Using One-Class Classification Techniques in the Anti-phoneme Problem

Gábor Gosztolya, András Bánhalmi, and László Tóth

MTA-SZTE Research Group on Artificial Intelligence
of the Hungarian Academy of Sciences and University of Szeged
H-6720 Szeged, Aradi vértanúk tere 1., Hungary
{ggabor,banhalmi,tothl}@inf.u-szeged.hu

Abstract. In this paper we focus on the *anti-phoneme modelling* part of segment-based speech recognition, where we have to distinguish the real phonemes from anything else which may appear (like parts of phonemes, several consecutive phonemes and noise). As it has to be performed while only having samples of the correct phonemes, it is an example of one-class classification. To solve this problem, first all phonemes are modelled with a number of Gaussian distributions; then the problem is converted into a two-class classification task by generating counter-examples; this way some machine learning algorithm (like ANNs) can be used to separate the two classes. We tested two methods for a counter-example generation like this: one was a solution specific to the anti-phoneme problem, while the other used a general algorithm. By making modifications to the latter to reduce its time requirements, we were able to achieve an improvement in the recognition scores of over 60% compared to having no anti-phoneme model at all, and it performed considerably better than the other two methods.

Keywords: speech recognition, one-class classification, counter-example generation, Artificial Neural Networks, Gaussian Mixture Models.

1 Introduction

One-class classification is an area of Artificial Intelligence where the task is to characterize one given class to distinguish it from anything else [1]. In this area examples of just this class are given; thus, in contrast with the conventional classification problem, there are no examples from any other class. An area where this kind of problem arises is the segment-based approach of speech recognition, where we have to determine whether speech segments correspond to a correct phoneme or not. To do this, we have a large number of examples of correct segments in the form of a hand-labelled corpus, but there are no given counter-examples which contain anything else that could occur in a sound recording (various noise, segments longer or shorter than one phoneme, etc.). These excerpts are called “anti-phonemes” [2] and the whole problem is called the “anti-phoneme problem”.

This problem can be solved in three entirely different ways: we can use a tool which models all the positive examples as a distribution, we can generate counter-examples in a task-specified way (using excerpts of multiple phonemes) and separate the two classes via some machine-learning method, or we can utilize a general counter-example generator algorithm (and then use the same machine-learning method). One good aspect about this problem is that the results will not be mere classification scores: a good one-class modelling method will lead to an improvement in a real application (in the accuracy of speech recognition).

2 Segment-Based Speech Recognition

In the speech recognition problem we are given some speech signal A and a list of possible words W, and our task is to find the most probable word $\hat{w} \in W$ via

$$\hat{w} = arg \max_{w \in W} P(w|A). \tag{1}$$

Using Bayes' theorem and noting that $P(A)$ is the same for all w's, we have that

$$\hat{w} = arg \max_{w \in W} P(A|w)P(w). \tag{2}$$

Now there are two distinct factors: the first describes the relation between the word and the speech signal, while the second simply states how probable the given word is. We will consider $P(w)$ as given, supplied by some *language model*, and concentrate on $P(A|w)$. In the segment-based approach we will assume that the signal A can be divided into non-overlapping segments, each of which corresponds to one of the o_j phonemes of the word $w = o_1, \ldots, o_n$. As the correct segmentation of A is not known, it appears as a hidden variable S:

$$P(A|w) = arg \max_{s \in S} P(A, s|w). \tag{3}$$

There are several ways of decomposing $P(A, s|w)$ further, depending on our modelling assumptions. What is common in all the derivations is that they trace the global probability back to the probabilities associated with the segments. The segments are usually assumed to be independent, so the corresponding local probability values will simply be multiplied. Glass et al. employ the formula [2]

$$\prod_{j=1}^{n} \frac{P(A_j|o_j)}{P(A_j|\alpha)} P(s_j|o_j), \tag{4}$$

where $P(s_j|o_j)$ is a duration model, A_j is the feature set extracted from the jth segment, and α denotes the "anti-phoneme" – a class that covers all the possible signal samples that are not real phonemes. Tóth et al. propose the formula [3]

$$\prod_{j=1}^{n} \frac{P(o_j|A_j)P(\overline{\alpha}|A_j)}{P(o_j)}, \tag{5}$$

where $P(\overline{\alpha}|A_j)$ denotes the probability that the given segment is *not* an anti-phoneme. The main difference between the two models is that in the former the acoustic observations are conditioned on the class labels (or the anti-phone), while in the latter it is the other way round. So in practice the components of the first formula are modelled by generative techniques, while discriminative ones are more straightforward for the latter. But as the posterior and class-conditional probabilities can always be easily converted to each other by Bayes' formula, these derivations do not limit us when choosing the machine-learning algorithm. In this paper we will focus on the anti-phoneme component $P(A_j|\alpha)$ or $P(\overline{\alpha}|A_j)$.

3 The Anti-phoneme Problem

Now we have examples for real phonemes, and we want to somehow distinguish them from any other speech segments that might appear. There are two main approaches for solving such a one-class classification problem: using a method which can model all these examples, or taking the actual occurrences of phonemes as positive examples and somehow creating anti-phonemes as negative ones. Then these two classes can be separated by classification methods like Artificial Neural Networks (ANNs) [4] or Support Vector Machines (SVM) [5]. But we have no training examples for the anti-phonemes, thus this generation is not trivial. We shall describe a solution for one-class modelling, and two approaches for automatic counter-example generation: a speech recognition-specific method [3] and the use of a general-purpose algorithm [6]. For the latter we also propose some modifications which have a surprisingly good effect on its running speed.

3.1 Modelling All Phonemes with Gaussians

Perhaps the most straightforward idea for describing all phonemes is to convert their occurrences to a probability distribution over the feature space, and model them with the sum of Gaussian curves. This is what Gaussian Mixture Models (GMM) [7] do: after the feature extraction part a clustering is performed on the set of resulting d-dimensional vectors (the positive examples) to divide them into n distinct subsets. Then a d-dimensional Gaussian is placed over every subset by calculating the mean and variance values of its elements.

3.2 The Incorrect Segment Sampling Algorithm

Tóth introduced a method for generating "incorrect" segments [3], based on the idea that the negative examples are probably parts of speech with incorrect segmentation bounds (they commence and/or end at positions where there is no real bound between phonemes). If we know the real phonetic boundaries – which is the case for any training database –, then it is easy to generate negative examples by choosing incorrect phoneme boundaries for the start and/or end segment bound. In the actual solution six anti-phonemes are generated for each phoneme by placing one or both phoneme boundaries earlier or later by δ milliseconds. In this method choosing the counter-examples is done *before* feature extraction.

3.3 Using General Counter-Example Generation

Another option is to use a general counter-example generation algorithm: in this case we take all our examples (all the phonemes *after* the feature extraction part) as the elements of one class, and generate a number of counter-examples. The input will be a set of d-dimensional vectors (X, $|X| = N$), while the output will be also a set of d-dimensional vectors somehow representing the opposite of our examples. Bánhalmi introduced such a counter-example generation algorithm [6], which will be briefly described in the following. The main idea here is to project each positive example $x \in X$ outside X. To do this, first the set of boundary elements is calculated, then each positive example is projected beyond the closest boundary point, producing N negative examples for N positive ones.

Determining the Boundary Points. First the boundary points B of the original example set are calculated ($B \subseteq X$). For an $x \in X$ first we place the k closest points into a set K, then an approximated center vector x_{center} is calculated from these elements: for each dimension, the lowest and the highest coordinates are added up after subtracting the appropriate coordinate of x from both of them. Then, for each $x_i \in K$, we calculate

$$\cos(\varphi_i) = \frac{(x_i - x)^T x_{center}}{\|x_i - x\| \, \|x_{center}\|}. \tag{6}$$

If all these values are nonnegative – so the angles between the vectors to the k nearest neighbors and the center vector are acute angles –, then x is added to the set of boundary points B. As this condition is only a sufficient one for being a boundary point, this method supplies only a subset of the real boundaries. An exact solution for this task is also given by attempting to separate x from the elements of K via an SVM [5]. If it is successful, x is a boundary element; but as this process is very slow, we used the one described above.

Projecting Beyond the Closest Boundary Point. After the set of boundary points B has been calculated, each element $x \in X$ is projected beyond the closest boundary point $x_b \in B$, resulting in a new point (hopefully) outside the region of positive examples. Besides x and x_b, it uses the center vector for x_b obtained earlier. There are two further parameters: *dist* sets the distance between the new point and the boundary point, while *curv* controls the curvature of the resulting hyper-surface. Fig. 1 shows a few examples with different *dist* and *curv* values.

Is It Really an Outlier? When the new point is determined, finally we check to see whether it is indeed an outer point, which is done in the same way as the boundary points were detected. If it is not an inner point, then it is added to the set of counter-examples; otherwise x_b is removed from the set of correct boundaries, and the whole transformation process has to be repeated with the now-closest boundary point $x_b{}' \in B$. In the end this algorithm will generate N counter-examples for a positive dataset of N data samples: one negative example for each positive one.

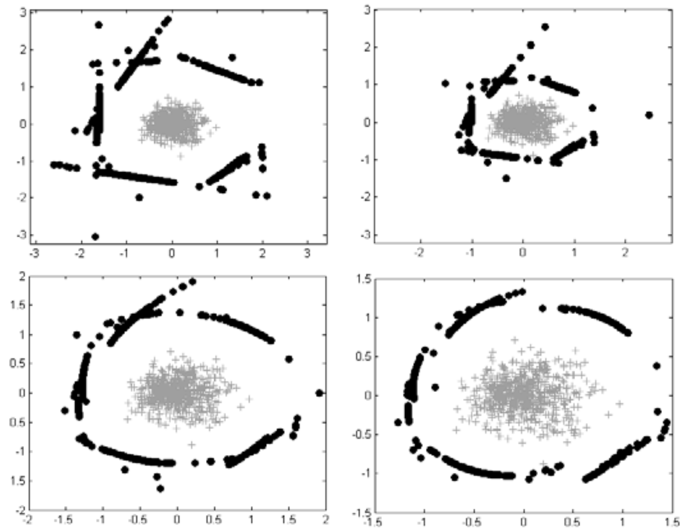

Fig. 1. Some counter-examples with different settings. Left to right: 1st: $(dist, curv) = (1, 0)$, 2nd: $(dist, curv) = (0.4, 0)$, 3rd: $(dist, curv) = (1, 0.5)$, 4th: $(dist, curv) = (1, 1)$.

Suggested Improvements of the Method. The obvious weakness of this algorithm is its time requirement: it is $o(kN^2 \log N)$ for the boundary point detection and $o(|B|Nk)$ for the projection for N positive examples and k neighbors. In our case large datasets were used (so N was quite big), thus executing this algorithm with different *dist* and *curv* parameters took a long time: even for one configuration the counter-example generation ran for a week on our test machine, making thorough testing practically impossible.

We found, however, that this algorithm can be divided into two distinct parts: the first one calculates the boundary points and the center vectors, while the second one carries out the actual counter-example generation. Luckily the first part uses most of the CPU time, while the *dist* and *curv* parameters appear only in the second part. Thus, regardless of the number of parameter combinations tested, the first and much slower part has to be computed just once. But the second part could be divided up further: for each element $x \in X$, before doing the actual projection, first the closest boundary point has to be found. The varying parameters also have no effect on this procedure, hence it can be separated from the projection and computed just once (although strictly after determining B).

With these modifications we reduced the running time of this method for a parameter pair to one day. (Of course it involves only the projection of points.) But in the last part, after the projections, the resulting point is checked to see whether it is an outer point for the positive examples. It is done in the same way as in the first part (finding the k closest points, calculating $\cos(\varphi_i)$s, etc.), so it is rather slow; and it cannot be pre-calculated since it is done on the newly generated counter-examples. We found, however, that this check is not always needed: for larger *dist* values (in our tests $dist \geq 1.5$) the projected point lies too far from X to be an inner point. Omitting this condition leads to an overwhelming speed-up, causing the run to finish in just 15 minutes. Applying this last modification made testing a large number of parameter pairs possible.

4 Experiments

At this point, having described the problem and the methods we used, we turn to testing. We describe the speech recognition environment briefly, discuss the testing methodology of each method applied, then present the results obtained.

4.1 The Speech Recognition Environment

Testing was done within our OASIS Speech Laboratory, which, due to its module-based structure, is quite suitable for experimenting. Phoneme classification was done by an ANN with one hidden layer, using typical segment-based features: they were averages of the 12 MFCC values and their derivates over specific parts of the segment. These ANNs were trained on a large, general database: 332 people of various ages spoke 12 sentences and 12 words each. The task was sentence recognition on medical reports with automatically generated phoneme labelling and segmentation. A simple word 2-gram was used as the model, i.e. the likelihood of a word only depended on it and the previous word; the vocabulary size was around 2,500. The tests were carried out on 150 sentences; the performance was measured via the widely-used accuracy and correctness values. Using no anti-phoneme model led to scores of 88.17% and 88.53% for (word-level) accuracy and correctness, respectively.

4.2 Testing

Testing the Gaussian Mixture Models was a rather straightforward task. Treating all phonemes as members of just one class, we did tests with 10, 15 and 20 Gaussian components. Since the GMM training procedure is not deterministic due to the initial clustering part, we performed three tests for each configuration, and averaged their results. Testing the Incorrect Segment Sampling Algorithm was also straightforward: for each correct phoneme, all six anti-phonemes were generated, and a feed-forward ANN was trained on this set with 100 hidden neurons. During evaluation the value of the appropriate output neuron served as an approximation of $P(\overline{\alpha}|A_j)$. As the standard neural net training procedure is a nondeterministic one, we compensated for it by performing the training three times; then all three nets were tested and their performance score was averaged.

Testing the General Counter-example Generation Method was the most complicated part as we sought to test several *dist* and *curv* pairs. First the promising region of these parameters was determined by preliminary tests, then this region was examined more closely: we performed tests with $1.0 \leq dist \leq 4.0$ and $0.0 \leq curv \leq 0.4$, of course, applying the proposed speed-up modifications. For one such parameter pairing an ANN was trained, just like in the previous case.

Table 1. Test results for the GMM and the Incorrect Segment Sampling method

Method	Accuracy		Correctness	
	Value	RER	Value	RER
GMM with 10 Gaussians	91.05%	24.60%	91.48%	25.72%
GMM with 15 Gaussians	90.98%	24.01%	91.58%	26.59%
GMM with 20 Gaussians	91.12%	25.19%	91.80%	28.51%
Incorrect Segment Sampling	91.97%	32.35%	92.62%	35.66%
No anti-phoneme modelling	88.17%	—	88.53%	—

Table 2. Accuracy scores for different *dist* and *curv* parameter values for the general counter-example generation method, averaged from three tests. Notably high values are highlighted in **bold**. With no anti-phoneme model, it results in a score of 88.17%.

		dist						
		1.0	1.5	2.0	2.5	3.0	3.5	4.0
	0.0	**93.91%**	**95.58%**	93.19%	90.32%	90.44%	91.52%	**93.67%**
	0.1	**93.91%**	93.31%	90.80%	90.68%	90.08%	89.13%	87.69%
curv	0.2	90.56%	91.40%	90.92%	89.25%	91.88%	89.13%	89.37%
	0.3	91.64%	92.47%	90.20%	91.88%	**94.03%**	92.35%	89.73%
	0.4	91.87%	92.59%	**93.43%**	91.16%	90.80%	91.87%	91.28%

Table 3. Correctness scores for different *dist* and *curv* parameter values for the general counter-example generation method, averaged from three tests. Notably high values are highlighted in **bold**. With no anti-phoneme model, it results in a score of 88.53%.

		dist						
		1.0	1.5	2.0	2.5	3.0	3.5	4.0
	0.0	**94.26%**	**95.82%**	93.55%	90.68%	90.80%	91.88%	**94.03%**
	0.1	**94.15%**	93.67%	91.16%	91.04%	90.44%	89.49%	88.05%
curv	0.2	91.75%	92.35%	91.28%	89.73%	92.48%	89.49%	89.73%
	0.3	92.83%	93.67%	90.80%	92.47%	**94.38%**	92.71%	90.56%
	0.4	93.31%	93.91%	**94.02%**	91.63%	91.75%	92.35%	92.11%

4.3 Results

Table 1 shows our results using GMMs. The performance scores improved only slightly, but it is quite insensitive to the number of Gaussians used: a relative error reduction (RER) of about 25% was achieved. The Incorrect Segment Sampling Algorithm proved to be more effective (see Table 1 again): the RER scores of 32% and 35% are quite good. Although this method is a little more complicated, we recommend it in place of GMMs. Overall, the General Counter-example Generation Method yielded the best results (see tables 2 and 3). Hardly any *dist*-*curv* pair made the recognition scores worse, but usually performance remained at the level of GMMs. At certain points, however, even the method of incorrect segment sampling was significantly outperformed; with $dist = 1.5$ and $curv = 0.0$ we attained scores of 95.58% and 95.82% (accuracy and correctness, respectively), which values, coming from averaging three ANNs, cannot be due to mere chance. It means a surprisingly large (over 60%) score of relative error reduction compared to having no anti-phoneme model. Of course, testing numerous *dist-curv* pair values would not be possible without our proposed speed-up improvements, which are clearly useful in other tasks too. Lastly, we would like to stress that the actual *curv* and *dist* values are of little importance; it is the improvements in the recognition scores, which were achieved by setting them.

5 Conclusions and Future Work

In this study we investigated several strategies for tackling the anti-phoneme problem in segment-based speech recognition. As in this task we have to characterize the "phone-ness" of a part of speech knowing only correct phonemes, methods appropriate for one-class classification should be considered. Apart from the traditional GMM we tested two methods which generate counter-examples for the given examples: one especially designed for the anti-phoneme problem and one of a general type. By introducing speed-up modifications for the latter one, we achieved a big reduction in the error rates, significantly outperforming the other two techniques. This result justifies our efforts of applying this generation method, and our modifications could be used in other one-class learning tasks as well. There are other methods for one-class modelling like one-class SVM, as well as ones for separating the two classes of examples like SVM [5]; as they could also work well here, it will be the subject of a future study.

References

1. Tax, D.M.J.: One-class classification; Concept-learning in the absence of counter-examples. PhD thesis, Delft University of Technology (2001)
2. Glass, J.R.: A probabilistic framework for segment-based speech recognition. Computer Speech and Language 17(2), 137–152 (2003)
3. Tóth, L., Kocsor, A., Gosztolya, G.: Telephone speech recognition via the combination of knowledge sources in a segmental speech model. Acta Cybernetica 16(4), 643–657 (2004)
4. Bishop, C.M.: Neural Networks for Pattern Recognition. Clarendon Press, Oxford (1995)
5. Schölkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J., Williamson, R.C.: Estimating the support of a high-dimensional distribution. Neural Computation 13(7), 1443–1471 (2001)
6. Bánhalmi, A., Kocsor, A., Busa-Fekete, R.: Counter-example generation-based one-class classification. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) ECML 2007. LNCS, vol. 4701, pp. 543–550. Springer, Heidelberg (2007)
7. Duda, R., Hart, P.: Pattern Classification and Scene Analysis. Wiley & Sons, New York (1973)