DIALOG ACT TAGGING WITH SUPPORT VECTOR MACHINES AND HIDDEN MARKOV MODELS

Dinoj Surendran, Gina-Anne Levow

Computer Science Department
University of Chicago
dinoj,levow@cs.uchicago.edu

ABSTRACT

We use a combination of linear support vector machines and hidden markov models for dialog act tagging in the HCRC MapTask corpus, and obtain better results than those previously reported. Support vector machines allow easy integration of sparse high-dimensional text features and dense low-dimensional acoustic features, and produce posterior probabilities usable by sequence labelling algorithms. The relative contribution of text and acoustic features for each class of dialog act is analyzed.

1. INTRODUCTION

There are several possible cues that humans and machines can use to identify the meaning of an utterance, such as whether it is a question, acknowledgement, or clarification. Several approaches have been proposed to integrate such cues [1] [2] [3], such as how to combine sparse text representations and dense acoustic measurements of different kinds. Previous methods have used a combination of neural networks, decision trees, hidden markov models, principal components analysis, and nearest neighbor algorithms.

We investigate the use of support vector machines [4], and find that no special effort is required to obtain dialog act classification accuracy on a standard corpus that is better than previously published results. Using SVMs and hidden markov models (just forward decoding) results in classification accuracies of 42.5% and 59.1% respectively using acoustic and text features separately and 65.5% together.

In this document, we briefly describe the task and classification algorithms used, followed by several measurements of perfomance in experiments using text features only, acoustic features only, and a combination thereof.

2. TASK DESCRIPTION

The HCRC MapTask corpus [5] is a collection of 128 2-speaker dialogs, of which we used the 64 ‘no-eye-contact’ dialogs. Each dialog has a ‘giver’ giving directions on a shared map to a ‘follower’, and is segmented into parts called ‘Dialog Acts’ (DAs). We assume this segmentation has already been carried out manually. A DA cannot span speaker turns, but turns can consist of multiple DAs.

DAs are labelled with one of a number of tags; the HCRC labellers used twelve tags, and a thirteenth for ‘uncodable’. As this study was explorational, we simply took the task to be a 13-class classification problem. The tags, and their relative frequencies, are described in Table 1.

Table 1. *Tags used by the HCRC MapTask Labellers, from the Dialog Structure Coding Manual. Also shown is the percentage of dialog acts of each type.*

Tag	%age	Description
`instruct`	15.1	commands partner to carry out action
`explain`	7.5	states information that partner did not elicit
`align`	7.2	checks attention & agreement of partner, or their readiness for next DA
`check`	8.0	requests partner to confirm information that checker is partially sure of
`query-yn`	6.0	Yes/No question other than a check or align
`query-w`	3.0	any other question
`ack`	21.0	acknowledgement: minimal verbal response showing that speaker heard preceding DA
`clarify`	4.0	repetition of information already stated by speaker, often in response to a check DA
`reply-y`	12.6	affirmative reply to any query
`reply-n`	3.3	negative reply
`reply-w`	3.2	any other reply
`ready`	7.9	DA that occurs after end of a dialog game and prepares conversation for a new game
`uncodable`	1.2	None of the above

The experiments reported here were done with four-fold crossvalidation. The 64 dialogs considered here are organized into eight parts, q1 to q8. We used four splits, with the

n-th split ($n = 1, 2, 3, 4$) having test data from q$\{2n-1\}$ and q$\{2n\}$ and the training data from the six other conversations. For example, the first split has training data from parts q3 to q8 and test data from q1 and q2. The text features differed in each split, as they could only make use of the training part of the split.

In this way, each of the 14810 examples in the corpus was a test example in exactly one split. The confusion matrices presented here are sums of all confusion matrices from all splits. The final classification accuracy is a weighted sum of the classification accuracies, i.e. the sum of all correctly classified test examples divided by 14810.

In the interests of result reproducibility, we have placed the data and data splits used in our experiments online[1]. Using the exact splits is important, as results were quite variable across splits. For example, the final classification accuracy we report is 65.8%. The actual accuracies obtained for each split were 63.4%, 61.9% 69.4% and 70.9%.

3. CLASSIFICATION ALGORITHMS

Our strategy is to use linear support vector machines on individual data points, and then Viterbi decoding to make use of some contextual information. The Viterbi decoding procedure is slightly non-standard as we have posterior probabilities instead of output symbol probabilities.

3.1. Support Vector Machines

First, we briefly describe what the linear Support Vector Machine (SVM) algorithm does, in the case of binary classification. It is given a set of training examples, where each example is a D-dimensional vector and is labelled as -1 or 1. A linear SVM determines the weights $w \in \mathbf{R}^D$ that should be given to each of the D components/features, and a threshold $b \in \mathbf{R}$ so that the final decision function is

$$f(x) = \text{sign}(w^T x + b)$$

The fastest way to generalize SVMs to n-class, $n > 2$, classification is to split the problem into $n(n-1)/2$ binary classification problems [6] [7] and then using a voting procedure. Label bias was handled by associating a weight of $\frac{1}{p}$ to each class with empirical training set probability p.

A linear SVM is just one of a family of kernel-based algorithms [8] [9]. In general, a SVM algorithm is given n training examples $x_1, \ldots, x_n \in \mathcal{X}$ (here, $\mathcal{X} = \mathbf{R}^d$) with their labels $y_1, \ldots, y_n \in \mathcal{Y}$, *and* a measure of similarity encoded in a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$ with $k(x, z)$ higher when x and z are more similar. The SVM algorithm then says which of the training examples are useful for classification, and how important each of them are, by outputting $\alpha_1, \ldots, \alpha_n \in \mathbf{R}$, and a threshold b. The more important training point x_i is, the higher $|\alpha_i|$ is. The final classification function is then

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(x_i, x) + b\right)$$

Usually, most of the α_i will be zero. Only the x_i where $\alpha_i \neq 0$ are actually used in the classification function; such x_i are termed 'support vectors'.

In a linear SVM, the kernel function is $k(x, z) = x^T z$. The weights w are a linear combination of the training examples: $w = \sum_{i=1}^{n} \alpha_i x_i = \sum_{i \in sv}^{n} \alpha_i x_i$, where sv is the set of support vector indices. Another commonly used kernel function is the Radial Basis Function (RBF) kernel $k(x, z) = e^{-\frac{||x-z||^2}{\sigma^2}}$, where σ is found with cross-validation.

3.2. Hidden Markov Model Decoding

Previous work in this domain [10] [3] [11] reports improved recognition rates when knowledge of the previous dialog act is assumed. This is because the distribution of dialog acts differs greatly with the previous dialog act — Table 2 shows these distributions for part of the MapTask corpus. For example, the probability of an affirmative reply is about 0.5 after a binary question or check or alignment, but under 0.05 after other types of dialog acts. And while checks and alignments are both types of questions as well, the probability of a negative reply after an alignment is about 0.01, after a check 0.08 and after a binary question 0.27.

Table 2. *Transition matrix showing which dialog acts followed which dialog acts. The (i, j)th entry is the percentage of DAs after one of class i that are of class j. For example, of the dialog acts immediately following a binary question (`qy`), 45% were affirmative replies (`ry`), 27% were negative replies (`rn`) and 7% were some other reply (`rw`). These empirical percentages were computed from the training set in the first data split.*

	in	ex	al	ch	qy	qw	ac	cl	ry	rn	rw	rd	un
in	4	5	8	13	9	4	48	0	1	0	1	5	2
ex	8	10	5	7	7	3	43	1	2	2	0	11	2
al	12	4	3	7	4	2	6	2	51	1	2	5	1
ch	3	3	2	2	2	2	4	10	55	8	2	4	2
qy	1	3	0	2	4	1	4	0	45	27	7	3	2
qw	4	2	4	3	4	2	5	11	3	2	48	10	2
ac	29	10	12	6	6	3	11	4	2	1	1	14	1
cl	4	5	11	15	3	2	47	4	4	1	0	4	0
ry	17	7	7	9	5	2	23	9	4	0	1	14	1
rn	7	19	3	6	5	3	34	12	1	1	4	7	0
rw	7	6	4	8	3	5	47	2	2	1	6	10	2
rd	46	11	8	7	12	3	3	2	1	0	2	3	2
un	15	10	4	6	7	4	18	3	10	3	4	9	6

If the computer is a participant in the dialog, knowledge of previous dialog acts on one conversation side can be assumed, as in Taylor et. al.[10]. In general, knowledge of the preceding dialog act cannot be assumed. Of course, it can be estimated, and Stolcke et. al. [2] found that a Hidden Markov Model (HMM) improved perfomance.

[1] http://people.cs.uchicago.edu/∼dinoj/da

The HMM assumption is that the sequence of observations $x_1, \ldots, x_T$ is generated by an underlying first-order Markov sequence of states $q_1, \ldots, q_T$ with each observation x_t generated by the corresponding state q_t only. Note that we are using the dialog act classes as states.

Suppose there are N states. If we know the transition probabilities $P(q_{t+1}|q_t)$, the output symbol probabilities $P(x_t|q_t)$, and the initial probability distribution $P(q_1)$, the Viterbi forward-decoding algorithm [12] is a dynamic programming algorithm that outputs the most likely state sequence $q_1, \ldots, q_T$ given an output sequence $x_1, \ldots, x_T$.

We can estimate the transition probabilities from the training set. However, we do not know $P(x_t|q_t)$, but do know the posterior probability $P(q_t|x_t)$ for each x_t in the test set [13]. Bayes Rule and some other assumptions can be employed to give the slightly modified Viterbi algorithm below [14].

$$\delta_{jt} := \max_{q_1,\ldots,q_{t-1}} P(x_1, \ldots, x_t, q_1, \ldots, q_{t-1}, q_t = j)$$

$$\psi_{jt} := \text{argmax}_{q_{t-1}} \delta_{jt}$$

We use $P(q_1|x_1)$ for the initial state distribution, and then compute δ and record ψ recursively:

$$\delta_{j,1} = P(q_1 = j|x_1)$$

$$\delta_{j,t\geq 2} = P(q_t = j|x_t) \sum_{n=1}^{N} P(q_t = j|q_{t-1} = n)\delta_{n,t-1}$$

$$\psi_{j,t\geq 2} = \text{argmax}_{q_{t-1}} P(q_t = j|q_{t-1} = n)\delta_{n,t-1}$$

Note that the equation for $\delta_{j,t\geq 2}$ is probabilistically incorrect; it should be $P(x_t|q_t)$. Since we only want to find the most likely state sequence, and not its probability as well, we can use $c \cdot P(x_t|q_t)$ instead of $P(x_t|q_t)$, as long as c is a positive real number independent of q_t. Bayes Rule says that $P(x_t|q_t) = P(q_t|x_t)P(x_t)/P(q_t)$. Since we reweighted the probabilities in SVM training so that all classes were equally likely, $P(q_t) = \frac{1}{N}$ is independent of q_t, as is $P(x_t)$. Thus we can substitute $P(q_t|x_t)$ for $P(x_t|q_t)$ in our modified Viterbi procedure; $\psi_{j,t}$ remains the same.

Finally, backtracking obtains the most likely state sequence $\pi_1, \ldots, \pi_T$:

$$\pi_T = \text{argmax}_q \delta_{qT}$$

$$\pi_{t<T} = \psi_{\pi_{t+1},t+1}$$

4. CLASSIFYING WITH ACOUSTIC FEATURES

We used the acoustic features mentioned by Stolcke et. al. [1] that made use of duration (but not pauses), intensity, pitch, speaking rate [15], and speaker identity. Pitch was measured with the ESPS algorithm implemented in the Snack Toolkit [16]. A brief description of our features can be found in Table 3. Each feature were scaled to between -1 and 1 based on the minimum and maximum values of the feature among training examples.

Table 3. *A description of the 49 acoustic and other acoustic features used in these experiments. Range refers to 'maximum - minimum'. The end and penultimate regions are the last and next-to-last 200ms of each DA. F0 measurements were only obtained on certain frames; such frames were termed 'good' while other frames used linearly interpolated pitch values or were assumed to have the conversation-side F0 mean. All values were enventually z-normalized by conversation side. The speaker-based features are both binary.*

Energy	Mean over entire DA, end, and penultimate regions Differences, absolute differences, and ratios between the three means above Standard deviation over all frames in DA Gradient and intercept of best regression line through energy values
F0	Mean, standard deviation, maximum, minimum, and range, over all frames in DA The above parameters, minus the conversation-side means of the same parameters, and divided by the conversation-side standard deviations Gradient and intercept of best regression line through F0 values in each of the three regions Mean over end, and penultimate regions Number of frames with good pitch Fraction of frames with good pitch
Enrate	Mean and standard deviation of speaking rate over all 400ms sections of the DA (stepped every 200ms)
Duration	Length of DA in seconds
Speaker	Whether speaker is follower or giver Whether speaker is same as that of previous DA

This resulted in a classification accuracy of 41.4% with a linear SVM, which was increased to 42.5% after Viterbi decoding. However, as the corresponding confusion matrix in Table 4 shows, nearly half the classes were rarely recognized, and over half the examples were classified as the most common classes (`ack`, `instruct`) suggesting that our method of accounting for label bias was inadequate.

The other possibility is that acoustic features simply did not separate the data. While we certainly believe that our acoustic features need to be improved, previous work on this dataset using only acoustic features (and no context) produced similar recognition rates, such as 42% in[10].

Table 4. *Confusion matrix obtained when using acoustic features with a linear SVM followed by Viterbi decoding. Classification accuracy was 42.5%*

	inst	ex	al	ch	qy	qw	ac	cl	ry	rn	rw	rd	un
in	1932	7	72	4	29	0	25	14	44	3	4	107	0
ex	308	211	25	245	11	17	178	4	52	3	7	43	0
al	441	15	171	43	29	1	83	5	90	4	3	184	1
ch	89	135	12	516	11	17	315	2	65	2	6	15	0
qy	368	37	89	154	89	11	81	5	28	2	6	19	0
qw	57	51	12	92	8	13	165	1	20	3	3	13	0
ac	59	55	21	159	6	6	2067	8	354	7	6	358	3
cl	448	12	18	7	11	1	12	8	48	2	7	37	0
ry	130	19	21	56	3	3	780	8	597	15	10	218	0
rn	27	5	5	23	0	1	173	1	146	13	4	85	0
rw	165	39	12	79	8	0	55	13	63	0	11	24	0
rd	63	5	29	9	4	0	246	1	137	7	3	662	3
un	10	0	12	5	3	0	60	1	46	1	0	43	1

5. CLASSIFICATION WITH TEXT FEATURES

We represented text features of each DA using a sparse bag-of-n-grams model. Our features included all unigrams, bigrams, and trigrams that appeared at least twice in the training set. We also had a feature for a unigram that was the only word in a dialog act.

Dialog acts in the training set would generate a set of candidate features. A dialog act with N words $w_1, \ldots, w_N$ would generate features $w_1, \ldots, w_N, w_{12}, w_{23}, \ldots, w_{N-1,N}$, $w_{123}, w_{234}, \ldots, w_{N-2,N-1,N}$ if $N > 1$ and would generate features w_1, w_1' if $N = 1$. The feature w' means that the dialog act consisted of just the word w. For example, if there was a dialog act "What is what" in the training set, it would generate the candidate features `what`, `is`, `what is`, `is what` and `what is what`. The dialog act "Right" would generate the candidate features `right` and `right'`. Other dialog acts in the training set would generate other candidate features.

Suppose F of these features occurred at least twice. Then each dialog act x in the training and test set would be represented by a $(F{+}1)$-dimensional feature $v = [v_1 v_2 \cdots v_{F+1}]^T$ where v_j, $j = 1, \ldots, F$ would be the number of times x generated feature v_j while v_{F+1} would be the number of times x generated a unigram feature that was not in F. This deals with the out-of-vocabulary problem, albeit crudely.

With this data and experimental protocol, F was between 9000 and 10000, depending on the data split. One of the benefits of SVMs, particularly with a linear kernel, is that they are well suited to dealing with high dimensional data, with fast training times (about ten minutes per split on a Linux box with 2 Intel Xeon 2.4 GHz processors and 2 Gb RAM), making it possible to investigate different kinds of features.

Applying a linear SVM to this resulted in 58.1% classification accuracy, and in 59.1% once Viterbi decoding was applied. The corresponding confusion matrix is in Table 5. This is much better than the 42.8% when using 1-nearest neighbors i.e. classifying each test DA with the label of the training DA with the highest cosine similarity [3]. While it is less than the 62.1% reported using Transformational Based Learning by Lager and Zinovjeva [11], their algorithm assumed knowledge of the previous dialog act.

Table 5. *Confusion matrix obtained when using text features with a linear SVM followed by Viterbi decoding. The classification accuracy is 59.1%*

	inst	ex	al	ch	qy	qw	ac	cl	ry	rn	rw	rd	un
in	1780	90	47	101	27	18	61	33	17	2	18	43	4
ex	162	591	17	93	35	11	84	22	21	20	26	20	2
al	119	29	382	54	44	8	327	9	9	2	7	80	0
ch	182	114	42	520	106	23	96	30	34	12	14	9	3
qy	66	27	39	129	550	15	30	8	8	5	11	1	0
qw	38	16	11	27	25	254	30	4	5	2	11	10	5
ac	49	42	86	53	13	8	2103	10	349	49	2	326	19
cl	332	60	12	54	8	8	35	44	11	3	33	5	6
ry	26	38	13	28	6	4	420	14	1234	3	11	59	4
rn	2	12	0	3	6	0	33	0	3	419	2	0	3
rw	99	87	8	27	10	6	23	16	18	7	146	17	5
rd	28	6	36	12	3	6	355	4	10	2	5	692	10
un	12	3	2	18	3	7	59	3	2	4	5	31	33

6. CLASSIFICATION WITH TEXT AND ACOUSTIC FEATURES

Recall that we have so far for each dialog act a G-dimensional dense vector representing its acoustic properties and a F-dimensional sparse vector representing its textual features. $F \sim 10000$ is much larger than $G = 49$. The easiest way of integrating them is to concatenate the two vectors to form a $F + G$-dimensional sparse vector and feed this to the SVM.

Like other classification algorithms, SVMs are not immune to numerical instability when the input data is not scaled. That said, they are usually very robust and converge in practice even when the input data is not scaled. (Convergence always happens in theory.) For example, here we combined acoustic features, which were scaled to between -1 and 1, and text features, which were raw counts of features in training data, and convergence was fast. On the other hand, convergence was very slow when the acoustic features were not scaled, even before we added text features.

With vector concatenation, classification accuracy was 61.8% using a linear SVM and 65.5% after Viterbi decoding. For more details, see the confusion matrix in Table 6. This compares with accuracies of 59.1% and 42.5% using text and acoustic features separately.

This is better than previously reported results that we know of for this dataset. Higher accuracy, such as 73.9% by Serafin and di Eugenio [3], has only been achieved by assuming knowledge of higher level discourse information such as game segmentation and game type for each DA.

A better sense of how the different features helped can be found in Table 7, which has precision, recall and F-scores for the cases when the acoustic and text features were used separately and together. The precision for a class, say `check`, is the fraction of DAs labelled as `check` that were actually `check`. Its recall is the fraction of real `check`s that were labelled as `check`s. The F score is the harmonic mean of precision and recall.

Unsurprisingly, all classes were better recognized with better precision using text than acoustic features. Bear in mind that we are using manually, not automatically, transcribed text features. Considering F-scores, acoustic features did not aid text features in recognizing binary questions, possibly because most `query-yn` DAs have helpful bigrams like “have you” or “do you” or “am i”. They help a little with recognizing complex queries, since though over 75% of most `query-w` DAs have the word “where”, “how” or “what”, so do other DAs. Acoustic features also help with recognizing questions that were `check` or `align`, but not with recognizing any replies. They did help with recognizing `ready`, `instruct` and `ack` DAs.

Table 6. *Confusion matrix obtained when using both acoustic and text features with a linear SVM followed by Viterbi decoding. The classification accuracy was 65.5%.*

	inst	ex	al	ch	qy	qw	ac	cl	ry	rn	rw	rd	un
in	1923	48	42	30	24	5	23	67	9	6	18	41	5
ex	122	598	36	120	32	12	64	31	22	21	23	18	5
al	126	24	587	34	49	7	71	15	9	2	8	137	1
ch	50	118	27	683	102	38	93	17	22	7	13	13	2
qy	35	35	54	154	534	18	16	19	3	5	8	6	2
qw	12	26	6	33	21	280	29	5	6	2	6	7	5
ac	19	54	51	52	10	8	2300	5	280	42	4	267	17
cl	356	36	19	23	5	3	7	87	15	0	40	18	2
ry	35	33	10	16	3	2	408	29	1282	6	10	22	4
rn	6	13	0	9	6	0	33	2	4	404	1	2	3
rw	87	89	9	30	7	3	12	26	23	9	153	15	6
rd	32	7	39	5	1	4	229	6	7	2	5	824	8
un	16	4	2	9	2	10	56	4	2	6	4	26	41

Table 7. *Precision, Recall, and F scores for each class using acoustic (A), text (T) or both (B) features. All values are percentages.*

Tag	Precision			Recall			F-score		
	A	T	B	A	T	B	A	T	B
instruct	47	61	68	86	79	86	61	69	76
explain	36	53	55	19	54	54	25	53	55
align	34	55	67	16	36	55	22	43	60
check	37	46	57	44	44	58	40	45	57
query-yn	42	66	67	10	62	60	16	64	63
query-w	19	69	72	3	58	64	5	63	68
ack	49	58	69	66	68	74	56	62	71
clarify	11	22	28	1	7	14	2	11	19
reply-y	35	72	76	32	66	69	34	69	72
reply-n	21	79	79	3	87	84	5	83	81
reply-w	16	50	52	2	31	33	4	38	40
ready	37	54	59	57	59	70	44	56	64
uncodable	13	35	41	1	18	23	1	24	29

7. OTHER EXPERIMENTS

7.1. Online Testing

The acoustic features we used were normalized by conversation side, which requires that one needs to see the entire dialog before classifying any dialog act. This makes online testing impossible.

We therefore investigated the possibility of not normalizing the acoustic features. Scaling was still done, of course, but this can be applied online to individual test examples. The classification accuracy was then 42.4% and 65.3% using acoustic and acoustic+text features respectively, an absolute drop of only 0.1% and 0.2% respectively from the non-normalized case.

Since all other parts of our algorithmic framework are online in the test phase (including the Viterbi algorithm, since it is only a forward pass), this algorithm is suitable for online testing.

7.2. Preprocessing Text with PCA

Serafin and di Eugenio [3] suggest using Principal Components Analysis on the sparse text features before applying a nearest neighbors classifier. We therefore ran a linear SVM on the first 100 principal components of the sparse text features and obtained classification accuracy of 55.7%, compared to 58.1% without using PCA.

7.3. SVM with a RBF kernel

In many domains, RBF SVMs (after model selection) work significantly better than linear SVMs. Surprisingly, RBF SVMs, even with model selection, only obtained classification accuracies of 53.6% and 36.8% on text and acoustic features separately, compared to 58.1% and 41.4% respectively with a linear SVM.

This may be because the RBF parameter σ is applied to all features, instead of having a different σ for each feature. The features here are quite different and may need different σ's. Scaling is not the issue, at least for the acoustic data, as we scaled all acoustic features to between -1 and 1.

During model selection, we consistently found that the best value of σ for both text and acoustic features, was 4, which is high enough to suggest that the decision boundary for the RBF SVM approached that of the linear SVM. This agrees with the note by Shriberg et. al [1] that “complex combinations of features (as far as the network could learn them) may not better predict DAs for the task than linear combinations of our input features”, although they were referring to a neural network.

8. CONCLUSION

In this paper, we showed that support vector machines can easily integrate text and acoustic features, and that their outputs can be input to a hidden markov model, resulting in an algorithm that can be applied online to new data.

Our acoustic features improved the quality of recognition for a few classes, namely instructions, acknowledgements, and all queries other than binary ones.

Future work will investigate the use of better prosodic features, Multiple Kernel Learning [17][18] to integrate different features, SVMs that directly incorporate sequential information [19], and methods for DA segmentation.

9. ACKNOWLEDGEMENTS

Funding for this project came from NSF Grant 0414919. We are very grateful to the Dialogue Group at the Human Communication Research Centre in Edinburgh for creating and releasing the Maptask corpus, and to Chih-Jen Lin and Chih-Chung Chang for their LIBSVM toolkit [7] used for our experiments. Nelson Morgan provided the source code to compute enrate parameters. Thanks also to Gunnar Rätsch, Alexander Zien, and Arthur Gretton for very helpful discussions, and to Benhard Schölkopf for his hospitality at the Department of Empirical Inference at the Max Planck Institute for Biological Cybernetics in Tübingen, where the first author did part of the work for this paper.

10. REFERENCES

[1] Elizabeth Shriberg, Rebecca Bates, Andreas Stolcke, Paul Taylor, Daniel Jurafsky, Klaus Ries, Noah Coccaro, Rachel Martin, Marie Meteer, and Carol van Ess-Dykema, "Can prosody aid the automatic classification of dialog acts in conversational speech?," *Language and Speech*, 1998.

[2] Andreas Stolcke, Klaus Ries, Noah Coccaro, Elizabeth Shriberg, Rebecca Bates, Daniel Jurafsky, Paul Taylor, Rachel Martin, Carol van Ess-Dykema, and Marie Meteer, "Dialogue act modeling for automatic tagging and recognition of conversational speech," *Comp. Linguistics*, vol. 26, pp. 339–373, 2000.

[3] Riccardo Serafin and Barbara di Eugenio, "FLSA: Extending latent semantic analysis with features for dialogue act classification," in *Proceedings of the Fortieth Annual Meeting of the Association for Computational Linguistics*, 2004, pp. 692–699.

[4] Corinna Cortes and Vladimir Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[5] Jean Carletta, Amy Isard, Stephen Isard, Jacqueline C Kowtko, Gwyneth Doherty-Sneddon, and Anne H Anderson, "The reliability of a dialog structure coding scheme," *Comp. Linguistics*, vol. 23, pp. 13–31, 1997.

[6] Chih-Wei Hsu and Chih-Jen Lin, "A comparison of methods for multi-class support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, pp. 415–425, 2002.

[7] Chih-Chung Chang and Chih-Jen Lin, "LIBSVM : a library for support vector machines," 2001.

[8] Bernhard Schölkopf and Alexander J. Smola, *Learning with Kernels*, MIT Press, Cambridge, MA, 2002.

[9] John Shawe-Taylor and Nello Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004.

[10] Paul Taylor, Simon King, Stephen Isard, and Helen Wright, "Intonation and dialog context as constraints for speech recognition," *Language and Speech*, vol. 41, pp. 489–508, 1998.

[11] Torbjorn Lager and Natalia Zinovjeva, "Training a dialogue act tagger with the μ-tbl system," in *Third Swedish Symposium on Multimodal Communication, Linkoping University Natural Language Processing Laboratory (NLPLAB)*, October 1999.

[12] Andrew Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–267, April 1967.

[13] Ting-Fan Wu, Chih-Jin Lin, and Ruby C. Weng, "Probability estimates for multi-class classification for pairwise coupling," *Journal of Machine Learning Research*, vol. 5, pp. 975–1005, 2004.

[14] Gunnar Rätsch and Stefan Sonnenburg, *Accurate Splice Site Prediction for Caenorhabditis Elegans*, pp. 277–298, MIT Press, Cambridge, MA, USA, 2004.

[15] Nelson Morgan, Eric Fosler, and Nikki Mirghafori, "Speech recognition using on-line estimation of speaking rate," in *Proc. Eurospeech '97*, Rhodes, Greece, 1997, pp. 2079–2082.

[16] Kåre Sjölander, "The snack sound toolkit," 1997.

[17] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan, "Multiple kernel learning, conic duality, and the smo algorithm," in *ICML '04: Twenty-first international conference on Machine learning*, New York, NY, USA, 2004, ACM Press.

[18] Stefan Sonnenburg, Gunnar Rätsch, and Christin Schäfer, "Learning interpretable SVMs for biological sequence classification," in *RECOMB 2005, LNBI 3500*. 2005, pp. 389–407, Springer-Verlag Berlin.

[19] Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann, "Hidden markov support vector machines," in *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.