Lecture 14: Characterizing the Speech Signal for Speech Recognition: Code Books

1. Vector Quantization

Once a distance metric is defined, we can further reduce the representation of the signal by vector quantization. This approach classifies each frame into one of N categories, each represented by canonical vector that is associated with a symbol in the **code book**. Once a frame is classified by a codebook, can be represented by a single symbol that indicates which code value it is closest to. Many speech recognition systems use code book representations, though not all. In a later lecture we will see an approach that uses the vectors directly and does not need a code book.

How many categories are needed to make a good code book? Ideally, we want enough elements to capture all the significant different types of spectra that occur in speech. You might think that we could base the code book on phonemes, and thus need only 40. But there are several reason why this is a bad idea. First, a code book category represents a short segment of speech, say 20 ms, while phonemes typically cover longer stretches, sometimes up to 500 ms. Also, the sound over the course of a single phoneme is not constant. Consider the signal over a stop - there is silent section, an onset of signal and voiced or unvoiced segment that follows. This is far too complex to represent as one canonical vector! To be effective, code books must be based on acoustic properties, and we will need to develop additional mechanisms to identify phonemes from a sequence of code book vectors. As a result, there is not necessarily a strong correspondence between any one code book vector and a particular phoneme. A single code book vector might play an important role in many different phonemes (e.g., a "silence vector would play a key role in all stops). In practice, good recognition systems have been built that use 256 vector code books, though better recognition can be obtained by larger code books such as 512 or 1024. Counterbalancing the added precision we get with a larger code book size is the problem of the added complexity in recognizing the phonemes from the larger code book. Specifically, the larger the code book that larger the amount of training data we would have to have to define the recognition system.

We could just make up our code book vectors, but it would be very unlikely that this set would produce a good recognition system. Rather, code book vectors are defined by training on a training corpus of speech that minimizes the **overall distortion**, which is the sum of each input vectors' distance from the code book vector that it is identified with. To see this, consider a 2-dimensional example. Figure 1a shows a plot of four input vectors (the squares) classified into two groups by two code book vectors (the circles). It also shows the distance between each input vector and its associated code book vector. Figure 1b shows the same four input vectors with a different pair of code book vectors. The accumulated distortion for the code book in 1a is 4.416, while the accumulated distortion for 1b is 5.416. Thus the code book in 1a is better by this measure. Note that if we evaluated code books by how evenly they divided up the space, we would prefer code book 1b. But dividing up the space evenly is unlikely to be a reasonable goal in itself as we would not expect the range of different speech sounds to be evenly distributed. It is much better to obtain code book vectors that minimize the distortion as measured by the distance function.

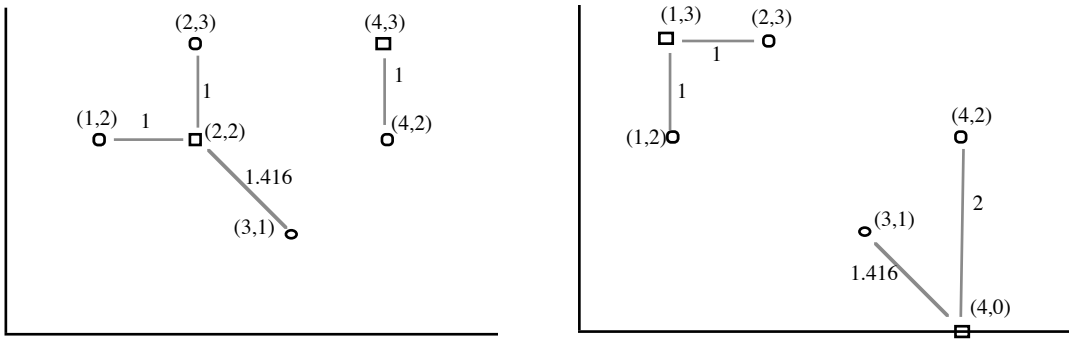

Figure 1: a. Code book Vectors: (2,2) and (4,3) b. Code book vectors: (1,3) and (4,0)

You may have noticed that there are better code book vectors for this example than shown in 1a. For instance, the (4,3) vector could be moved to (4,2), giving no distortion for that point and reducing the total accumulated distortion to 3.416. But an even better pair of code book vectors would be (1.5, 2.5) and (3.5, 1.5), which would give a total distortion of 2.8 (i.e., each distance is .7).

Given a specific code book, we can check if it can be improved by the following procedure.

For each code book vector, gather the training vectors that are closest to it.

Compute the **centroid** of this set, and make this the new code book vector.

The centroid is the value that minimizes the sum of the distances to each of a set of vectors. Thus this technique can only improve the overall distortion measure. If the code book vector already is the centroid, then this does not change the value. The method of computing the centroid differs depending on the distance measure used. For the Euclidean metric we are using in this example, the centroid is simply the mean vector, i.e., given a set of N vectors v_i, each of form ($v_i(1)$, ..., $v_i(k)$), then the centroid vector is

$$(\sum_{i=1,N} v_i(1))/N, ..., \sum_{i=1,N} v_i(k))/N)$$

For example, consider the three training vectors classified with the code book vector (2,2) in figure 1a: (2, 3), (1, 3), and (2, 1). The centroid is ((2 + 1 + 3) / 3, (2 + 3 + 1) / 2) = (2, 2). Thus it can't be improved. For the code book vector (1, 3) in Figure 1b we have the training vectors (1, 2) and (2, 3). The centroid of these two vectors is (1.5, 2.5), which does reduce the distortion.

These ideas can be generalized to produce an algorithm that iteratively improved the entire set of code book vectors, called the **K-means clustering algorithm**. Given an initial set of N code book vectors C_i and a set of training vectors, we do the following steps:

Classification: Cluster the training vectors by its closest code book vector according to the distance function;

Centroid Update: Update each code book vector to be the centroid (relative to the difference function used) of the training vectors assigned to it.

Iteration: If the improvement in overall distortion is greater than some threshold δ, then repeat from step 1.

Consider an example. We can start this algorithm with code book vectors that are randomly chosen from the training set. Consider this with six vectors shown in figure 2.

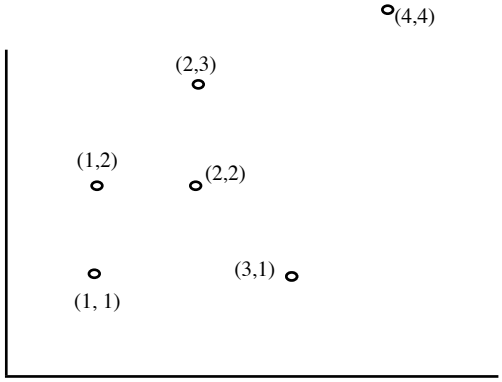

Figure 2: Six training vectors

Say we randomly pick (1,1) and (1,2) as the initial code book vectors. We then get the following clusters of points:

For (1,1): (1,1), (3,1)
For (1,2): (1,2), (2,2), (2,3), (4, 4)

The centroids of these sets give us the new code book vectors (2, 1) and (2.25, 2.75), and the overall distortion is reduced from 8.02 to 6.71.

Reclassifying the training set we now get the following clusters:

For (2,1): (1,1), (3, 1), (1,2)
For (2.25, 2.25): (2,3), (4, 2), (4, 4)

The centroids of these sets give us the new code book vectors (1.66, 1.33) and (3.33, 3) and reduce the overall distortion from 6.71 to 6.34.

Reclassifying the training set we now get the following clusters:

For (1.66, 1.33): (1, 1), (3, 1), (1, 2), (2,2)
For (3.33, 3): (2, 3), (4, 4)

The centroids of these sets give us the new code book vectors (1.75, 1.5) and (3, 3.5) and reduce the overall distortion from 6.34 to 5.94. If we reclassify the training set again we obtain exactly the same clusters and hence the estimates cannot be improved. In a realistic sized example, it would be very unlikely not to change the clusters with each iteration and the algorithm would continue until the improvement falls below the threshold set. Figure 3 summarizes the iteration steps and the values of the code book vectors at each iteration.

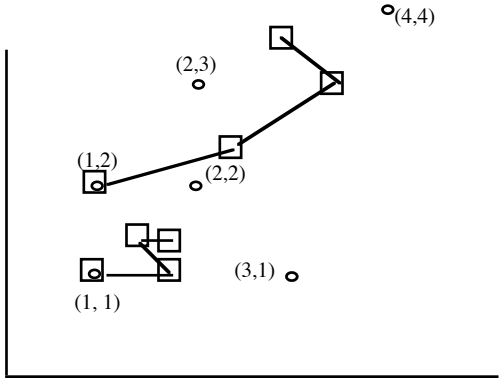

Figure 3: The refinement of code book vectors

While this algorithm works reasonably well most of the time, it is not guaranteed to find the optimal set of code books. It is a hill-climbing algorithm and may find a set of code book vectors that can't be improved using the procedure and yet are sub optimal. There are methods to help choose the initial codebook vectors to improve the chances of obtaining a good set. One technique resembles a binary search. It first computes the best codebook of size 1, then uses this to obtain the best of size 2, then 4 and so on. The best vector for a codebook of size one is simply the centroid of the entire training set. This point is then split by adding or subtracting some small increment I to the vector. The K-means iterative algorithm is then used to refine these estimates. Then the resulting best values are split by adding or subtracting the same increment to each and repeating the process. For example, with the above training set, the centroid is (2.17, 2.17). If we split using an increment of .05, we get two starting points for the 2 vector codebook of (2.22, 2,22) and (2.12, 2.12). After clustering and computing the centroid we'd get the new values of (1.75, 1.5) and (3, 3.5), which is the best answer obtained above. If we split again, we'd get four vectors (1.70, 1.45), (1.8, 1.55), (2.95, 3.45), and (3.05, 3.55). With only six vectors, however, there's not much point in continuing the example!

Consider a slightly better example. Figure 4 shows a plot of 35 instances of the F1 and F2 frequencies of five different vowel sounds (UW, UH, AO, AH, ER). The centroid of these 35 points is (532,1167).

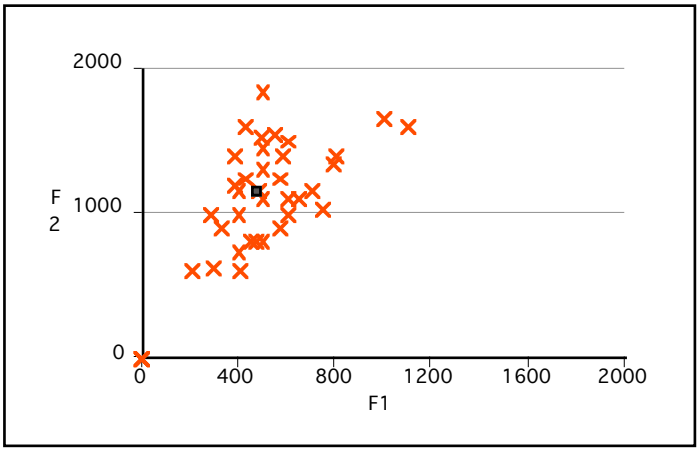

Figure 4: Plot of F1,F2 for 35 vowels showing centroid at (532, 1167)

If we split this into (531, 1166) and (533, 1168), they create two clusters of size 17 and 18. The centroids of these produce the codebook vectors (418, 919) and (639,1400), which after one iteration of the algorithm converges to (456, 926) and (620, 1452) and produces the partition shown in Figure 5.

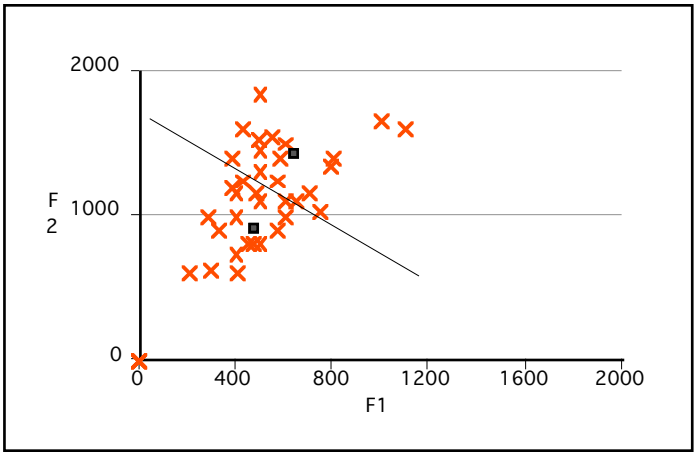

Figure 5: The 2-vector partition

If we split these points to (455, 925), (457, 927), (619, 1451), (621, 1453) and reclassify the training set, the centroids of the new clusters are (373, 762), (533,1074), (508, 1370), (764, 1557), which after five iterations produce the vectors (373, 762), (537, 1121). (505, 1535), (921, 1500). This partition consists of clusters of 9, 14, 8 and 4 training vectors respectively and is shown in Figure 6.

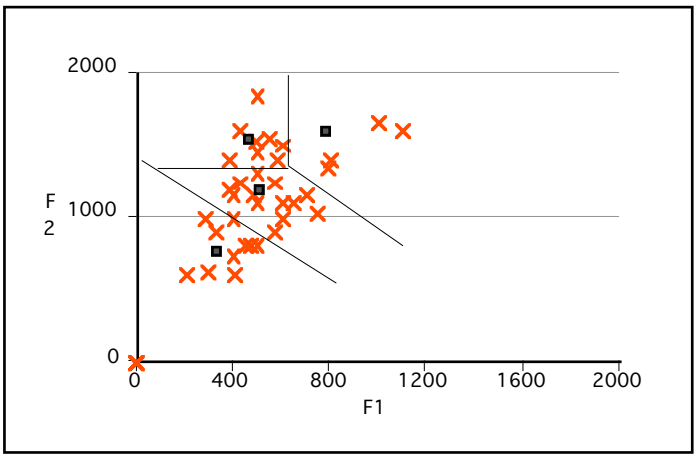

Figure 6: The 4 vector partitioning

If we now look at where each vowel in the original set falls, we see that the following classification:

(373, 762): 5 UW, 1 UH, 3 AO
(537, 1121): 2 UW, 4 UH, 4 AO, 3 AH, 1 ER
(505, 1535): 2 UH, 6 ER
(921, 1500): 4 AH

Note the cluster tends to gather together instances of the same vowel (e.g., 6 out of 7 ER's are in the third cluster, but UH is spread across three clusters). There are two observations to make

from this result. First, if we split the vectors another time to produce 8 code book vectors, we probably would get better association between clusters and particular vowels. But we will never find a set of clusters that uniquely identify the vowels as the ranges of the F1-F2 values for each vowel overlap. If we had a higher dimension representation we might get better discrimination, but some instances of different vowels will always be indistinguishable. In fact, vowels that are in unstressed syllables tend to start to all look alike, converging on a neutral vowel sound often called a "schwa".

2. Multiple Codebooks

Many systems, allow the input to be analyzed along different aspects, sometimes called streams. For instance, instead of using a vector of 21 elements (7 filter banks, seven delta, seven acceleration), it allows the input to be represented as 3 vectors:

7-element static coefficients
7-element delta coeff
7-element accel. Coeff

What effect could this have? The claim is that it gives more uniform coverage. Why? Major differences in one area might get smoothed by similarities in others. For instance, consider a case with 2 element vectors, producing a vector of size 9. The combined vector would be (f1, f2, d1, d2, a1, a2, e, de, ac). Say we have two instances of the same sound, but one instance was said when a door slammed, causing a significant spike in the delta coefficients. Say the vector without the noise was

V1 = (5, 8, 2, 1, 1, 0, 9, 2, 1)

and the vector with the door slam

V2 = (7, 10, 4, 6, 1, 0, 10, 4, 1).

Say we have two codebook vectors: A represented by (6, 7, 1, 1, 1, 1, 8, 2, 1) and B by (5, 5, 5, 5, 1, 2, 8, 1, 1). Using Euclidean distance, we find that the following distance measures

	A	B
V1	2.23	6.3
V2	7.2	6.9

As a result, V1 would be classified as an A, and V2 would be classified as a B.

Consider now the case where we have four streams, and vector V1 becomes V11 (first two values), v12 (the delta values, 3rd and 4th), v13 (the acceleration values, 5th and 6th), and V14 (the energy, delta energy and acceleration energy values, the 7th, 8th asnd 9th values), and we break up V2 similarly. Similarly, the codebook symbols A and B break up into A1, A2, A3, A4, B1, B2 B3, and B4. The distances are then as follows. In stream one, the inputs are (5,8) and (7, 10) and code A1 is (6, 7) and B1 is (5, 5). The distances are

	A1	B1
V11	1.4	3
V21	3.16	5.4

Thus, on stream one, both are classified as A1. Continuing for each stream, we get the classifications shown here:

Vector	Stream 1	Stream 2	Stream 3	Stream 4
V1	A1	A2	A3	A4
V2	A1	B2	A3	A4

With 4 streams, V1 and V2 are still classified by the same codebook symbols in 3 streams, and differ only on one. We could then use these complex values in an HMM by having each node emit four symbols rather than one at each time point, and the probability of a sequence would be some combination of the probabilities of each stream.

3. Doing Without Codebooks: Continuous Density Models

There is another way to build HMMs that does not rely on a codebook at all. Rather than each node having a probability distribution over a set of codebook symbols, the node has a probability distribution over the vectors themselves. This is called a **continuous density model**, and is often preferable because it avoids any errors that could be introduced in the quantization phase.
To implement this, we need to construct a probability density function over the vectors, which because each vector in the training data may be distinct, cannot be done directly.
The space of elements is just too large (theoretically is infinite if we allowed arbitrary numbers for each vector value). But even if we only allowed 256 values for each value, the number of possible n element vectors would be 256^n! Instead, we need to assume that the probability density function takes a certain form, in practice the Gaussian or Normal distribution, and then use the data to best estimate the parameters that define the distribution (namely the mean and variance). The formula for the Normal distribution is

$$N(x, \mu, \sigma) = 1 / (\mathrm{SQRT}(2\pi)*\sigma) \; * \; e^{-(x-\mu)^2 / (2\sigma^2)}$$

Where μ is the mean of the distribution and σ is the standard deviation. This is the classic Bell curve centered on μ with σ as a measure of "width". For instance, figure 7 shows two normal distributions, both with mean 5. The first (blue) has a standard deviation of 2, and the second a standard deviation of 1. Note that more probability mass is distributed further from the mean as the standard deviation gets larger.

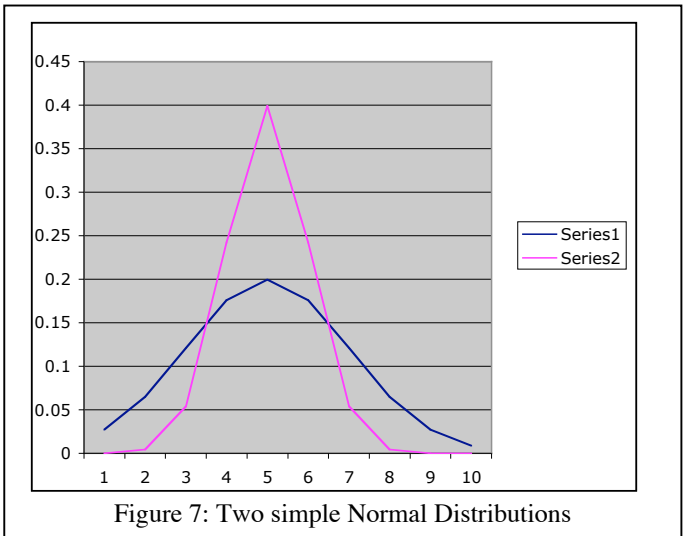

Figure 7: Two simple Normal Distributions

A Simple Guassian Model of Points

Consider the simplest example of using Guassian distributions as a representation using 1-element vectors (i.e., points). If the training data consists of the 2, 5, 9 and 10, then we can construct a normal distribution that maximizes the probability of these values by computing the mean and variance of the data using the formulas:

$\mu = \sum_i x_i / N$, where N is the number of samples
$= 26/4 = 6.5$

for the above four points. The formula for variance is

$\sigma = SQRT(\sum_i (x_i - m)^2 / N)$

Applying this to the four points, we get

$=SQRT(((2 - 5.3)^2 + (5 - 5.3)^2 + (9 - 5.3)^2 + (10-5.3)^2) / 4)$
$= SQRT(10.25)$
$= 3.2$

Using N(x, 6.5, 3.2) as the probability mass function, if we multiply the results for the four values we get a corpus mass of 3.2e-5. It can be proven that no other Normal distribution would assign a higher total probability to these four numbers.

Simple Guassian Mixtures

We could get a better approximation of this data if we allowed a more complex probability density function. For instance, we could assume that the pdf is defined by a linear combination of 2 Guassians, $N(x, \mu_1, \sigma_1)$ and $N(x, \mu_2, \sigma_2)$, i.e.,

$p(x) = \lambda_1 * N(x, \mu_1, \sigma_1) + \lambda_2 * N(x, \mu_2, \sigma_2)$ such that $\lambda_1 + \lambda_2 = 1$.

In practice, it is hard to estimate the lambdas, and so typically some prior, like the uniform distribution, is used. So then our formula would be

$p(x) = .5 * N(x, \mu_1, \sigma_1) + .5 * N(x, \mu_2, \sigma_2)$

We can use a generalization of the K-means algorithm to find two reasonable normal distributions that (locally) maximize the probability of the unseen data. Just by inspection, we might pick two clusters, 2 and 5 in one, and 9 and 10 in another. With these we'd get

$\mu_1 = 3.5$, $\sigma_1 = 1.5$ and
$\mu_2 = 9.5$, $\sigma_2 = .25$.

With distribution one, the probability of 2 is 0, 5 is 0, 9 is .48 and 10 is .48. With distribution two, the probability of 2 is .16, 5 is .16, 9 is .0003 and 10 is .00002.

Combing these two Guassians to obtain our probability distribution, we get:

$P(2) = .5 * 0 + .5 * .48 = .24$

P(5) = .5 * 0 + .5 * .48 = .24
P(9) = .5 * .16 + .5 * .0003 = .1603
P(10) = .5 * .16 + .5 * .00002 = .16.

As is easily seen, this new model greatly increases the probability of seeing the unseen data (consider the entropy calculation of (.05, .11, .09, .07) vs (.24, .24, .16, .16)).

Multivariate Guassian Models

To apply this technique to speech recognition, we need to generalize it to handle non-trivial vectors. One way to generalize this would be to model each dimension of the vector by its own Guassian. Thus, for a vector of size n, we would compute a mean μ_i and standard deviation σ_i for each of the n dimensions. Assuming independence of each dimension, the probability of a n-dimensional vector $\mathbf{v_1} = (v_{1,1}, v_{1,2}, \ldots, v_{1,n})$ would be

$P(v_i) = \prod_j N(v_{i,j}, \mu_j, \sigma_j)$

We can do a little better than this, and relax the independence assumption a bit. This involves computing a better variance model that captures correlations between different dimensions. This is captured by the **covariance matrix** defined by

$Cov(i, j) = \sum_k ((v_{k,i} - \mu_i) * (v_{k,j} - \mu_j))$

Let the matrix defined by the covariance be Σ. Note that the diagonals of this matrix are the variances of each dimension, i.e.,

$Cov(i,i) = \sigma_i^2$.

We can then define the multivariate normal distribution as

$N(v_i) = 1 / (SQRT(2\pi^n * |\Sigma|) * e$

|S| =