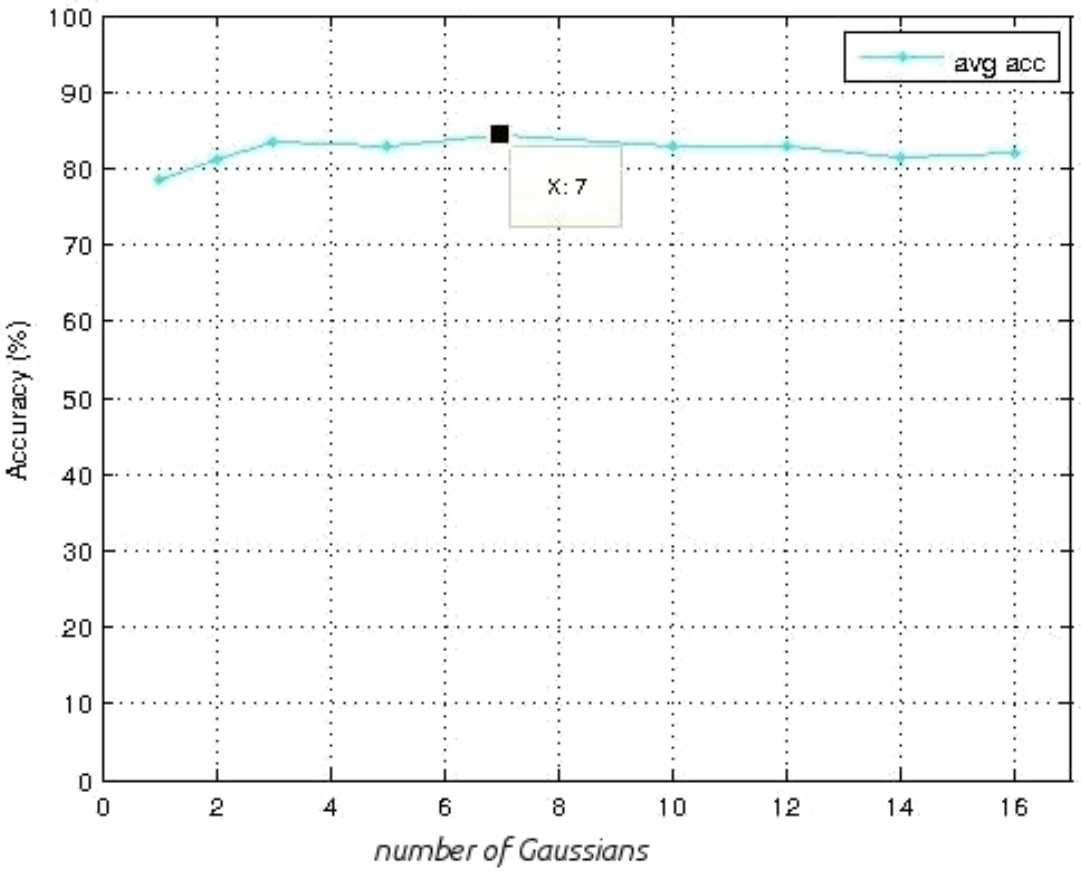


Figure 12: Accuracy of our classifier for different number of Gaussians

It can easily be noticed from Figure 12 that our results seem to create an almost straight line. The highest accuracy is given for 7 gaussians and the second highest accuracy is given for 3 gaussians. However, all the results are nearly equal. As a consequence, the number of the gaussians in the GMMs seems not to play an essential role.

Trying to explain this conclusion, we created 3-D depictions, Figure 13, of 3 only dimensions of the svd-reduced term-document matrix with tf-idf weighting for two random classes. Observing these 3-D depictions we can understand that since the classes are not totally distinct, only one gaussian is not enough. However we can see that the data are distributed in such a way that not a high number of gaussians is necessary,in order to discribe them.

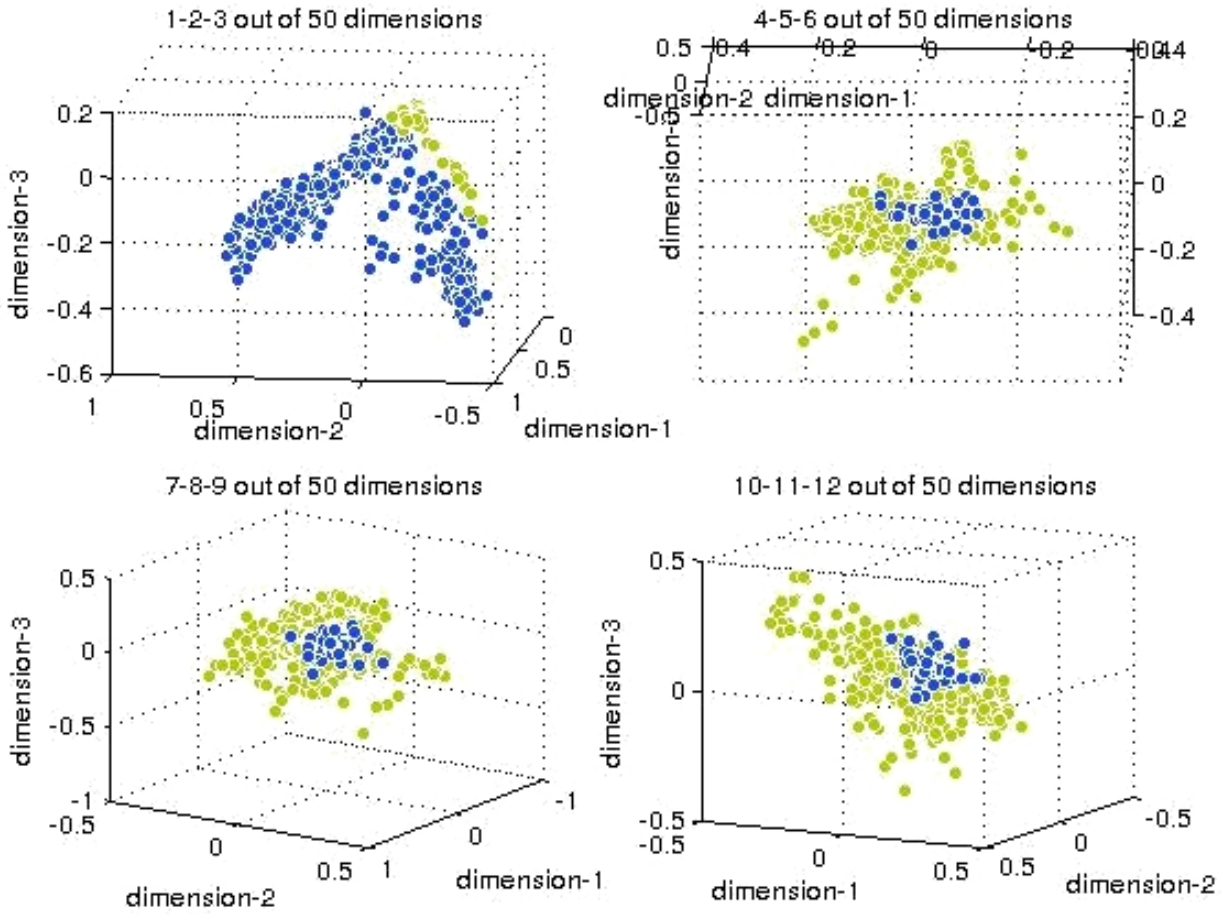


Figure 13: 3-D depiction of 3 dimensions

6.4.4 *SVD dimension*

One of the most important parameters in our algorithm is the dimension in which we reduce our term-document matrix through the SVD procedure. The choice of the most appropriate dimension depends on the data. For this reason we carried out an experiment and we calculated the accuracy of the algorithm for each dimension as the average accuracy of the algorithm for each covariance type. The figure which follows shows the results.

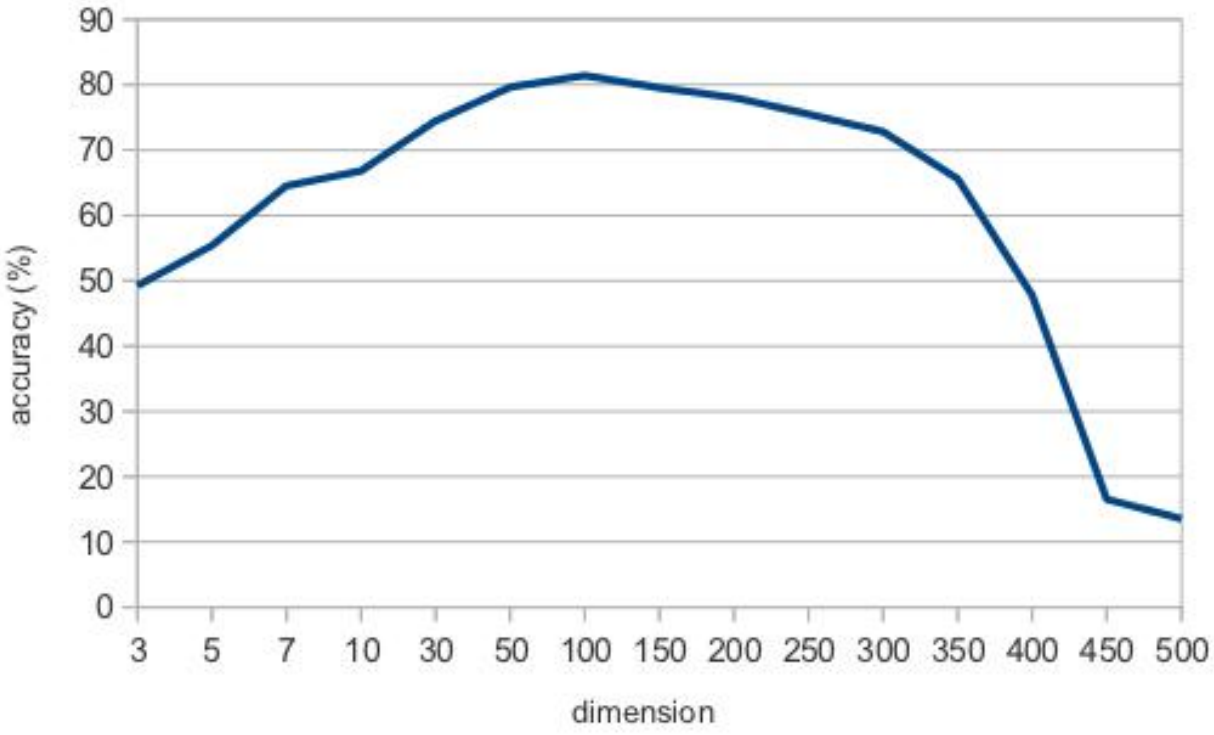


Figure 14: Comparing the SVD dimensions

As far as it can be seen in Figure 14, for too low dimensions we do not get high accuracy. From this fact we conclude that more information is needed (lack of information). For higher but not the highest dimensions such as 50 or ever better 100, we get the best possible accuracy. On the other hand, as we increase the dimension the accuracy gets worse and worse. The obvious reason is that the more information we use, the more unimportant information is taken into account. To put it differently, more noise is taken into consideration.

6.5 COMPARISON OF THE RESULTS

Here is a comparison of the results of the common algorithms we applied.

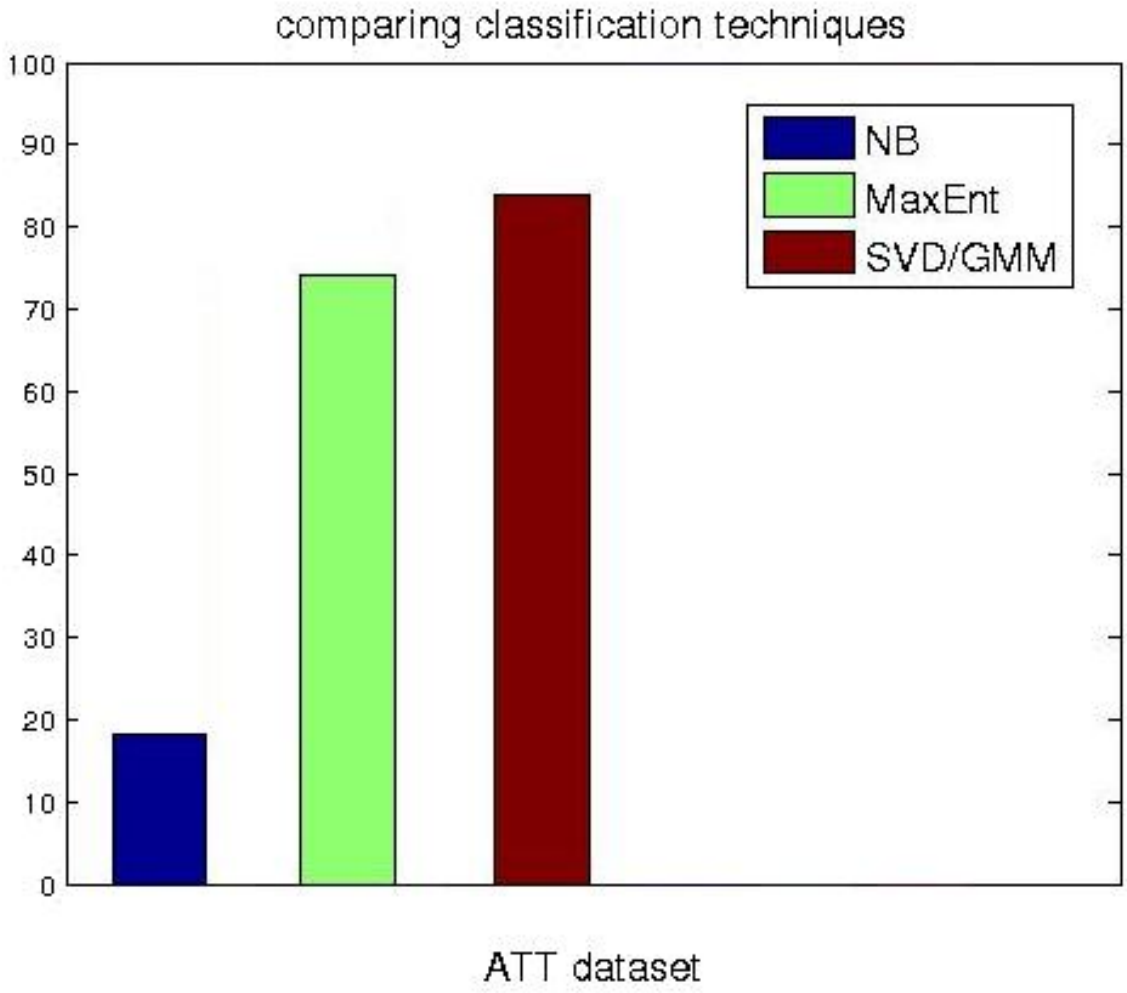


Figure 15: Comparing the baseline classifiers to GMM for the ATT dataset

As we see in Figure 15, for the ATT database we get different results for different techniques. The Naive Bayes classification gives the lowest accuracy and even maxEnt is not as effective as someone could expect. The highest accuracy is achieved using Gmm. This result was expected due to the fact that ATT dataset consists of a high number of classes and as we mentioned in Section 5.2 there is lack of data for many classes of the former dataset.

Next, we will use two additional classification methods, hoping for an improvement. This time we will apply the tree classifier from Section 3.2.2. Furthermore, we will classify each call request based on the cosine similarity which was referred in Section 2.2.6.1. Here are the results.

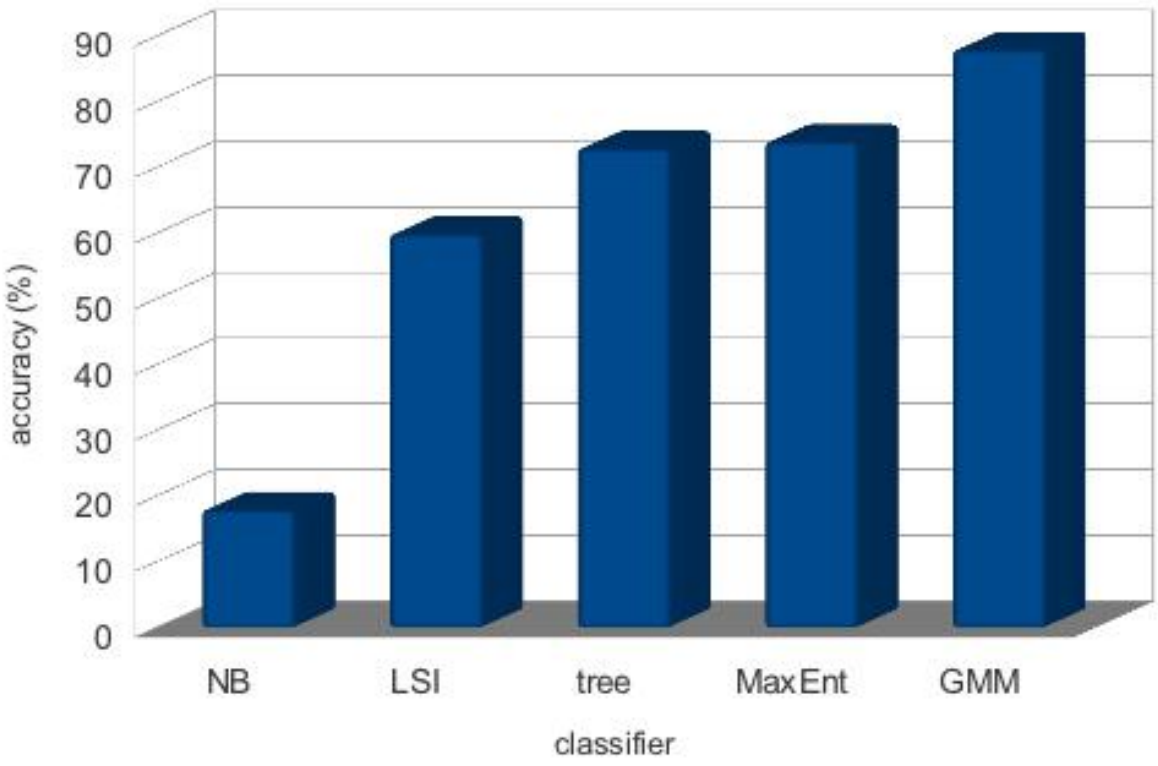


Figure 16: Comparing the accuracy

In Figure 16 we notice that the tree classifier gives almost the same results as MaxEnt. In addition to this, the accuracy with cosine similarity is nearly 10% lower that the tree classifier. As a consequence, the best classifier for the ATT dataset is GMM, which is nearly 14% higher in accuracy that the second better classifier which is MaxEnt.

To sum up, after having taken all the results of both datasets into account, we conclude that GMM is the most appropriate classification technique for our Call Routing task.

Here we compare the precision and the recall between the two best classification methods for our task, MaxEnt and GMM.

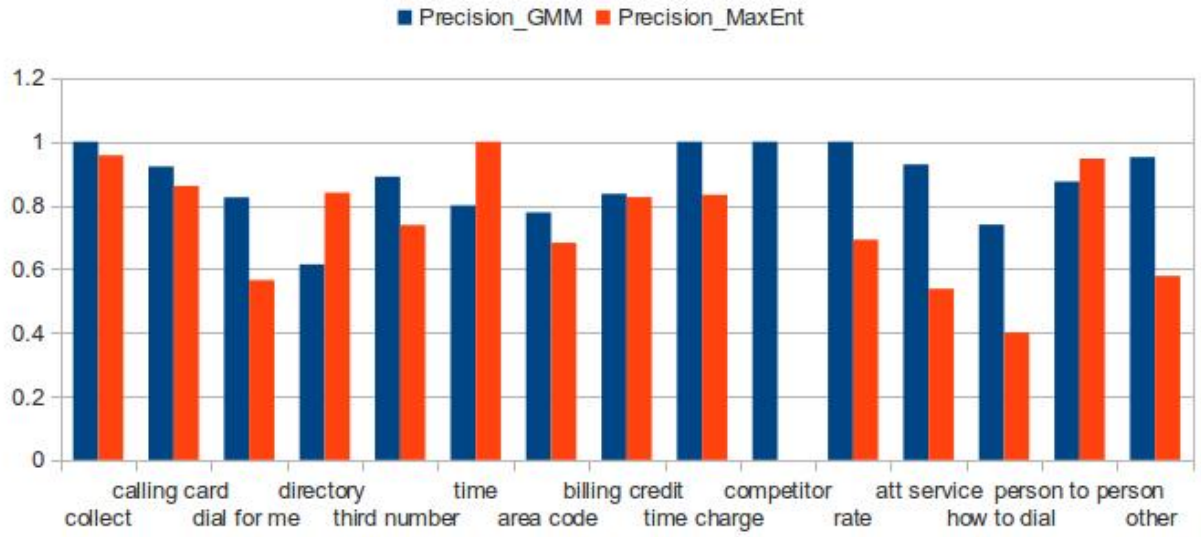


Figure 17: Comparing the Precision

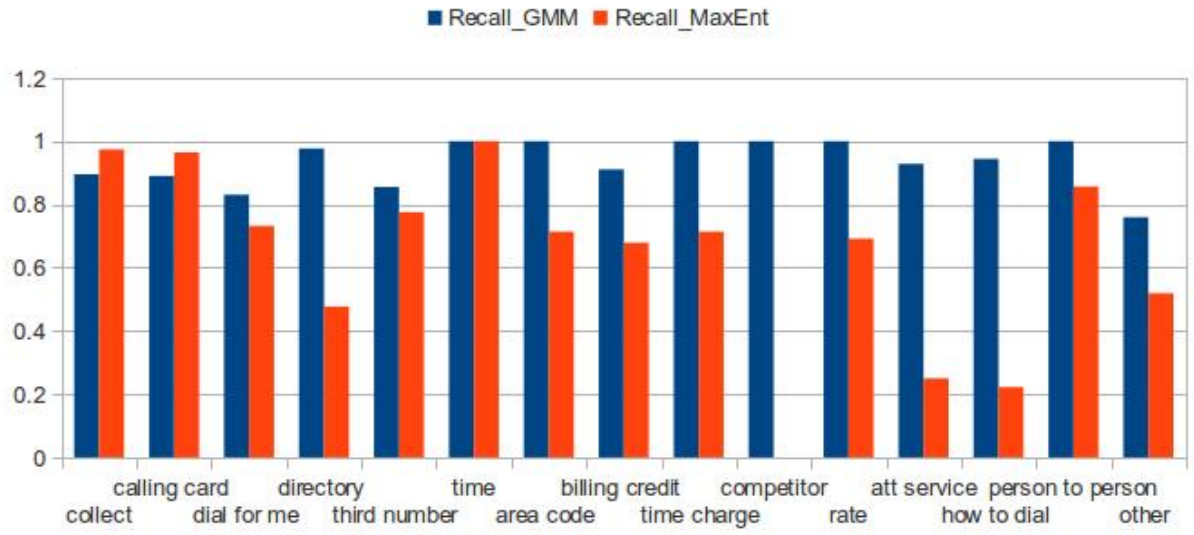


Figure 18: Comparing the Recall

As we can see both precision and recall are higher with GMM for most of the classes.

Table 7: Macro-Average Precision,Recall,F1 (%)

classifier	Precision	Recall	F1
Tree	71.9	63.1	67.2
GMM	87.7	93.3	90.4
MaxEnt	69.7	63.8	66.6

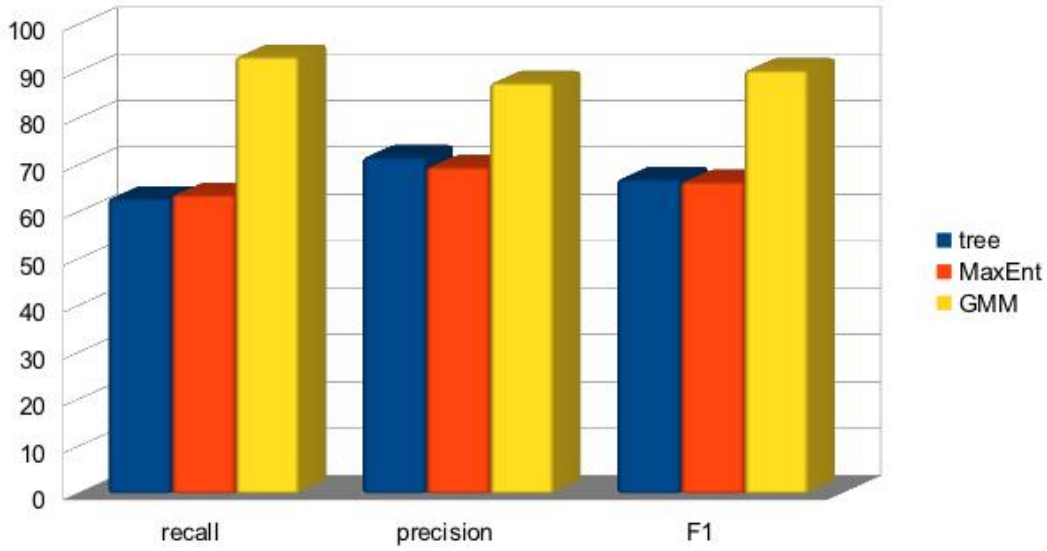
Table 8: Accuracy of Baselime methods

Baseline Method	**acc %**
Naive Bayes	18.1
LSI	60.8
Maximum Entropy	74.2

Table 9: Accuracy of our proposed methods

Our Proposed Method	**acc %**
Tree Classifier	72.8
GMM Classifier	88.2

Figure 19: Precision, Recall, F1 of the top 3 methods



7

CONCLUDING REMARKS AND FUTURE WORK

This chapter summarizes and features the results and the conclusions from our work. In addition to this, possible future work is proposed.

7.1 CONCLUDING REMARKS

In this work we talked about spoken dialogue systems and spoken language understanding and we focused on Call Routing systems. In our Call Routing task callers have to be automatically routed to the right place in the call center (live agent or automated service), letting the users describe the reason for their call in their own words.

Call Routing is a classification task and in order to accomplish it we implemented two different algorithms, which we applied on real American-English call requests (from ATT). Our algorithms are a tree classifier and a GMM classifier, and we compare their results to those of the baseline classifiers, Naive Bayes and Maximum Entropy.

The tree classifier uses mutual information and salience so as to extract features-phrases from the training data. The results we get from it (72.8%) are comparable to those of the Maximum Entropy (74.2%). They are the second best classifier we tested on our data. The worst results are given by Naive Bayes (18%).

The highest accuracy (88.2%) in all the experiments is from the GMM classifier. This approach starts from the pre-processing of the train calls, followed by the creation stage of dictionary and term weighting of bringing a text version of the classical representation through words in matrix format. The following step is the reduction of the dimensions of the matrix using the Singular Value Decomposition. Finally, this matrix is used for the training of the GMM.

Taking all the results into account, we come to the conclusion that the vector-based methods we used and combined work effectively on our data. On the other hand, the results are not ideal since in a real call center 12 out of 100 users would be routed to a wrong service. This means that there is space for improvement, trying more different methods. Some possible future work is proposed in the following section.

7.2 FUTURE WORK

As we mentioned above, the results of this work are quite high, but in the real world and in a real call center we need all the users to be served correctly. Consequently, there is space as well as need for improvement. Here, we propose some ideas which would be of great interest to implement and test on our data in order to achieve better results.

A first proposal would be to try to find better methods for the extraction of the salient phrases for each class of the training data. In our work we used mutual information and salience. However, we noticed that some of the extracted phrases were not as salient as they should be. In addition to this, after the salient phrases extraction, we could use a rule-based instead of a statistical approach. So, we could apply grammar-based parsing on each utterance, but this method would require more supervision.

One further proposal would be to use discriminative training (DT) for call classification and routing. Discriminative training considers both positive and negative examples during training to minimize the classification error and increase the score separation of the correct hypothesis from competitors. DT improves portability by making the classifier robust to different feature selection and by decreasing the amount of training data needed [17]. Early work with discriminative classification algorithms was completed on the AT&T HMIHY system [4] using the Boostexter tool, an implementation of the AdaBoost.MH multiclass multilabel classification algorithm [29].

Moreover, in our GMM classifier we could not use the SVD technique and we can instead try the NMF method. The nono-negative matrix factorization is a method with which we can reduce the dimension of a term-document matrix.

Other alternatives would include a markov random field (MRF) approach and of course many combinations of different techniques. Ofcourse, one final idea which is important would be to check our proposed methods on a different database such as the ATIS [13] database.

Part II

APPENDIX

APPENDIX

A.1 AT&T

AT&T Inc. (stylized as at&t) is an American multinational telecommunications corporation, headquartered at Whitacre Tower in downtown Dallas, Texas. AT&T is the second largest provider of mobile telephony and the largest provider of fixed telephony in the United States, and also provides broadband subscription television services. AT&T is the third-largest company in Texas (the largest non-oil company, behind only ExxonMobil and ConocoPhillips, and also the largest Dallas company). AT&T is the 21st largest company in the world by market value, and the 13th largest non-oil company. It is also the 20th largest mobile telecom operator in the world, with over 250 million mobile customers.

The current iteration of AT&T Inc. began its existence as Southwestern Bell Corporation, one of seven Regional Bell Operating Companies (RBOC's) created in 1983 in the divestiture of parent company American Telephone and Telegraph Company (founded 1885, later AT&T Corp.) due to the United States v. AT&T antitrust lawsuit. Southwestern Bell changed its name to SBC Communications Inc. in 1995. In 2005, SBC purchased former parent AT&T Corp. and took on its branding, with the merged entity naming itself AT&T Inc. and using the iconic AT&T Corp. logo and stock-trading symbol.

The current AT&T reconstitutes much of the former Bell System and includes ten of the original 22 Bell Operating Companies, along with one it partially owned (Southern New England Telephone), and the original long distance division.

A.2 SALIENT PHRASES TABLES

The following tables presents some salient phrases as extracted for each class of the HMIHY training data from the AT&T call center.

Table 10: AT&T Class Collect

Collect	MI	P(c | f)
a collect call please	3.1226267036	1.0
make a collect	3.72527304241	0.995
a collect call	3.91654884615	0.988
to place a collect	3.7658134087	0.99

Table 11: AT&T Class Calling Card

Calling Card	MI	P(c | f)
my calling card	4.09454057103	0.95
card call please	4.2857844263	1.0
a calling card	4.08388635475	0.95
calling card call please	4.07831373082	1.0

Table 12: AT&T Class Dial for me

Dial for me	MI	P(c | f)
you dial a	4.04172107877	0.96
number for me	6.41728161359	0.88
could you dial	4.77296497704	0.91
for me please	4.98134392947	0.82

Table 13: AT&T Class Directory

Directory	MI	P(c | f)
I need directory assistance	7.81260867438	1.0
I need information	3.7186528151	1.0
need a number	4.13773714744	0.9

Table 14: AT&T Class Third Number

Third Number	MI	P(c | f)
third party billing	7.71883414229	0.94
a third party	7.61473598701	0.92
to a different number	5.39202849732	1.0
bill it to another	6.10082286924	1.0

Table 15: AT&T Class Time

Time	MI	P(c | f)
what time it	6.55996843769	1.0
time it is	7.04212013283	1.0
what time it is	7.03133189257	1.0
time it is in	7.07969491413	1.0

Table 16: AT&T Class Area Code

Area Code	MI	P(c | f)
the area code	5.08609989874	0.63
area code for	4.57977879848	1.0
need the area	5.34891439831	1.0
the country code for	4.68961327711	1.0

Table 17: AT&T Class Time and Charges

Time and Charges	MI	P(c | f)
time and charges	6.50823389171	0.99
time and charges please	6.01992908071	1.0
and charges please	6.00770017215	1.0
charges for this	7.51584707582	1.0

Table 18: AT&T Class Billing Credit

Billing Credit	MI	P(c | f)
a wrong number	5.19294528629	0.99
dialed the wrong number	5.09750650405	0.92
I need credit	5.58823769793	1.0
get credit for	6.4044237219	1.0

Table 19: AT&T Class Competitor

Competitor	MI	P(c | f)
MCI and I	7.5695151864	0.72
I'm on sprint	8.30648078057	1.0
an MCI operator	8.72151827985	1.0

Table 20: AT&T Class Rate

Rate	MI	P(c | f)
rates are for	6.92729392945	1.0
how much it would	6.63987755969	0.9
know how much it	6.56163998536	1.0
how much this call	5.85287128303	1.0

Table 21: AT&T Class ATT Service

ATT Service	MI	P(c | f)
customer service please	5.75425159696	1.0
questions on my bill	8.7576472061	1.0
my phone isn't working	11.079575301	1.0

Table 22: AT&T Class How to dial

How to dial	MI	P(c | f)
how to dial it	6.02081939918	1
how do I do	5.4510681487	0.6
I have to dial	5.51982376941	0.66
tell me how to	4.19017382714	0.52

Table 23: AT&T Class Person to Person

Person to Person	MI	P(c | f)
person to person	4.43636288221	0.99
to person call	5.08610638864	0.99
wanna make a person	4.75323752133	1.0
person to person call	5.11053013395	0.99

A.3 THE STANFORD CLASSIFIER

The Stanford Classifier is a general purpose classifier - something that takes a set of input data and assigns each of them to one of a set of categories. It does this by generating features from each datum which are associated with positive or negative numeric "votes" (weights) for each class. In principle, the weights could be set by hand, but the expected use is for the weights to be learned automatically based on hand-classified training data items. (This is referred to as "supervised learning".) The classifier can work with (scaled) real-valued and categorical inputs, and supports several machine learning algorithms. It also supports several forms of regularization, which is generally needed when building models with very large numbers of predictive features.

This software is a Java implementation of a naive bayes and a maximum entropy classifier. Maximum entropy models are otherwise known as conditional loglinear models, and are essentially equivalent to multiclass logistic regression models (though parameterized slightly differently, in a way that is advantageous with sparse explanatory feature vectors).

For using Stanford Classifier in this work, we downloaded the latest version of the classifier from the official site of the Stanford NLP Group: 'http://nlp.stanford.edu/software/classifier.shtml'.

A.4 EVALUATION OF SEMANTIC DISTORTION MEASURE FOR TWO FRAGMENTS

Let N_1,N_2 be the number of training utterances for which the fragments f_1, f_2 occur, respectively, X_{1k}, X_{2k} be the number of training utterances of call type c_k for which the fragments f_1, f_2 occur, respectively, Y_{12k} be the number of training utterances of call type c_k for which both the fragments f_1, f_2 occur.Then a simple version of the semantic distortion measure which works well in practice, is $d_M(f_1, f_2)$ where K is the number of classes.

$$d_M = \frac{1}{k} \prod_{k=1}^{K} \frac{(N_2 X_{1k} - N_1 X_{2k})^2}{N_1 N_2 (X_{1k} + X_{2k} - Y_{12k})} \tag{40}$$

A.5 NUMERICAL RESULTS OF THE EXPERIMENTS

The following table shows the accuracy of the GMM algorithm for ATT dataset with tf-idf weighting.

Table 24: GMM accuracy for the AT&T dataset with tfidf weighting

dimension	spherical	tied	diag	full	average
3	45.936	54.888	44.994	51.001	49.20475
5	51.826	61.131	53.004	55.595	55.389
7	59.246	69.729	61.955	67.138	64.517
10	61.955	71.967	63.369	69.965	66.814
30	72.909	75.147	72.32	77.503	74.46975
50	76.325	81.037	78.092	83.039	79.62325
100	82.214	80.448	78.916	83.981	81.38975
150	85.041	76.796	76.914	79.388	79.53475
200	**85.395**	74.087	73.38	79.27	78.033
250	84.688	69.611	70.2	77.503	75.5005
300	84.688	65.842	65.253	75.501	72.821
350	83.274	60.424	49.823	68.905	65.6065
400	76.09	56.066	17.314	41.578	47.762
450	1.531	51.237	1.531	11.779	16.5195
500	1.531	48.881	1.531	2.238	13.54525

Table 25: GMM accuracy for the AT&T dataset with logEntropy weighting

dimension	spherical	tied	diag	full
3	46.525	55.241	45.465	50.530
5	51.826	61.366	52.650	54.888
7	59.717	69.611	63.133	67.962
10	63.251	73.027	63.251	70.554
30	74.441	78.681	75.383	82.214
50	78.799	82.332	79.034	85.159
100	83.274	80.565	79.388	85.395
150	85.277	76.796	77.150	82.450
200	86.808	74.205	72.085	79.388
250	**88.221**	70.082	70.318	78.681
300	86.219	65.724	65.724	77.856
350	86.572	60.660	47.232	67.138
400	75.383	55.124	17.432	39.576
450	1.531	50.648	1.531	12.839
500	1.531	48.528	1.531	2.002

A.6 PART OF GMM IMPLEMENTATION IN PYTHON

Listing 1: Term weighting

```
1
from gensim import models

# here is some code for tfidf weighting using the tool gensim

6 tfidf = models.TfidfModel(corpus)
corpus_normalized = tfidf[corpus]


# here is some code for logEntropy weighting using the tool
    gensim
11
log_ent = models.logentropy_model.LogEntropyModel(corpus)
corpus_normalized = log_ent[corpus]
```

Listing 2: Gaussian mixture model

```
from sklearn.mixture import GMM

# Here is some code to create a Gaussian mixture model using
    sklearn.mixture.GMM


g1 = mixture.GMM(n_components=3,covariance_type='spherical',
    init_params='wmc' , params='wmc', n_iter=100 )


g1.fit(train[indexes['1']])


# fit: estimates model parameters with the expectation-
    maximization algorithm


g2 = mixture.GMM(n_components=3,covariance_type='spherical',
    init_params='wmc' , params='wmc', n_iter=100 )


g2.fit(train[indexes['2']])


# we do the same for all the classes
```

Listing 3: LSI model

```


from gensim import models, matutils

# using the models.LsiModel and matutils.corpus2dense we manage
    to reduce the dimension of the initial matrix and convert it
    from sparse to dense


lsi = models.LsiModel(corpus, id2word=dictionary, num_topics=
    dimensions )


corpus_lsi = lsi[corpus]


train = matutils.corpus2dense(corpus_lsi,dimensions).T
```

Listing 4: GMM-probability under the model

```
from sklearn import mixture

# Compute the log probability under the model using the method
    score of sklearn.mixture


probability_class1 = g1.score(test_data)


probability_class2 = g2.score(test_data)


# we do the same for all the classes and the class which
    maximizes the probability is the final choice
```

A.7 IMPORTANT TOOLS IN PYTHON

Here is a list of the most important tools we needed in our python code in order to implement our algorithms:

- gensim: gensim is a free Python library designed to automatically extract semantic topics from documents, as efficiently (computer-wise) and painlessly (human-wise) as possible.
- sklearn: it is a simple and efficient tool for data mining and data analysis
- NumPy: NumPy is the fundamental package for scientific computing in Python

Note: all the experiments were carried out in Python 2.7.1+, using Linux-Ubuntu.

BIBLIOGRAPHY

[1] Precision and recall. URL http://en.wikipedia.org/wiki/Precision_and_recall.

[2] S. A. Della Pietra A. L. Berger and V. J. Della Pietra. *A maximum entropy approach to natural language processing*. 1996.

[3] Stephen A. Della Pietra Adam L. Berger, Vincent J. Della Pietra. *A Maximum Entropy Approach to Natural Language Processing*.

[4] Jeremy H. Wright Allen L. Gorin, Giuseppe Riccardi. *How May I Help You?*

[5] Dan McCarthy Barbara Freeman David Getty Katherine Godfrey Bernhard Suhm, Josh Bers and Pat Peterson. *A Comparative Study of Speech in the Call Center: Natural Language Call Routing vs. Touch-Tone Menus*.

[6] S. Chen and R. Rosenfeld. *A survey smoothing techniques for ME models*. 2001.

[7] Stanley F. Chen and Ronald Rosenfeld. *A survey of smoothing techniques for maximum entropy models*. 2000.

[8] Wei Chen. *Building Language Model on Continuous Space using Gaussian Mixture Models*.

[9] Hinrich SchÃŒtze Christopher D. Manning, Prabhakar Raghavan. *An Introduction to Information Retrieval*.

[10] Christopher D.Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*.

[11] Enrico Bocchieri Giuseppe Riccardi, Roberto Pieraccini. *Stochastic automata for language modeling*.

[12] Renato De Mori Gohkan Tur. *Spoken Language Understanding - Systems for extracting semantic information from speech*.

[13] Larry Heck Gokhan Tur, Dilek Hakkani-Tur. *What is left to be understood in ATIS?*

[14] P. S. Gopalakrishnan. *An inequality for rational functions with applications to some statistical estimation problems*. 1991.

[15] Allen L. Gorin. *Processing Of Semantic Information In Fluently Spoken Language*.

[16] Allen L. Gorin. *On Automated Language Acquisition*. 1994.

[17] Imed Zitouni Eric Fosler-Lussier Egbert Ammicht Hong-Kwang Jeff Kuo, Chin-Hui Lee. *Discriminative training for call classification and routing*.

[18] Frederick Jelinek and Robert Mercer. *Interpolated estimation of Markov source parameters from sparse data in Pattern Recognition in Practice*. 1980.

[19] Bob Carpenter Jennifer Chu-Carroll. *Vector-Based Natural Language Call Routing*.

[20] A.L.Gorin J.H.Wright and G.Riccardi. *Automatic Acquisition Of Salient Grammar Fragments for Call-Type Classification*.

[21] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*.

[22] Giuseppe Riccardi Allen L. Gorin Kazuhiro Arai, Jeremy H. Wright. *Grammar Fragment acquisition using syntactic and semantic clustering*.

[23] Brian Kingsbury Mohamed Afify, Ruhi Sarikaya. *Tied-Mixture Language Modeling in Continuous Space*. .

[24] Ruhi Sarikaya Mohamed Afify, Olivier Siohan. *Gaussian Mixture Language Models for Speech Recognition*. .

[25] Peter E. Hart Richard O. Duda and David G. Stork. *Pattern Classification*. 2001.

[26] Bhuvana Ramabhadran Ruhi Sarikaya, Geoffrey E. Hinton. *Deep Belief Nets for Natural Language Cal-Routing*.

[27] V. D. Pietra S. D. Pietra and J. Lafferty. *Inducing features of random fields*. 1997.

[28] Haym Hirsh Sarah Zelikovitz. *Using LSI for Text Classification in the Presence of Background Text*.

[29] R. E. Schapire and Y. Singer. *Boostexter: A boosting-based system for text categorization,*.

[30] R. E. Schapire and Y. Singer. *Boostexter:A boosting based system for text categorization, Machine Learning*. 2000.

[31] George W. Furnas Scott Deerwester, Susan T. Dumais and Richard Harshman Thomas K.Landauer. *Indexing by Latent Semantic Analysis*.

[32] Gabriel Skantze. *Error Handling in Spoken Dialogue Systems*.

[33] Allen L. Gorin Susan J. Boyce. *User Interface Issues for Natural Spoken Dialog Systems*.

[34] V. Vapnik. *The Nature of Statistical Learning Theory*. 1995.

[35] S. V. N. Vishwanathan. K-means and gaussian mixture models, 2011. URL http://learning.stat.purdue.edu/wiki/_media/courses/fall2011/598z/kmeans.pdf.

[36] Ciprian Chelba Brendan Frey Ye-Yi Wang, Alex Acero and Leon Wong. *Combination of statistical and rule-based approaches for Spoken Language Understanding*.

DECLARATION

Chania, Greece, July 2014

I hereby declare that this thesis is my own work and effort and that it has not been submitted anywhere for any reason. Where other sources of information have been used, they have been referred.

Georgiadou Despoina, July 10
2014