COMBINING CLASSIFIERS FOR SPOKEN LANGUAGE UNDERSTANDING

Mercan Karahan[*] *Dilek Hakkani-Tür* *Giuseppe Riccardi* *Gokhan Tur*

Computer Sciences Department,
Purdue University
West Lafayette, IN 47907
{mkarahan}@cs.purdue.edu

AT&T Labs − Research
Florham Park, NJ 07932
{dtur,dsp3,gtur}@research.att.com

ABSTRACT

We are interested in the problem of understanding spontaneous speech in the context of human-machine dialogs. Utterance classification is a key component of the understanding process to determine the intent of the user. This paper presents methods for combining different statistical classifiers for spoken language understanding. We propose three combination methods. The first one combines the scores assigned to the call-types by individual classifiers using a voting mechanism. The second method is a cascaded approach. The third method employs a top level learner to decide on the final call-type. We have evaluated these combination methods over three large spoken dialog databases collected ($\sim 10^6$ dialogs) using the AT&T natural spoken dialog system for customer care applications. The results indicate that it is possible to significantly reduce the error rate of the understanding module using these combination methods.

1. INTRODUCTION

Spoken dialog systems aim to recognize and understand the speaker's utterance and then take an action accordingly [1]. In a call routing system, a critical part of understanding a speech utterance is classification into predefined types of intents (i.e. *call-types*). To this end, practical natural language understanding systems employ statistical classifiers[1] since they perform reasonably well given enough training data and do not require any human expertise. As an example, consider the utterance "`I would like to know my account balance`", in a customer care application. Assuming that the utterance is recognized correctly, the corresponding intent or the call-type would be *Account Balance Request* and the action would be prompting the balance to the user by getting the account number with some further dialog or routing this call to the billing department.

State-of-the-art classifiers are not able to achieve a perfect performance from a dialog task success point of view. Usually dialog management is used to recover from the classification mistakes. In this paper, instead, we try to refine classification accuracy by combining multiple methods or independent sources of knowledge at a given stage of the dialog borrowing ideas from machine learning.

A classifier is considered to be different from another either because its training algorithm is different or its training data or features used during training are different. For example, some algorithms are informative, some are discriminative [2, 3], or a classifier trained with one view of the data is different than the one trained with another view of the data [4].

We propose three methods to combine different classifiers. The first method combines the scores of the individual classifiers using a voting mechanism. The second one is a cascaded approach, where another classifier is consulted if the current one fails to assign a call-type to the utterance with a high enough confidence. Intuitively we begin with the best classifier available, and then continue with weaker ones, optionally remembering the previous ones' output. The third method employs a top level classifier, such as a decision tree or regression to decide on what to do. As features, it uses the outputs of classifiers we wish to combine.

The organization of this paper is as follows: Next section summarizes the earlier work on combining classifiers, especially for text categorization. Section 3 reviews the individual classification algorithms we have used in this study. Section 4 presents the combination methods we propose, in detail. Section 5 explains our experiments and results.

2. RELATED WORK

Combining classifiers is a well studied topic in machine learning. It has been recently applied to text categorization domain. In this section, we will briefly summarize what has been done for combining classifiers in text categorization due to the similar nature of text categorization and call classification. More information on automated text catego-

[*]The research reported here was carried out while the author was visiting at AT&T Labs − Research.

[1]During this paper whenever we say *classifier* we mean only a *statistical classifier* not a knowledge-based classifier

rization can be found in [2]. The most common classifier combination methods used in text categorization are *majority voting* (MV) and *weighted voting* (WV). In MV, each classifier votes for classes and the class that gets most votes is taken as the combined decision. In WV, every classifiers' vote is multiplied by its weight, combined score of a class is the sum of the weighted scores. In both approaches, the vote of each classifier can be its confidence on the decisions.

A version of WV is used by Larkey and Croft for combining k-nearest-neighbor classifier with relevance feedback and Bayesian classifier in medical domain for text categorization. [5].

Van Halteren *et al.* analyzed performance of combining classifiers on part-of-speech (POS) tagging [6]. They used Hidden Markov Models, maximum entropy modeling, memory based tagging and transformation based learning system as base classifiers. For combining these classifiers, they analyzed the performance of various ways of voting between classifiers and a second level classifier as a combiner. Note that the classifiers they try to combine only return the most probable POS tag. They got best results on *Wall Street Journal* and *LOB* data sets by using an extended version of voting, which exploits both the context and the POS tags assigned by taggers.

Another version of WV was used to combine classifiers for text categorization by Kofahi *et al.* [7]. In their method, every classifier assigns a similarity value to every class and document pair. Each classifier is assigned a weight that is learned during a tuning phase, and the combination algorithm generates a combined similarity value for every class by performing weighted sum of the similarities of the component classifiers.

As a different approach, Li and Jain applied an adaptive classifier combination (ACC) and a dynamic classifier selection (DCS) method for combining classifiers for text categorization [8]. In DCS, given a test document D, k training samples most similar to D is selected (e.g by k-nearest-neighbor approach), then the decision of the classifier that has highest total precision in this neighborhood is picked. On the other hand in ACC, class based precisions of classifiers in that neighborhood is summed up and the class that got the highest precision in total is picked.

In the addition to the above combination methods, Boosting is also a method to combine many weak classifiers to get a strong classifier. We describe this method in Section 3.2. However, in this work, we deal with little number of very strong classifiers.

3. CLASSIFICATION ALGORITHMS

The aim of call-type classification for spoken language understanding is to assign a confidence score or a binary value to each pair $\langle F, c_i \rangle \in \mathcal{F} \times \mathcal{C}$, where F is the set of features extracted from the spoken utterance, and $\mathcal{C}$ is a predefined set of call-types. The call-types of the pairs that are assigned a score higher than some threshold are given to the dialog manager, which decides on the next action. The features can be the n-grams of the recognizer output of the spoken utterance, as well as the dialog context. In this work, we combined an informative (e.g. Bayesian) and a discriminative (e.g. Boosting) classifier to improve call-type classification.

3.1. Bayesian Classifier

The Bayesian Classifier assigns the call-type $\hat{c}$ which maximizes the conditional probability $P(c_i|F)$ to the spoken utterance, which can be computed using the Bayes' rule:

$$\hat{c} = \underset{c_i}{\operatorname{argmax}} P(c_i|F) = \underset{c_i}{\operatorname{argmax}} \frac{P(F|c_i) \times P(c_i)}{P(F)}$$
$$= \underset{c_i}{\operatorname{argmax}} P(F|c_i) \times P(c_i)$$

$P(F)$ is eliminated since it is constant for a given utterance, and does not effect the final decision. The Naive Bayes assumption is that the features used for the description are all conditionally independent:

$$P(F|c_i) \approx \prod_{j=1}^{n} P(f_j|c_i)$$

The probability of the call-type given the features of the utterance, $P(c_i|F)$, can be used as the confidence score of the classifier.

3.2. Boosting

Boosting aims to combine "weak" base classifiers to come up with a "strong" classifier. This is an iterative algorithm, and in each iteration, a weak classifier is learned so as to minimize the training error.

More formally, the algorithm (for the simplified binary (+1 and -1) classification case) is as follows:

- Given the training data from the instance space X: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in X$ and $y_i \in -1, +1$
- Initialize the distribution $D_1(i) = 1/m$
- For each iteration $t = 1, ..., T$ do
  - Train a base learner, $h_t : X \rightarrow \mathcal{R}$, using distribution D_t.
  - Update $D_{t+1}(i) = D_t(i)e^{-\alpha_t y_i h_t(x_i)}/Z_t$ where Z_t is a normalization factor and α_t is the weight of the base learner.
- Then the output of the final classifier is defined as: $H(x) = sign(f(x))$ where $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$

The confidence of a call-type, c_i, is then given by the formula:

$$P(c_i = +1|x) = \frac{1}{1 + exp^{-2\times f(x)}}$$

A more detailed explanation and analysis of this algorithm can be found in [9].

4. COMBINATION METHODS

We propose 3 combination methods: voting, cascading and learning a top level classifier as combiner.

4.1. Voting

We analyzed two kinds of voting, which were proposed by van Halteren et al.[6]. The difference of our voting methods is that, the confidence score of the classifiers are used as their votes. In MV, the call-type that gets the maximum vote is taken as combined decision. In WV, each classifiers' vote is multiplied with its weight, combined score of a call-type is the sum of the weighted votes. The call-type that gets the highest combined score is assigned to the text.

We propose three WV methods. In the first method, *TotPrec*, we assign total precisions of classifiers as their weight, and compute the overall confidence of call-type c_i as follows:

$$conf_{c_i} = \sum_{k=0}^{K} TP_k \times conf_{k,c_i}$$

where TP_k is the total precision of classifier k and $conf_{c_i}$ is the confidence score of call-type c_i assigned by classifier k. Total precision values are obtained from the training data by doing n-fold cross validation.

Second method, *CTPrec*, is very similar to the first one, but takes into consideration the precision of classifiers on individual classes. So, the final confidence of call-type c_i is computed as follows:

$$conf_{c_i} = \sum_{k=0}^{K} P_{k,c_i} \times conf_{k,c_i}$$

where P_{k,c_i} is the call-type based precision of classifier k on training data obtained by n-fold cross validation. This method favors the diversity of classifiers. A classifier can be more successful on detecting on some of the call-types, while others fail on them.

The third WV method, *PrecRec*, merges features of first and second method. Here is the pseudo-code for this method:

for all classifiers
  if c_i is the top scoring call-type of classifier k
    $conf_{c_i} += (P_{k,c_i} \times conf_{k,c_i})$
  else
    $conf_{c_i} += ((1 - R_{k,c_i}) \times conf_{k,c_i})$

where R_{k,c_i} is the call-type based recall of classifier k on training data, obtained by n-fold cross validation.

4.2. Cascading

In cascading, the decisions of classifiers are combined in a stepwise fashion such that if the base classifier can not assign a confidence score above a threshold Θ_k to any of the call-types, another classifier is consulted. If the same call-type, c_i, has also been picked by second classifier as a top scoring call-type, confidence score of c_i is raised to a value above the rejection threshold. Θ_k is learned from the training data by doing n-fold cross validation. Below is the algorithm of combining a pair of classifiers k and $k+1$.

if $conf_{k,c_i} < \Theta_k, \forall c_i$
  if $(TS_k == TS_{k+1})$
    $conf_{k,c_i} = \Theta_k + \alpha$

TS_k is the top scoring call-type assigned to an utterance by classifier k; α is some positive constant. This cascade can be generalized up to K classifiers by looping over all classifiers. While combining classifiers with cascading, we aim to boost-up scores of some call-types which always get low scores even if they are top scoring call-type of an utterance. It also provides a mechanism to recover in rejection[2] cases. The order of classifiers employed impacts the performance. Ideally one should employ first the stronger classifiers then the weaker ones.

4.3. Learning a Combiner

In addition to voting and cascading, it is possible to train a top level classifier to make a final decision using the outputs of the classifiers to be combined. For this purpose we have employed well-known machine learning methods, such as linear and logistic regression, decision trees, and Boosting.

4.3.1. Linear Regression

Linear regression fits a line to a set of points in d-dimensional space. In our case, each classifier forms a different dimension. We performed regression based on call-types, this way at the end of regression we have learned separate regression parameters, a_{k,c_i} for each classifier, k, on each call-type, c_i. Linear regression then becomes a combiner where we use these parameters to compute combined confidence score of the call-types as in the below formula:

$$conf_{c_i} = b + \sum_{k=1}^{K} a_{k,c_i} \times conf_{k,c_i}$$

Here $conf_{k,c_i}$ is the confidence score of call-type c_i assigned by classifier k. Further information on linear regression can be found in [10].

[2]Explained in detail in Section 5.2.

4.3.2. Logistic Regression

Logistic regression is similar to linear regression; only the regression formula is different:

$$conf_{c_i} = \frac{1}{1 + exp^{-(b + \sum_{k=1}^{K} a_{k,c_i} \times conf_{k,c_i})}}$$

As it can be understood from the formula, logistic regression fits a curve instead of a line to a set of points. In our experiments we have used the Newton-Raphson Method to learn the regression parameters [11].

4.3.3. Decision Trees

Decision trees (DTs) classify instances by sorting down the tree from the root to some leaf node following a set of if-then-else rules using the predefined features [12]. The difference of decision trees from regression methods is that the continuous features (such as the confidences of the call-types) are automatically discretized (or quantized) during the decision tree training and it is possible to augment them by additional features, such as the length of the utterance to be classified. For our purpose, we have tried various feature sets, such as using only the top scoring call-types of all classifiers or all the call-types along with their confidences.

4.3.4. Boosting

Actually Boosting is a method to combine many weak classifiers to get a strong classifier. So, it makes sense to combine multiple classifiers using the Boosting algorithm. However in this work, we deal with little number of very strong classifiers. Still we employ Boosting to combine the outputs of the individual classifiers like we employ the decision trees. We have used similar features as in decision trees.

5. EXPERIMENTS AND RESULTS

In order to evaluate these combination methods, we carried out experiments using human-machine dialogs as collected by the AT&T *How May I Help You?*^{SM} (*HMIHY*^{SM}) natural spoken dialog system. We first describe our test domain and data, and define the evaluation metrics. We then give the results obtained by combining multiple classifiers.

5.1. Data

In order to evaluate the proposed combination methods, we have used the utterances collected from 3 different applications, namely T_1, T_2, and T_3. In all these applications, the users of the system are greeted by the open ended prompt of *How May I Help You?*. Then according to their responses the users are either routed to specific sub-systems or are re-prompted for clarification or confirmations.

	T_1	T_2	T_3
Training Data Size	35551	9094	29561
Test Data Size	5000	5171	5537
Number of Call-Types	65	84	97
Call-Type Perplexity	15.95	32.64	32.81
Average Length	8.97	10.66	10.13

Table 1. Data characteristics used in the experiments.

Table 1 summarizes the amount of data used for training and testing for these 3 applications along with the total number of call-types, average utterance length, and call-type perplexity. Perplexity is computed using the prior distribution of the call-types in the training data.

5.2. Evaluation Metrics

While evaluating the classification performance, we used mainly 2 metrics both using micro-averaging allowing multiple call-types. The first one is the *top class error rate* (TCER), which is the fraction of utterances in which the call-type with maximum probability was not one of the true call-types. Inspired by the information retrieval community, the second metric we have used is the *F-Measure* (F-M), which is a combination of *recall* and *precision*:

$$F - Measure = \frac{2 \times recall \times precision}{recall + precision}$$

where recall is defined as the proportion of all the true call-types that are correctly deduced by the classifier. It is obtained by dividing the number of true positives by the sum of true positives and false negatives. Precision is defined as the proportion of all the accepted call-types that are also true. It is obtained by dividing true positives by the sum of true positives and false positives. True (False) positives are the number of call-types for an utterance for which the deduced call-type has got a confidence above a given threshold, hence accepted, and is (not) among the correct call-types. False negatives are the number of call-types for an utterance for which the deduced call-type has got a confidence less than a threshold, hence rejected, but is among the true call-types.

As seen, the F-Measure changes with respect to the given confidence threshold. For lower thresholds, the precision is lower but recall is higher, and vice versa for higher thresholds. In order to optimize the F-Measure we check its value for all thresholds between 0 and 1, and use the best one as the F-Measure of that system, since it is always possible to change the operational threshold of the system.

One difference between these two evaluation metrics is that, the top class error rate only evaluates the top scoring call-type for an utterance, whereas the F-Measure evaluates

all the call-types exceeding the given threshold. So, when the threshold is 0, the recall is 100%, but precision is low.

5.3. Results

Basically we have tried two sets of experiments:

- Combining the classifiers whose training algorithms are different
- Combining the classifiers whose training data are different

For the first set, we have combined the two classifiers explained in the previous section, namely, Bayesian and Boosting classifiers, using the same training data. For the second set, we have partitioned our training data with respect to utterance features (such as length) and using just Boosting, we have trained multiple classifiers from each data partition and combined their outputs on the test set.

In our experiments we have always used n-grams as features while training the classifiers, and did not employ any feature selection. For Boosting, we used the AdaBoost algorithm using Boostexter [13]. For the decision tree system, we have used YaDT tools of Ruggieri which is an implementation of the EC4.5 algorithm [14].

5.3.1. Combining the Classifiers with Different Algorithms

Before trying any of the combination methods, we have computed the baseline performances of the Bayesian and Boosting classifiers on our test data. Table 2 presents these baseline results, when the two classifiers are trained using all the training data available for each task. In all the experiments with the Naive Bayes classifier we only used word unigrams as features. Adding word bigrams and trigrams to the feature set did not result in a significantly different performance, and increased the run-time of the classification. This may be due to the lack of a feature selection step, and data sparseness for higher order n-grams. Note that, about half of the features Boosting has selected are also unigrams.

In the baseline experiments, Boosting outperforms the simple Naive Bayes classifier on all three tasks. However there is still a big room for improvement by using the combination of the classifiers. In the T_2 task, the performance of the classifiers is worse than the other tasks, mostly due to the lack of training data and relatively higher call-type perplexity. Naive Bayes suffers more from this scarcity, as the difference between the performance of the two classifiers is the highest on this task.

We have also computed an upper bound for the performance of the combination classifier, assuming that the combination algorithm can only select one of the top scoring call-types of the two classifiers. In order to compute this upper bound, we performed a cheating experiment and selected the correct call-type if it is selected as the top scoring call-type by any of the classifiers. As seen from these bounds, using a combination of the two classifiers, there is a significant room of improvement for all tasks.

App.	T_1		T_2		T_3	
	TCER	F-M	TCER	F-M	TCER	F-M
Boost	16.52	80.40	30.28	66.99	19.00	78.43
Bayes	19.44	77.12	35.39	61.74	22.50	74.84
Bound	11.18		24.88		14.89	
MV	15.58	**81.61**	30.48	67.79	18.93	79.36
TotPrec	15.36	81.49	30.17	67.88	18.55	79.43
CTPrec	15.34	81.50	30.15	68.07	18.82	79.49
PrecRec	15.32	81.44	30.21	67.79	18.87	79.41
Cascade	16.56	81.03	30.28	67.66	19.00	78.95
Lin. Reg.	**14.92**	81.49	**29.36**	67.89	**18.11**	**79.82**
Log. Reg.	15.70	81.24	29.74	**68.31**	18.87	79.41
Boost	16.28	81.07	29.90	67.21	19.41	78.79
DT	16.72		30.15		20.17	

Table 2. Top class error rates (TCER) and F-Measures (F-M) for combining different learners using 3 different applications, namely, T_1, T_2, and T_3. All numbers are in percentages. The best combination performances are marked with boldface.

As also listed in Table 2, we have tried the proposed combination methods, namely voting, cascading, and training a top level classifier. More specifically, we have tested MV, and 3 versions of WV as explained in Section 4.1, cascading, and four different top level learners, linear regression, logistic regression, Boosting, and decision trees, as explained in Section 4.3. The best combination performances are marked with boldface fonts in the table. One impressive result is that, regardless of the method used we have got some improvement in the performance of the classifier. Furthermore using a top level classifier, namely regression, is found to be the best combination method in our experiments. In F-Measure, we have observed significant[3] improvements for all applications (1%-1.5% absolute increase). Using decision trees or Boosting as top level learners have not helped. Features of Boosting were the utterance and top scoring call-type of Bayesian classifier for that utterance. We have used only top scoring calltypes of Boosting and Bayes as features of decision tree. Note that for the decision tree, we do not have the F-Measure since we do not get a confidence score for all possible call-types.

5.3.2. Combining the Classifiers with Different Training Data

Another approach is combining classifiers trained with different partitions of the training data. The optimum would be using partitions that are separable. As a preliminary step, we divided the data into two, using the length of the utterances, with the assumption that some intents (i.e, call-types) are usually expressed by short utterances and others by long ones. We created two partitions from the data, one has utterances that are shorter than or equal to average number

[3] Based on a 95% confidence interval

	T_1		T_2		T_3	
	TCER	F-M	TCER	F-M	TCER	F-M
All	16.52	80.40	30.28	66.99	19.00	78.43
Long	19.60	76.93	36.63	58.88	26.58	71.57
Short	22.28	76.10	34.04	63.07	23.26	75.36
Bound	10.92		22.85		13.74	
MV	16.98	80.44	29.92	67.97	19.20	79.40
TotPrec	**15.72**	81.48	**28.97**	67.80	18.58	79.20
CTPrec	15.80	81.55	29.16	67.94	18.46	79.29
Cascade	16.62	80.66	30.36	67.32	19.02	78.68
Lin Reg	15.80	81.55	29.78	67.90	**18.10**	**79.87**
Log Reg	16.24	**81.70**	31.06	**68.21**	19.67	79.41
Boost	16.00	80.95	30.19	66.87	19.54	78.48
DT	17.16		30.40		19.07	

Table 3. Top class error rates (TCER) and F-Measures (F-M) for combining learners trained with different data for three applications. All numbers are in percentages.

of words (about two thirds of the data), the other has the rest, and trained two classifiers using them (which we call as *Long* and *Short*), and another using all the data (*All*). Since Boosting performed better in the previous experiments, we used Boosting as the base classifier in these experiments. We classified the test utterances using all the three classifiers, and combined their output with the various combination approaches. Table 3 summarizes the baseline results of the individual classifiers, as well as the upper bounds for the combination classifier, and the performance of the combination methods.

Similar to the results in Table 2, we have observed improvements over the baseline (i.e. "All" row) using most of the combination methods. Regression methods improve the F-Measure (1%-1.5%) whereas in this case using total precision (first method of WV) gave better top class error rates for 2 applications. For cascading, if the *All* model fails, we tried the *Long* or *Short* model depending on the length of the utterance, since we observed that *Long* performs better on long utterances and *Short* performs better on short utterances. In this experiment, features of Boosting are the utterance and top scoring call-types of *Long* and *Short* for that utterance. Features of decision tree are top scoring call-types of the three classifiers.

6. CONCLUSIONS

We have presented three methods, namely voting, cascading, and learning a top level classifier, for combining different statistical classifiers for spoken language understanding. We have evaluated them using the AT&T natural spoken dialog for customer care applications. We combined classifiers whose algorithms or training data are different. One impressive result of this study is that, regardless of the combination method used we have got some improvement in the performance of the classifier, even when the performance of individual classifiers are not comparable. Moreover, using a top level classifier, and especially regression, is found to be the best combination method in our experiments and resulted in significant improvements for all applications.

7. REFERENCES

[1] A. L. Gorin, G. Riccardi, and J. H. Wright, "Automated natural spoken dialog," *IEEE Computer Magazine*, vol. 35, no. 4, pp. 51–56, April 2002.

[2] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[3] Y. D. Rubinstein and T. Hastie, "Discriminative vs informative learning," in *Proceedings of the KDD*, 1997.

[4] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the Workshop on Computational Learning Theory (COLT)*, Madison, WI, July 1998.

[5] L. S. Larkey and W. B. Croft, "Combining classifiers in text categorization," in *Proceedings of the SIGIR*, Zurich, Switzerland, August 1996.

[6] H. van Halteren, J. Zaviel, and W. Daelemans, "Improving accuracy in word class tagging through combination of machine learning systems," *Computational Linguistics*, vol. 27, no. 2, pp. 199–230, 2001.

[7] K. Al-Kofahi, A. Tyrrell, A. Vachher, T. Travers, and P. Jackson, "Combining multipleclassifiers for text categorization," in *Proceedings of the CIKM*, Atlanta, GA, November 2001.

[8] Y. H. Li and A. K. Jain, "Classification of text documents," *The Computer Journal*, vol. 14, no. 8, pp. 537–546, 1998.

[9] R. E. Schapire, "The boosting approach to machine learning: An overview," in *Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[10] C.D. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.

[11] A. Agresti, *Categorical Data Analysis*, chapter 4, pp. 84–117, John Wiley and Sons, 1990.

[12] J. R. Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufmann, 1993.

[13] R. E. Schapire and Y. Singer, "Boostexter: A boosting-based system for text categorization," *Machine Learning*, vol. 39, no. 2/3, pp. 135–168, 2000.

[14] S. Ruggieri, "YaDT - Yet another Decision Tree builder," http://kdd.di.unipi.it/YaDT/.