OPTIMIZING SVMS FOR COMPLEX CALL CLASSIFICATION

Patrick Haffner *Gokhan Tur* *Jerry H. Wright*

AT&T Labs-Research,
180 Park Avenue, Florham Park, NJ, 07932, USA
{haffner,gtur,jwright}@research.att.com

ABSTRACT

Large margin classifiers such as Support Vector Machines (SVM) or Adaboost are obvious choices for natural language document or call routing. However, how to combine several *binary* classifiers to optimize the whole routing process and how this process scales when it involves many different decisions (or classes) is a complex problem that has only received partial answers [1, 2]. We propose a global optimization process based on an optimal channel communication model that allows a combination of possibly *heterogeneous* binary classifiers. As in Markov modeling, computational feasibility is achieved through simplifications and independence assumptions that are easy to interpret. Using this approach, we have managed to decrease the call-type classification error rate for AT&T's *How May I Help You* (*HMIHY*$^{(sm)}$) natural dialog system by 50%.

1. INTRODUCTION

Since the first demonstration of the *HMIHY*$^{(sm)}$ system [3], Interactive Voice Response (IVR) systems increasingly rely on natural language call routing, where a computer attempts to recognize many possible outcomes to an open-ended prompt. The traditional solution to sentence classification is the bag-of-words approach used in Information Retrieval. Because of the very large dimension of the input space, large margin classifiers such as SVMs [4, 5] or Adaboost [1] were found to be very good candidates. To take into account context when two sentences are compared, one can match *N-grams* (a substring of N words) rather than words [1]. Sub-word features can also be considered [6].

Large margin classifiers were initially demonstrated on binary classification problems, where the definition of the margin is unambiguous, and the reasons why this margin leads to good generalization are reasonably well understood. Usually, their performance can be expressed as a single number: the binary classification accuracy. Unfortunately, to generalize these binary classifiers to *Multi-Class Classifiers* (MCC) is not straightforward (in the rest of the paper, the word classifier will only cover the binary case).

The definition of a multi-class margin whose maximization is supposed to optimize the generalization, and the resulting optimization problem, lead to solutions which may be too complex to be practical [7]. The most popular approach is to combine binary classifiers and to produce a vector whose components are the output of one binry classifier. This vector is then used to make the multiclass decision. *Error Correcting Output Codes* (ECOC) provide a good framework for this approach [2]. However, this leaves us with the choice of the code . In the *1-vs-others* case, we have one classifier per class c, with the positive examples taken from class c and the negative examples from the other classes. Class c is recognized when the corresponding classifier yields the largest output. In the *1-vs-1* case, each classifier only discriminates one class from another: we need a total of $M(M-1)/2$ classifiers.

After choosing a multiclass combination of binary classifiers, we are still left with a large number of error criteria to choose from, most of them based on a combination of precision and recall for each class [5] or how the classes are ranked [1]. Another problem which has seldom been addressed is the choice of the model, which is usually assumed to be the same for all the classifiers. However, some of the classifiers have to learn tasks with richer data than others, and may benefit from more complex models [6].

Ideally, one would like define a framework that provides us with a complete solution, that covers all choices in a globally optimal way. Section 2 describes an approach that, under well understood simplifying assumptions, attempts to get closer to this ideal.

2. A CHANNEL OPTIMIZATION FRAMEWORK

We have seen in the introduction that defining the optimal large margin multiclass architecture is still an open problem, with a large variety of solutions proposed for each of the critical choices: (i) the *loss function* that must be minimized, (ii) the *output code* (i.e. the set of binary classifiers to choose) and (iii) the *optimization* procedure to minimize this loss. Note the order in which we want to make these choices: given an application, the loss function should be uniquely determined (and measured in dollars or customer satisfaction), and all the other choices must be derived from this *imposed* loss function. In practice, there are many ways to "idealize" a real loss, and we start with the most common one for telecommunication applications.

2.1. The optimal communication channel

The first simplifying assumption is that we have an optimal communication channel. Take the example of interactive call routing: the objective is to obtain as much information as possible from the user within the smallest interaction time. We assume that after a sufficient number of confirmation prompts, the user request is satisfied, so the goal is to minimize the coding costs of such prompts. For a problem with two possible outcomes c and $\bar{c}$, the ideal communication channel would work in three steps. First, the user sends the request $\mathbf{x}$. Second, the computer replies[1] with an estimate of the posterior probability to detect each class $p(c|\mathbf{x}) = o(\mathbf{x})$ and $p(\bar{c}|\mathbf{x}) = 1 - o(\mathbf{x})$. Usually, a logistic remapping is applied to obtain an output ranging from 0 to 1: $o(\mathbf{x}) = \frac{1}{1+e^{-(af(\mathbf{x})+b)}}$. Third, the user sends enough bits to correct the result so that it matches

[1]Our ideal scheme assumes no cost to send $o(\mathbf{x})$ as we can duplicate a model of the computer that computes $o(\mathbf{x})$ on the user side.

the target y. Using an optimal entropic coder, this would correspond to the Kullback-Liebler (KL) divergence [2]

$$H(y|o(\mathbf{x})) = y \log \frac{y}{o(\mathbf{x})} + (1-y) \log \frac{1-y}{1-o(\mathbf{x})} \quad (1)$$

2.2. Class independence assumption

This first assumption gave us the *loss function* we shall minimize, namely the KL divergence. However, when moving to a problem with more than 2 outcomes, the optimization of this KL divergence cannot always be reduced to binary classifications. The second assumption is to uncouple the problem, and consider we have independent problems for each class. This means that observing that an example belongs to classes c or d are independent, *non-exclusive* events, i.e. $P(c, d|\mathbf{x}) = P(c|\mathbf{x})P(d|\mathbf{x})$. An obvious consequence is that an example can belong to several different classes (*multiple label* examples). Denote n the number of independent classes and $o_c(\mathbf{x})$ the estimated probability to observe class c, one can show that the total KL divergence is the sum of the class KL divergences between the $o_c(\mathbf{x})$ and the target outputs y_c. This assumption gives us the *code*: we build an MCC out of n 1-vs-others binary classifiers. Each classifier can be optimized separately. Other conditional independence assumptions lead to the 1-vs-1 case, but they are significantly more complex.

2.3. Optimization: the right balance between regularized and exhaustive search learning

Learning *should* attempt to find the function o over the input data that minimizes an estimate of $E_{\mathcal{D}}(H(y|o(\mathbf{x})))$ where $E_{\mathcal{D}}$ is the expectation over the true distribution of the data. Techniques based on gradient descent or Expectation-Maximization could directly minimize $E_{\mathcal{T}}(y|o(\mathbf{x})))$ on a training set $\mathcal{T}$. However, because of the high dimensionality of the input, overfitting would be a major issue, and we prefer to rely on techniques with good control over generalization.

If the number of hypotheses corresponding to different parameters configurations in the model is finite, we can consider *exhaustive search learning*: for each hypotheses $h \in \mathcal{H}$, compute $o_h(\mathbf{x})$ and measure $E_{\mathcal{V}}(y|o_h(\mathbf{x})))$ on a set of examples $\mathcal{V}$ (called the *validation* set). The minimum hypotheses is kept. Note that we should only choose between a small number of hypotheses $|\mathcal{H}|$, as uniform convergence bounds ([4] page 123) show that the estimate on $\mathcal{V}$ may overfit the generalization error by up to $\sqrt{\log |\mathcal{H}|/|\mathcal{V}|}$.

Regularized learning should be used when the hypotheses space becomes too large. Typically, a penalty term over the size of the parameters is added to prevent excessive overfitting on a set of examples $\mathcal{T}$ (called the *training* set). In the case of the SVMs, this penalty is the inverse of the margin. Unfortunately, not every parameter in our model can be subjected to regularized learning as we do not know how to penalize some of the parameters. Also, efficient optimization algorithms require a convex form, which cannot be obtained after logistic remapping.

In practice, we should separate parameters between the vast majority that can be estimated through regularized learning of a convex function, and a small minority that we can afford to estimate through exhaustive search.

The function f that performs binary classification before logistic remapping is a good candidate for regularized learning. Using SVMs with a kernel k, this function is parameterized by the multipliers α with $f(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{T}} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$. When $\alpha_i \neq 0$, $\mathbf{x}_i$ is called a support vector. Denoting $g(\mathbf{x}) = af(\mathbf{x}) + b$, the loss corresponding to a mis-classified example in Eq.(1) is

$$\log \frac{1}{1+\exp -g(\mathbf{x})} \approx \begin{cases} g(\mathbf{x}) \text{ if } g(\mathbf{x}) \ll 0 \\ 0 \text{ if } g(\mathbf{x}) \gg 0 \end{cases} \quad (2)$$

While, for convexity reasons, SVMs cannot be directly optimized for this loss, Eq.(2) shows it can be approximated with the *hinge* loss traditionally used in SVMs [4]. Bayesian approaches [8] (Gaussian processes, Relevant Vector Machines) should lead to a finer level of understanding of how binary SVMs can be used in a probabilistic model. The α that maximize the margin while minimizing the loss are the Lagrange multipliers of a convex optimization problem that we solve with an iterative procedure that updates the N maximum gradient violators and was first implemented in **SVM*light*** [9].

The parameters left to exhaustive search are the parameters a and b in the logistic remapping, the kernel parameters and the parameter C (C_+ and C_- if we use different ones for positive and negative examples) that weight the SVM loss. Parameters a and b can be chosen using a univariate logistic regression, to minimize the total KL divergence [10] on a first set of validation data $\mathcal{V}_1$ $\sum_{(\mathbf{x},y)\in\mathcal{V}_1} H(y|o(\mathbf{x}))$.

For each choice of kernel or C parameters, the SVM needs to be retrained, so we can only afford a small set of parameter hypotheses $\mathcal{H}$. In summary, for each $h \in \mathcal{H}$, we train the SVM $f_h(\mathbf{x})$ on set $\mathcal{T}$, obtain a_h, b_h and $o_h(\mathbf{x}) = \frac{1}{1+\exp -(a_h f_h(\mathbf{x})+b_h)}$ on set $\mathcal{V}_1$ and choose h that minimizes $\sum_{(\mathbf{x},y)\in\mathcal{V}_2} H(y|o_h(\mathbf{x}))$ on a second validation set $\mathcal{V}_2$. To maximize the amount of validation data available, and avoid reducing the amount of training data, we can use a double N fold cross-validation process to build $\mathcal{V}_1$ and $\mathcal{V}_2$ over all the training data available.

3. EXPERIMENTS

We now evaluate our ideas for SVMs for call classification on the spoken language understanding (SLU) component of the the AT&T's *HMIHY*$^{(sm)}$ natural dialog system. In this system, users are asking questions about their bill, calling plans, etc. Within the SLU component, we are aiming at classifying the input telephone calls into classes (call-types), such as *Billing Credit*, or *Calling Plans* [11, 12]. First, Section 3.1 describes an implementation and measures the impact of a proper optimization of the SVMs against baseline approaches whose performance is considered as state-of-the-art for text classification, namely **Boostexter** [1] and linear SVMs [5]. Second, Section 3.2 shows how this approach was used to reduce the call-type classification error rate in the actual system.

3.1. Implementation and validation

Our methodology has been implemented as a software package called **Llama** where, besides the training data, the user only needs to specify the list of hypotheses kernels to explore. Validation sets are set apart automatically and used to determine the optimal kernel for each class. To our knowledge, this is the first implementation of multi-class SVMs with heterogeneous kernels. **Llama**

[2] In practice, the computer would ask for a confirmation that the outcome should be $y = 1$ if $o(\mathbf{x}) > 0.5$ and $y = 0$ otherwise. User would send back 0 if correct and 1 if wrong. So we need to encode a sequence of bits where the 0 is much more likely than the 1. If the interactivity could be ignored, arithmetic coding could compress this sequence to the entropic limit.

MCC	Hamming Error	Multiclass Error Test	Multiclass Error Train	Threshold Error
Boostexter		33.7	19.9	**15.1(*)**
SVM1	4641	36.9	32.39	**15.1(*)**
SVM1map	4704	35.1	27.8	**15.1**
SVM1mapC	4789	33.7	25.6	**14.1**
SVM2	3820	31.9	7.92	**11.3(*)**
SVM2map	3807	31.1	6.64	**11.2**
SVM2mapC	3846	31.1	5.81	**11.8**
SVM12map	3807	31.1	6.64	**11.2**

Table 1. Error rates using 6058 unigrams as input

optimization also takes advantage of the fact that the kernel cache can be shared between classifiers. For strict binary SVM classification, we verified that its performance is the same as SVM*light*.

To show that this methodology significantly improves performance over a simple recombination of binary SVMs, a large amount of data, representative of the problems that need to be solved, was needed. We used 27892 training utterances, representing 48 call-types. This recently labeled data was preferred over our standard data set which is used in Section 3.2: a much larger number of examples allows more significant results and more call-types makes classification more difficult. We did not use easy to classify dialog acts classes such as "Yes", "No" or "Hello". Because our methodology does not explicit model rejection yet, we also avoided the "Other" class used to train an explicit rejection model.

Results are reported for both unigram (Table 1) and bigram (Table 2) inputs. In the case of unigrams, a sparse feature vector detecting word occurrences in the best[3] sentence recognized by the speech recognizer is computed. In both cases, the Euclidean norm of the feature vector is normalized to 1. We compared SVMs with linear and degree 2 polynomial kernels. As a baseline, we also report results on **Boostexter** using real Adaboost.MH with logistic loss (it has been used for this type of task [13] and we found it to be as good or better than Adaboost with exponential loss). The following SVM configuration were tried: **SVM***p* uses polynomial of degree *p*, with C=1 and without logistic remapping; **SVM***p*map adds logistic remapping; **SVM***p*mapC lets C_+ and C_- be chosen independently for each classifier (between 1 or 2). In addition, **SVM**12map represents an optimal recombination of **SVM**1map and **SVM**2map, where linear and degree 2 polynomials are selected independently for each classifier.

In both figures the first column reports the Hamming error, which is the number of binary mis-classifications accumulated over the 48 classes and 7105 test utterances. The other columns report a multiclass error, which is the percentage of examples where the class with the largest output does not match one of the target classes. Note that is error may not be fully correlated with the Hamming error. The *test* error is computed over 7105 test utterances, the *train* error over 27892 training utterances and the *threshold* error over the 4263 test utterances (a 40% reject rate) with the largest output for the top candidate[4].

Because they perform large margin classification in the same

[3]Feature extracted from full lattices are described in the next section. The definition of efficient lattice kernels is the topic of a separate study (C. Cortes and P. Haffner and M. Mohri, *Lattice Kernels for Spoken-Dialog Classification*, Submitted to ICASSP'03)

[4]The asterisk stresses out MCCs which were not subjected to logistic remapping, and where using the same threshold for all the classes may be questionable.

MCC	Hamming Error	Multiclass Error Test	Multiclass Error Train	Threshold Error
Boostexter		34.2	8.4	**14.5(*)**
SVM1	4015	34.1	22.5	**13.1(*)**
SVM1map	3968	31.6	15.2	**12.2**
SVM2	3777	31.8	3.56	**11.1(*)**
SVM2map	3750	30.8	3.29	**11.1**
SVM12map	3749	30.8	3.29	**11.0**

Table 2. Error rates using 15002 bigrams as input

feature space (with a different metric though), linear SVM and **Boostexter** [5] have similar performances. After optimizing **SVM**1 performance with logistic remapping, **SVM**1map applied to unigrams can be improved using either 2 degree polynomials or bigrams. The percentages of improvement are similar and typically reduce the multiclass error from 34% down to 31.5%. The features induced by 2-degree polynomial kernels correspond to every pair of words found in an utterance (not just consecutive words). Our best choice is **SVM**12map on bigrams: because it uses linear SVMs whenever possible, it is faster than **SVM**2map. Note that no further significant improvements were observed with degree 3 polynomial kernels or trigram inputs, and results are not reported here for lack of space. In summary, properly optimizing the SVM architecture results in a 30% reduction of the *threshold* error, which is our most significant measure of performance. Note that this threshold error corresponds to a false reject rate of about 15%.

A surprising result found in Table 1 is that the automatic selection of C performed in **SVM***p*mapC actually increases the Hamming error when compared to **SVM***p*map, while properly reducing the KL error over the test data. This discrepancy suggests that the hinge approximation in Eq.(2) is too strong. A similar phenomenon is observed with **SVM**12map: a significant reduction in the KL error over **SVM**2map does not translate into a reduction of the Hamming error. We tried automatic selection of C or the kernel to minimize each classifier binary error rather than the KL error, but results actually got worse, with excessive overfitting.

3.2. Deployment in the *HMIHY* system

The SLU component of HMIHY is a series of modules. The input of the SLU component is a compressed form of an ASR lattice, called word confusion network or *sausage* [11]. First a preprocessor converts certain groups of words into a single token (e.g. converting tokens "A T and T" into "ATT".). Then, *Salient Grammar Fragments* (SGFs) [14] are matched in the preprocessed utterance, filtered and parsed [11] to be used as features for the call-type classifier. SGFs are graphs combining *salient phrases*. How to automatically acquire salient phrases from a corpus of transcribed and labeled training data and cluster them into SGFs in the format of *Finite State Machines (FSMs)* [15] is described in [11, 14].

In the previous classifier [14], the learned SGFs are attached to given call-types with weights. Then the classifier decides on the call-type by combining the SGFs occurring in an utterance. With SVMs, we obtain the SGFs as features, but let the SVM decide on how to exploit the occurrences of them in an utterance. Thanks to the modular structure of the SLU component, this has been a

[5]As illustrated in the case of bigram input, **Boostexter** may overfit, and no obvious regularization parameter is available to control this. The 34.2% error rate reported in Table 2 corresponds to full convergence. Stopping learning at the minimum of the test error would have yielded 33.2%.

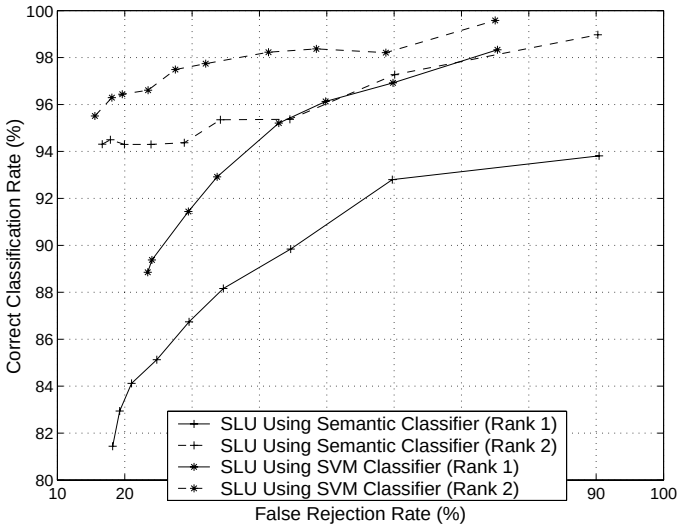

Fig. 1. Improvements in call classification performance.

minor change in the implementation.

During our experiments, we have used 7647 utterances as our training data and 1405 utterances as our test data. All the utterances are responses to the first greeting in the dialog (e.g. "*Hello, This is AT&T. How May I Help You?*".) The word error rate for these utterances is 31.7% and the oracle accuracy, defined as the accuracy of the path in the lattice closest to the transcription, is 91.7%. Sausages yield about the same word error rate and oracle accuracy while being about 100 times smaller than lattices. We used Mangu *et al.*'s algorithm to convert lattices into sausages [16].

In order to evaluate the change in performance due to the change in classifier, we keep the ASR engine, the SGF features and other components unchanged.

Figure 1 presents the results using the **SVM**12map architecture on 19 call-types. Besides the number of call-types, another major difference with the previous section is that we explicitly train a garbage class called "Others". Results are presented in the form of an ROC curve, in which we vary the threshold. The utterances for which top classifier output is below that threshold are rejected. The correct classification rate is the ratio of corrects among the accepted utterances. The false rejection rate is the percentage of rejected utterances which were labeled as one of the non-"Other" call types. Hence, there is a trade-off between correct classification and false rejection by varying the rejection threshold, and the ultimate aim is to reach a correct classification rate of 100%, and false rejection rate of 0%. Rank-2 curves compute the performance using the top two classifier outputs. We have found that these curves give an indication of performance in a dialog system where we have the opportunity for confirmation and error correction. The error rate has decreased about 50% relatively on the rank-2 ROC curve for almost all thresholds. This shows the effectiveness of SVMs for this task. For rank-1, we have managed to improve the classification accuracy by 4-5% points absolute.

4. CONCLUSION

Approaches based on probabilistic modeling have been very successful in speech and language understanding, and allowed to build large architectures that are globally consistent. This works describes a first step toward the optimal embedding of binary large margin classifiers into such models, and immediately shows significant improvements in performance. It also suggests strategies to further optimize the recombination of binary classifiers. Using 1-vs-1 classifiers (instead of 1-vs-others) would better model the fact that classes are exclusive. Replacing the hinge loss in SVMs with better approximations of the logistic loss [8, 13] should be considered. An explicit probabilistic modeling of the rejection would allow a better representation of the task.

5. ACKNOWLEDGMENTS

The authors wish to thank Allen Gorin for initiating and strongly supporting this collaborative project. They are also grateful to Dilek Hakkani-Tur and Tirso Alonso for their help withlatex the experiments.

6. REFERENCES

[1] 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.

[2] E.L. Allwein, R.E. Schapire, and Y. Singer, "Reducing multiclass to binary: A unifying approach for margin classifiers," in *Proc. of ICML*, 2000.

[3] A.L. Gorin, G. Riccardi, and J.H. Wright, "How May I Help You?," *Speech Communication*, vol. 23, pp. 113–127, 1997.

[4] V. N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, New-York, 1998.

[5] T. Joachims, "Text categorization with support vector machines: learning with many relevant features," in *Proc. of ECML-98*. 1998, Springer Verlag.

[6] M. Larson, S. Eickeler, G. Paa, E. Leopold, and J. Kindermann, "Exploring sub-word features and linear support vector machines for german spoken document classification," in *Proc. of ICSLP*, 2002.

[7] Y. Guermeur, A. Eliseeff, and H. Paugam-Moisy, "A new multi-class SVM based on a uniform convergence result," in *Proc. of IJCNN*, 2000.

[8] B. Scholkopf and A. Smola, *Learning with Kernels*, MIT Press, Cambridge, MA, 2002.

[9] T. Joachims, "Making large-scale svm learning practical," in *Advances in Kernel Methods — Support Vector Learning*. 1999, MIT Press.

[10] J. Platt, "Probabilistic outputs for support vector machines and comparison to regularized likelihood methods," in *NIPS*. 1999, MIT Press.

[11] G. Tur, J. Wright, A. Gorin, G. Riccardi, and D. Hakkani-Tür, "Improving spoken language understanding using word confusion networks," in *Proc. of ICSLP*, 2002.

[12] 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.

[13] R. Schapire, M. Rochery, M. Rahim, and N. Gupta, "Incorporating prior knowledge in boosting," in *Proc. of ICML*, 2002.

[14] J. Wright, A. Gorin, and G. Riccardi, "Automatic acquisition of salient grammar fragments for call-type classification," in *Proc. of Eurospeech*, Rhodes, Greece, September 1997, pp. 1419–1422.

[15] M. Mohri, F. Pereira, and M. Riley, "FSM Library – General purpose finite state machine software tools," http://www.research.att.com/sw/tools/fsm.

[16] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus in speech recognition: word error minimization and other applications of confusion networks," *Computer Speech and Language*, vol. 14, no. 4, pp. 373–400, 2000.