Data Quantization Optimized for Machine Learning Applications

Igor Fedorov
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Email: fedorov1@illinois.edu

I. Introduction

The goal of machine learning applications is to find a classification function, $y(x)$, for a test point $x \in \chi$, given a training dataset $\{(x^{(n)}, t^{(n)}), 1 \leq n \leq N\}$ in order to minimize some optimality criterion. In practice, the collection of data and feature extraction may not occur at the same time or in the same place as the application of the classification function. For instance, the user may want to store the collected data and perform classification later. If classification is computationally intensive, it may not be feasible to compute the classification function on the same device that was used to collect the data, so transmission of the collected data is required. In order to store or transmit data prior to classification, the data must first be quantized and then compressed. In this report, I will focus solely on the quantization process. I will assume that a classification function has been trained using unquantized data, but the testing data is quantized prior to being passed through the classifier. In other words, we test the classifier with some $\hat{x} = q(x)$. As such, we seek to design a quantizer, $q : \chi \rightarrow \chi$, which preserves the performance of the classifier y. In this report, I will study several different methods of designing the quantizing function q using clustering.

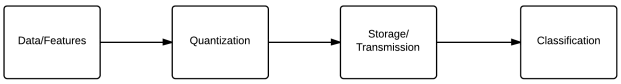

Fig. 1. Typical machine learning scenario

The approaches proposed in the following sections are all data clustering algorithms. Clustering and quantization share many fundamental characteristics. Both clustering and quantization define a mapping $q : \chi \rightarrow \chi$; both seek a codebook $B = R(q)$ (where $R(\cdot)$ is the range operator) which is, in some sense, optimal; both seek a partitioning of the input space into regions $\{R_k, 1 \leq k \leq C\}$ such that $\chi = R_1 \cup R_2 \cup \cdots \cup R_C$, where C, the number of clusters, can be allowed to grow to infinity in the case of data quantization. The main difference between data quantization, in the most general sense, and clustering is that clustering requires C to be finite and specified a priori.

II. Minimization of Empirical Information Loss Without Posterior PDF Estimation for Test Data

In [1], a general framework for performing feature quantization is presented. Let X be the original dataset of features, Y the set of labels, and K the quantized version of X. One way to measure the quality of the quantized dataset K is by measuring the information loss

$$L_1 = I(X;Y) - I(K;Y) \tag{1}$$

incurred from trying to estimate Y using K instead of X. The mutual information $I(X;Y)$ is defined as:

$$I(X;Y) = E_X[D(P_X, P)] \tag{2}$$

where P is marginal distribution of Y: $Y \sim P$. Since we do not have access to the marginal probability density of X, we can replace the expectation in (2) with an average over our dataset and compute the empirical mutual information, $\hat{I}(X;Y)$, as:

$$\hat{I}(X;Y) = \frac{1}{N}\sum_{i=1}^{N} D(\hat{P}_{X_i}||\hat{P}) \tag{3}$$

where N is the number of data points, $\hat{P}_{X_i} = \hat{P}(Y|X = X_i)$ is the empirical posterior probability of Y given $X = X_i$, $\hat{P}$ is the empirical marginal distribution of Y, and $D(\cdot||\cdot)$ is the Kullback-Leibler Divergence. Many of the design choices in using this algorithm come from how the empirical distributions are calculated.

The quantizer, $K(X_i)$, can now be defined as:

$$K(X_i) = \begin{cases} k & \text{if } X_i \in R_k \\ 0 & \text{else} \end{cases} \tag{4}$$

The mutual information $I(K;Y)$ is given by:

$$I(K;Y) = \sum_{k=1}^{C} p_K(k) D(P_{Y|K=k}||P) \tag{5}$$

where p_K is the marginal distribution of the codebook and C is the number of clusters. Once again, we do not have access to the true data generating distributions, so we can only estimate $I(K,Y)$ with $\hat{I}(K;Y)$:

$$\hat{I}(K;Y) = \sum_{k=1}^{C} \hat{p}_K(k) D(\hat{P}_{Y|K=k}, \hat{P}) \tag{6}$$

Now, we set $\hat{p}_K(k)$ to the fraction of observations which fall into R_k: $\hat{p}_K(k) = \frac{|R_k|}{N}$. $\hat{P}_{Y|K=k}$ is then set to the average of the posterior distributions of the observations which fall into R_k:

$$\hat{P}_{Y|K=k} = \frac{1}{|R_k|} \sum_{X_i \in R_k} \hat{\mathrm{P}}_{\mathrm{X_i}} \tag{7}$$

$$= \pi_k \tag{8}$$

So, we see that each region, R_k, is represented by a distribution π_k.

We now try to minimize the empirical loss of information:

$$L_2 = \hat{I}(X;Y) - \hat{I}(K;Y) \tag{9}$$

which reduces to:

$$L_2 = \frac{1}{N} \sum_{k=1}^{C} \sum_{X_i \in R_k} D(\hat{P}_{X_i} || \, \pi_{\mathrm{k}}) \tag{10}$$

Minimization of L_2 is a two-fold procedure, similar to the k-means algorithm. We find a new partitioning to minimize L_2, then we find the new π_k, and repeat the entire process until the solution converges.

Given a new partitioning $\{R_k\}_{k=1}^{C}$, we must find:

$$g = \arg\min_{\pi} \sum_{X_i \in R_k} D(\hat{\mathrm{P}}_{\mathrm{X_i}} || \pi) \tag{11}$$

The solution that $g = \pi_k$ is the unique minimizer of (11) is stated as an obvious fact in [1], but I will give a short proof here. We know that $D(p_1||p_2)$ is convex in the pair (p_1, p_2) [7]. Now, let's expand the quantity we are trying to minimize in (11):

$$\sum_{X_i \in R_k} D(\hat{\mathrm{P}}_{\mathrm{X_i}} || \pi) = D(\hat{P}_{X_{i_1}} || \pi) + \cdots + D(\hat{P}_{X_{i_{N_k}}} || \pi) \tag{12}$$

$$N_k = |R_k| \tag{13}$$

$$\{i_m\}_{m=1}^{N_k} = \{i : X_i \in R_k\} \tag{14}$$

Since we are trying to find the minimizer of (12), we can scale it by a constant without changing the solution:

$$g = \arg\min_{\pi} \frac{1}{|R_k|} \left(D(\hat{P}_{X_1} || \pi) + \cdots + D(\hat{P}_{X_{|R_k|}} || \pi) \right) \tag{15}$$

We now see that the quantity on the right in (15) is a convex combination, meaning:

$$\frac{1}{|R_k|} \left(D(\hat{P}_{X_1} || \pi) + \cdots + D(\hat{P}_{X_{|R_k|}} || \pi) \right) \tag{16}$$

$$\geq D\left(\frac{1}{|R_k|} \sum_{X_i \in R_k} \hat{P}_{X_i} \middle|\middle| \frac{1}{|R_k|} \sum_{X_i \in R_k} \pi \right) \tag{17}$$

$$= D\left(\frac{1}{|R_k|} \sum_{X_i \in R_k} \hat{P}_{X_i} \middle|\middle| \pi \right) \tag{18}$$

We can now use the information inequality [7], which states that $D(p_1||p_2) \geq 0$ with equality if and only if $p_1 = p_2$. Therefore

$$\frac{1}{|R_k|} \sum_{X_i \in R_k} D(\hat{P}_{X_i} || \pi) \geq 0 \tag{19}$$

where equality is achieved for $\pi = \pi_k$.

Now, given a set of $\{\pi_k\}$, we can find the optimal partitioning of the input space using:

$$R_k \ = \{X_i : D(\hat{\mathrm{P}}_{\mathrm{X_i}} \, || \, \pi_{\mathrm{k}}) \leq D(\hat{\mathrm{P}}_{\mathrm{X_i}} \, || \pi_j), j \neq k\} \tag{20}$$

By iterating between finding the optimal $\{\pi_k\}$ and the optimal $\{R_k\}$, we are guaranteed to find a local minimum of the objection function (10). The proof of this follows along the lines of the proof we gave for convergence of k-means. In other words, for a given $\{R_k\}$, the procedure we have given for calculating $\{\pi_k\}$ guarantees that (10) decreases (or stays the same). For a given $\{\pi_k\}$, re-assigning $\{R_k\}$ as specified here decreases (10) (or keeps it the same). Therefore, unless a local minimum has already been found, this iterative procedure is guaranteed to converge to a local minimum.

In the remainder of this section, I will consider several different implementations of this algorithm and report experimental results. Some of the main differences between the methods presented stem from how the posterior probability $\hat{\mathrm{P}}_{\mathrm{X_i}}$ is calculated and whether hard or soft clustering is desired.

A. Soft Clustering

The approach presented in [1] takes the view that a supervised clustering algorithm should not depend on having $\hat{\mathrm{P}}_{\mathrm{X_i}}$ for test data, since test data is unlabeled. As such, by assuming that the input space is a compact subset, the assignment criterion is modified to assign each X_i to the closest m_k, where distance is measured in the l_2 sense and m_k is the k'th centroid corresponding to π_k, as before. With this change, the resulting quantizer does not require test data to be labeled, but solving for the parameters of this quantizer remains a combinational optimization problem. As a result, a soft partitioning of the input space is introduced, where $w_k(X_i)$ represents the confidence of assigning X_i to R_k, with the constraint that $\|w(X_i)\|_1 = 1$ where $w(X_i) = [w_1(X_i) \cdots w_C(X_i)]^T$. A softmax-style form for the weights is adopted:

$$w_k(x) = \frac{e^{-\beta \|x - m_k\|^2/2}}{\sum_{j=1}^{C} e^{-\beta \|x - m_j\|^2/2}} \tag{21}$$

where β controls how soft the clustering is. The resulting cost function to be minimized is given by:

$$L_3 = \sum_{i=1}^{N} \sum_{k=1}^{C} w_k(X_i) D(\hat{\mathrm{P}}_{\mathrm{X_i}} \, || \, \pi_{\mathrm{k}}) \tag{22}$$

The update rules are then given by the following:

$$m_k^{(t+1)} = m_k^{(t)} - \alpha \sum_{i=1}^{N} \sum_{j=1}^{C} w_k(X_i) D(\hat{\mathrm{P}}_{\mathrm{X_i}} || \pi_j^{(t)}) \frac{\partial w_j^{(t)}(X_i)}{\partial m_k^{(t)}} \tag{23}$$

$$\frac{\partial w_j^{(t)}(X_i)}{\partial m_k^{(t)}} = \beta \left[1_{\{j=k\}} w_k(x) - w_k(x) w_j(x) \right] (x - m_k) \tag{24}$$

$$\pi_k^{(t+1)}(y) = \frac{\sum_{i=1}^{N} w_k^{(t+1)}(X_i) P(X_i)(y)}{\sum_{y'} \sum_{i=1}^{N} w_k^{(t+1)}(X_i) P(X_i)(y')}, \forall y \tag{25}$$

Given these update equations, the three remaining design decisions are how α is chosen, how β is chosen, and how $\hat{\mathrm{P}}_{\mathrm{X_i}}$ is calculated.

1) Selection of α Using Line Search: [1] suggests that α should be found using a line search, but details are not provided as to how the line search should be carried. I will present the details of the algorithm I used here. The idea behind a line search is to find the best $\alpha^{(t)}$ such that

$$m_k^{(t+1)} = m_k^{(t)} - \alpha^{(t)} d^{(t)} \tag{26}$$

minimizes $L_3(M^{(t+1)}, \Pi^{(t)})$ for a given search direction $d^{(t)}$, where $M^{(t+1)} = \{m_k^{(t+1)}, 1 \leq k \leq C\}$, $\Pi^{(t)} = \{\pi_{\mathrm{k}}, 1 \leq k \leq C\}$, and t is the iteration index. In effect, a line search is a one dimensional unconstrained optimization problem with the objective function given by [8]:

$$E(\alpha) = L_3(M^{(t+1)}(\alpha), \Pi^{(t)}) \tag{27}$$

where I have made explicit the dependence of L_3 on α through the dependence of $m_k^{(t+1)}$ on α. $E(\alpha)$ may not be globally convex, but we do know that it has local minima. Therefore, the first step is to find three points (a, b, c) satisfying $a < b < c$ such that $E(a) > E(b)$ and $E(c) > E(b)$, which is termed bracketing. The algorithm I have implemented to establish a bracket is a modification of the bracketing algorithm presented in [9]. The algorithm is summarized in Alg. 1. The idea is to randomly pick two points, a and b. Then, we establish a third point c which satisfies $E(c) < min(E(a), E(b)$. Then we shift the points (a, b, c) until a local minimum of $E(\alpha)$ falls in the range (a, b) or (b, c). One important edge case to consider was if $E(\alpha)$ is flat. To circumvent the issue of flat $E(\alpha)$, the bracketing algorithm is only allowed to run for a certain number of iterations and if $E(a) \approx E(b) \approx E(c)$, the bracket is deemed to have only one element:

$$\alpha_{flat} = \arg\min_{\alpha \in \{0, b\}} E(\alpha) \tag{28}$$

Once the bracket has been established, a local minimum of $E(\alpha)$ is found by iteratively shrinking the bracket. The shrinking algorithm uses two types of bracket reduction algorithms. The first is called the Golden section search [9]. The goal at each iteration is to pick two new points, d and e, between a and c. The new points are picked such that $w = |d - a| = |c - e|$,

Algorithm 1 Establishing bracket

Require: Loss function $E(\alpha)$, distance δ, maximum iteration count $maxiter$.

$a = rand(1)$
$b = a + \delta$
if $E(a) < E(b)$ **then**
  $c = a - \delta$
else
  $c = b + \delta$
end if
while $E(c) < \min(E(a), E(b))$ & $iter < maxiter$ **do**
  if $E(a) < E(b)$ **then**
    $c = a - \delta$
  else
    $c = b + \delta$
  end if
  $[a, b, c] = sort([a, b, c], \text{'ascend'})$
end while
if $E(a) \approx E(b) \approx E(c)$ **then**
  if $E(b) < E(0)$ **then**
    return b
  else
    return 0
  end if
end if
return $[a, b, c]$

where we have assumed that $|a - c| = 1$ (this assumption is valid since we can always scale w to reflect how large or small $|a - c|$ is). In order to ensure a constant reduction factor in the search interval, d and e are chosen to satisfy:

$$\frac{1 - w}{w} = \frac{w}{1 - 2w} \tag{29}$$

The only reasonable solution to (30) is $w = \frac{1}{2}(3 - \sqrt{5})$. Once d and e are chosen, the new bracket becomes (a, d, e) if $E(d) < E(e)$ and (d, e, c) otherwise. The best guess for the minimizer of E at each iteration is the middle point of the bracket. The algorithm is summarized in Alg. 2

Algorithm 2 Golden rule

Require: Loss function $E(\alpha)$, bracket (a, b, c).

$w = \frac{1}{2}(3 - \sqrt{5})$
$d = a + w|a - c|$
$e = c - w|a - c|$
if $E(d) < E(e)$ **then**
  New bracket, $[a', b', c']$, becomes $[a, d, e]$
else
  New bracket, $[a', b', c']$, becomes $[d, e, c]$
end if
return $[a', b', c']$

If the golden rule is followed, we can see that the bracket is guaranteed to decrease by a factor of $\frac{1}{1-w}$.

The second bracket shrinking strategy is called Brent's algorithm [8]. The idea is to assume that the error surface, $E(\alpha)$ is locally quadratic. We then fit a quadratic function to the three points $(a, E(a)), (b, E(b)), (c, E(c))$ using the regression approach we learned in class:

$$y = \begin{bmatrix} E(a) \\ E(b) \\ E(c) \end{bmatrix} A = \begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix} x = \begin{bmatrix} k_0 \\ k_1 \\ k_2 \end{bmatrix} \quad (31)$$

$$x = A^{\dagger} y \quad (32)$$

The resulting approximation for $E(\alpha)$ in the bracket (a, b, c) is given by:

$$E(\alpha) \approx k_0 + k_1\alpha + k_2\alpha^2 \quad (33)$$

Now, we can approximate the minimum of $E(\alpha)$ on the interval (a, b, c) by minimizing the quadratic form for of $E(\alpha)$ given in (33):

$$\arg\min_{\alpha \in (a,b)} \left(k_0 + k_1\alpha + k_2\alpha^2\right) = -\frac{k_1}{k_2} \quad (34)$$

A new point $d = -\frac{k_1}{k_2}$ is then chosen and the bracket is adjusted to (a, d, b) if $E(d) < E(b)$ and (d, b, c) otherwise. The algorithm is summarized in Algorithm 3.

Algorithm 3 Brent's algorithm

Require: Loss function $E(\alpha)$, bracket (a, b, c).

$y = [E(a), E(b), E(c)]^T$
$A = \begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix}$
$x = \begin{bmatrix} k_0 \\ k_1 \\ k_2 \end{bmatrix}$
$x = A^{\dagger} y = [k_0, k_1, k_2]^T$
$d = -\frac{k_1}{k_2}$
if $E(d) < E(b)$ **then**
    New bracket, $[a', b', c']$, $[a, d, b]$
else
    New bracket, $[a', b', c']$, $[d, b, c]$
end if
return $[a', b', c']$

The line search algorithm I used in my implementation is a hybrid of the Golden rule and Brent algorithms. Given a bracket $[a, b, c]$, a new point d is then chosen using Brent's algorithm. If the new point is reasonable ($a < d < c$ and $E(d) < E(b)$), then d is used to form a new bracket. If the new point is not reasonable, then the Golden rule is used to form a new bracket. This procedure is iterated until $|a - c| < \tau$, for some threshold τ. There is an edge case which should be mentioned. If the gradient descent algorithm has converged to a minimum, the line search should return $\alpha = 0$. Due to numerical errors, sometimes the line search does not recognize

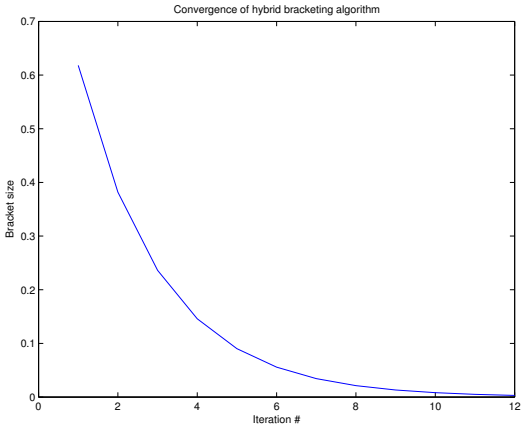

Fig. 2. Convergence of bracketing algorithm

that the algorithm has already converged and returns $\alpha >> 0$, which brings the algorithm far away from the local minimum. To prevent this behavior, I put in a safety net at the end of the line search which ensures that the minimizer of $E(\alpha)$ found by the line search achieves a minimum which is less than $E(0)$. The other edge case that must be considered if if $E(\alpha)$ is flat on the interval (a, c). In this case, if the bracket converges to a flat part of $E(\alpha)$, we terminate the bracket shrinking algorithm. The hybrid algorithm is summarized in Algorithm 4.

Algorithm 4 Hybrid line search

Require: Loss function $E(\alpha)$, bracket (a, b, c), threshold τ.

while $|a - c| > \tau \,\&\, ((E(a) - E(b))^2 + (E(c) - E(b))^2) > 1e - 3$ **do**
    $[a', b', c'] = BRENT([a, b, c])$
    if $E(b') < E(b)$ & $([a', c'] \in [a, c])$ **then**
        $[a, b, c] = [a', b', c']$
    else
        $[a, b, c] = GOLDEN([a, b, c])$
    end if
end while
if $E(b) < E(0)$ **then**
    return b
else
    return 0
end if

Fig. 2 shows bracket size as a function of iteration index for the hybrid bracketing algorithm for a real test setup. We can see that the bracket size decreases monotonically. Fig. 3 shows $E(\alpha)$ as a function of iteration index for the bracketing algorithm. The monotonic decrease in $E(\alpha)$ suggests that the bracketing algorithm is converging to a true local minimum of E.

2) β Selection Stragegies: [1] suggests two strategies for selecting β.

a) Fixed β: The easiest approach is to simply set β to a fixed value for the duration of the gradient descent algorithm.

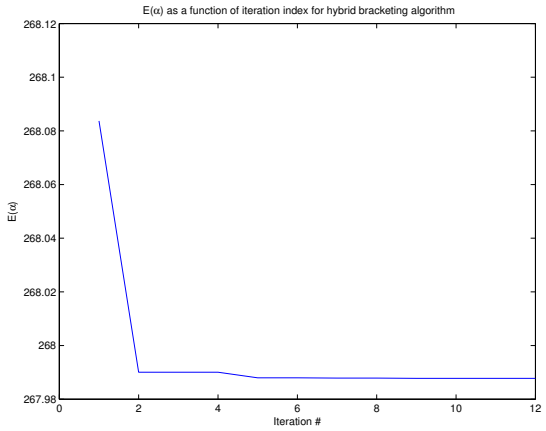

Fig. 3. $E(\alpha)$ as a function of iteration index for hybrid bracketing algorithm

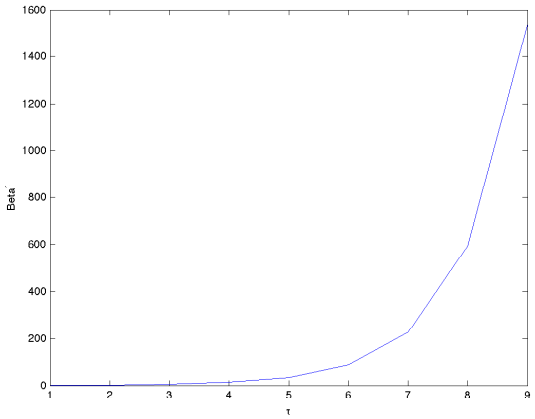

Fig. 4. β^{τ}

The approach suggested in [1] is

$$\beta = \frac{d}{\hat{\sigma}^2} \tag{35}$$

where d is the dimensionality of the input space and $\hat{\sigma}^2$ is defined as:

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{k=0}^{C}\sum_{X_i \in R_k^0} \|X_i - m_k^0\|^2 \tag{36}$$

where $M^0 = \{m_k^0, 1 \leq k \leq C\}$ is the set of centroids found by the k-means algorithm used to initialize the gradient descent.

b) Annealing: Since fixing β to a finite number results in a soft clustering algorithm which only approximates the hard clustering goal, annealing approaches have been suggested which progressively increase β [11]

$$\beta^0 = a \tag{37}$$

$$\beta^{(\tau)} = v\beta^{(\tau-1)}, v > 1 \tag{38}$$

At each step, τ, the entire gradient descent algorithm is run until convergence. In my experiments, I used 9 values of β, as shown in Fig. 4.

3) Calculation of $\hat{\mathrm{P}}_{\mathrm{X_i}}$*:* Several different non-parameteric approaches for calculating $\hat{\mathrm{P}}_{\mathrm{X_i}}$ are suggested in [1]. Namely, k-nearest neighbors and Parzen windows are suggested as valid ways of calculating $\hat{\mathrm{P}}_{\mathrm{X_i}}$ in [1]. I experimented with both of these approaches, one parameteric density estimation technique, and one neural network (NN) technique.

a) K-Nearest Neighbors (KNN): In the KNN approach, $\hat{\mathrm{P}}_{\mathrm{X_i}}$ is estimated for the training data by forming a histogram of the labels of the K nearest neighbors of each training point (including the point itself):

$$\hat{\mathrm{P}}_{\mathrm{X_i}} = \frac{1}{K}\sum_{j:X_j \in N_K(X_i)} \delta_{Y_j} \tag{39}$$

where $N_K(X_i)$ is the set of K nearest neighbors of X_i (where distance is measured in the l_2 sense), Y_j is the label for X_j, and δ_{Y_j} is the discrete Dirac-delta function $\delta(x - Y_j)$. The value of K is an important tuning parameter. In [1], $K = 10$ was used for all experiments. In my case, I experimented with several different values of K.

b) Parzen Windows: To estimate $\hat{\mathrm{P}}_{\mathrm{X_i}}$ using Parzen windows, we begin with Bayes's theorem:

$$\hat{p}(Y_i|X_i) = \frac{\hat{p}(X_i|Y_i)\hat{p}(Y_i)}{\sum_{k=1}^{C}\hat{p}(X_i|Y_k)\hat{p}(Y_k)} \tag{40}$$

We estimate $\hat{p}(Y_i)$ using:

$$\hat{p}(Y_i) = \frac{1}{N}\sum_{n=1}^{N}\delta_{Y_j}$$

To estimate $\hat{p}(X_i|Y_i)$, we employ the Parzen window approach:

$$\hat{p}(X_i|Y_i) = \frac{1}{N_i}\sum_{j:Y_j=i}\frac{1}{h^d}H\left(\frac{X_i - X_j}{h}\right) \tag{41}$$

where d is the dimensionality of the input space, H is the kernel function, and h is the width parameter. In my experiments, I used the Gaussian kernel function with diagonal covariance matrix, which leads to the following density estimate:

$$\hat{p}(X_i|Y_i) = \frac{1}{N_i}\sum_{j:Y_j=i}\frac{1}{(2\pi h^2)^{\frac{d}{2}}}e^{-\frac{\|X_i - X_j\|^2}{2h^2}} \tag{42}$$

The remaining design parameter is the choice of h. In the case of $d = 1$, we know that the optimal choice of h is given by:

$$h^* = 1.06\hat{\sigma}N^{-1/5}$$

where $\hat{\sigma}$ is the estimate of the standard deviation of the data and N is the number of data points. In the multivariate case, we can argue that the smoothing parameter should be different along different dimensions of the input space [10]. In effect, we would need to estimate a full covariance matrix for the data in order to reliably estimate the smoothing parameter for each dimension. As an alternative, we note that if the data had a unit covariance matrix, we could simply set h equal to to h^* for every dimension of the input space. We can achieve this by applying a simple linear transform to the data. Let

$X = [X_1, X_2, \cdots, X_N]$ be a matrix of N data points. Using the SVD, we can write X as:

$$X = U\Lambda V^* \tag{43}$$

where U is the matrix of left singular vectors of X, Λ is a diagonal matrix containing the singular values of X, and V is the matrix containing the right singular vectors of X. Let's see what happens when apply the linear operator $T(X) = \Lambda^{-1}U * X$:

$$Y = T(X) \tag{44}$$
$$= \Lambda^{-1}U^*X \tag{45}$$

where Λ^{-1} is a diagonal matrix whose entries are the singular values of X raised to the -1 power. Now, let's calculate the covariance matrix for Y:

$$YY^* = \Lambda^{-1}U^*X(\Lambda^{-1}U^*X)^* \tag{46}$$
$$=^1 \Lambda^{-1}U^*XX^*U\Lambda^{-1} \tag{47}$$
$$=^2 \Lambda^{-1}U^*U\Lambda V^*V\Lambda U^*U\Lambda^{-1} \tag{48}$$
$$=^3 \Lambda^{-1}\Lambda\Lambda\Lambda^{-1} \tag{49}$$
$$= I \tag{50}$$

where 1 uses the fact that $\left(\Lambda^{-1}\right)^* = \Lambda^{-1}$ since Λ is a diagonal matrix, 2 also uses the fact that Λ is a diagonal matrix, and 3 uses the fact that U and V are unitary matrices ($U^*U = UU^* = I$ and $V^*V = VV^* = I$). So, we see that Y has a unit covariance matrix. Therefore, the density estimation algorithm I used first learns a covariance diagonalizing linear transformation for each class in the learning dataset, applies the transform to each each class, and then computes the density estimate in the transformed vector space. The algorithm is summarized in Alg. 5.

Algorithm 5 Multivariate Parzen Window Density Estimation Algorithm

Require: Dataset $X = \{X_i, 1 \leq i \leq N\}$

1: **for** $1 \leq k \leq c$ **do**
2: $J^k = \{i : Y_i = k\}$
3: $X^k = [X_j], \forall j \in J^k$
4: $[U, S, V] = svd(X^k)$
5: $Y = S^{-1}U^*X^k$
6: $p(X_j|Y_j) = PARZEN(X_j, Y), \forall j \in J^k$
7: **end for**
8: **return** $p(Y_i|X_i) = \frac{p(X_i|Y_i)p(Y_i)}{\sum_{k=1}^{c} p(X_i|Y_k)p(Y_k)}$

c) Gaussian Mixture Model (GMM): The GMM approach to calculating $\hat{\mathrm{P}}_{\mathrm{X_i}}$ is similar to the Parzen window approach in that we start with Bayes' theorem (40) and estimate the class conditional densities $\hat{P}(X_i|Y_i)$ using a GMM. In other words, for each class k, we estimate $p(X_i|Y_i = k)$ as a mixture of M Gaussians:

$$p(X_i|Y_i = k) = \sum_{j=1}^{M} p(X_i|j_k)P(j_k) \tag{51}$$

where j_k represents the j'th Gaussian component for the k'th class.

I used the same EM algorithm to learn the GMM for each label as we learned in class. The only slight modification I was forced to make was to ensure that the covariance estimate, $\hat{\Sigma}$, at each iteration was positive definite by performing the following modification:

$$\hat{\Sigma} \leftarrow \hat{\Sigma} + \epsilon I \tag{52}$$

where ϵ is some small positive number.

d) NN: Neural networks naturally lend themselves to posterior density estimation with the correct choice of error criterion and output activation function [8]. Let t_k be defined as:

$$t_k^n = \begin{cases} 1 & \text{if } Y_n = k \\ 0 & \text{else} \end{cases} \tag{53}$$

Also, let y_k^n be the k'th output for input X_n of our NN. We would like y_k^n to represent $P(Y_n = k|X_n)$. Let y^n and t^n be defined as:

$$y^n = [y_1^b, y_2^b, \cdots, y_c^n] \tag{54}$$
$$t^n = [t_1^n, t_2^n, \cdots, t_c^n] \tag{55}$$

We can now write $p(t^n|X_n)$ as:

$$p(t^n|X_n) = \prod_{k=1}^{c} (y_k^n)^{t_k^n} \tag{56}$$

The likelihood of the entire dataset is then given by:

$$p(T|X) = \prod_{n=1}^{N} \prod_{k=1}^{c} (y_k^n)^{t_k^n} \tag{57}$$

If we form the negative log-likelihood, we find the objective function which we need to minimize to train the network:

$$L_4 = -\sum_n \sum_{k=1}^{c} t_k^n ln\left(y_k^n\right) \tag{58}$$

If we consider the fact that $t_k^n = \delta_{kl}$ where $Y_n = l$, we see that the absolute minimum of L_4 is obtained by setting y_k^n to its maximum possible value for $k = l$. Since y_k^n is a probability, the maximum value it can is 1. Consequently, the absolute minimum of L_4 is achieved for $y_k^n = t_k^n$ and is given by:

$$L_{4,min} = -\sum_n \sum_{k=1}^{c} t_k^n \ln y_k^n \tag{59}$$

If we subtract $L_{4,min}$ from L_4, we get a new error function, L_5, which is lowered-bounded by 0 and achieves a minimum of 0 for $y_k^n = t_k^n$:

$$L_5 = -\sum_n \sum_{k=1}^{c} t_k^n \ln\left(\frac{y_k^n}{t_k^n}\right) \tag{60}$$

Finally, we see that we can write L_5 in terms of the KLD:

$$L_5 = -\sum_n D(y^n||t^n) \tag{61}$$

In order to interpret y_k^n as a proper posterior probability $P(Y_n = k|X_n)$, we need to ensure that:

$$\sum_k y_k^n = 1 \tag{62}$$

$$y_k^n \geq 0, 1 \leq k \leq c \tag{63}$$

This leads us to use the logistic sigmoid activation function for the output nodes of the NN:

$$y_k = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'}} \tag{64}$$

where a_k is the activation for the k'th output node.

I implemented this approach in order to approximate $\hat{\mathrm{P}}_{\mathrm{X_i}}$ using a 2 layer NN. The only design parameter in my implementation was the number of hidden nodes, h. To test the influence of h on the quality of the resulting estimate of $\hat{\mathrm{P}}_{\mathrm{X_i}}$, I simulated various values of h and observed how they affect the objective function minimization. The results for the diabetes dataset are shown in Fig. 5. We can see that all of values of h have close to the same performance, so I chose $h = 5$ for the rest of my experiments.

For my experiments with the Human Activity Recognition (HAR) task, I had to modify the NN training algorithm slightly because of the high dimensionality of the search space. Whereas I used a fixed step size to train the NN for the diabetes dataset, I employed my line search algorithm to train the NN for the HAR dataset.

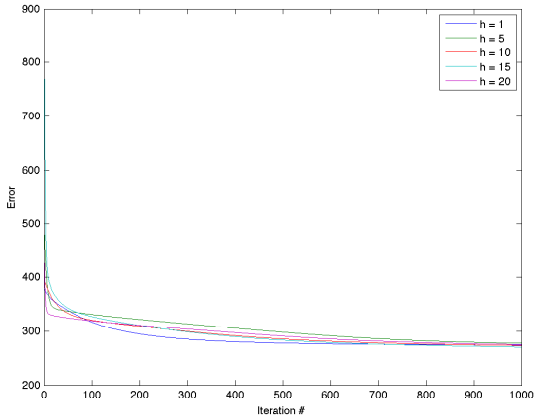

Fig. 5. Influence of h on quality of posterior estimation for the diabetes dataset using a NN

III. Diabetes Dataset Validation

I conducted experiments using the diabetes dataset to validate the ideas presented above. The diabetes dataset served as an easy to work with, low dimensional dataset. All testing was done in a 10-fold cross validation scheme. For each fold, 20% of the dataset was randomly selected to be the test set and the rest was left for the training set. The accuracy results were then averaged across all of the folds.

Two types of classifiers were used: a linear least squares classifier and a two layer neural network with tanh activation functions trained using the l_2 error function. For least squares linear classification, we approximate Y_i using:

$$\hat{Y}_i = WX_i \tag{65}$$

where the matrix W is calculated in order to minimize:

$$L_5 = \sum_{n=1}^{N} |Y_i - \hat{Y}_i| \tag{66}$$

The two layer neural network was also trained in a least squares framework in order to minimize:

$$L_6 = \sum_{n=1}^{N} \sum_{k=1}^{c} (t_k^n - \hat{t}_k^n)^2 \tag{67}$$

$$t_k^n = \delta(k - Y_n) \tag{68}$$

$$\hat{t}_k^n = g\left(\sum_{j=0}^{H} \tilde{w}_{kj} g\left(\sum_{i=0}^{D} w_{ji} X_i[i]\right)\right) \tag{69}$$

$$g(a) = tanh(a) \tag{70}$$

where c is the number of classes, H is the number of hidden nodes, and D is the dimensionality of the input space.

IV. Pose Codebook Learning for Human Activity Recognition

A more interesting validation experiment is to see how the supervised clustering approach performs with the task of human activity recognition using depth videos [12]. Let the depth video sequence be denoted by $\{D_t\}_{t=1}^N$, where $D_t \in \mathbb{R}^{(a \times b)}$. The first step is to perform skeletal tracking on the depth video sequence in order to generate a sequence of joint locations J_t^k, where $J_t^k \in \mathbb{R}^2$ and $1 \leq k \leq 15$. J_t^k represents the pixel coordinates of the k'th joint in the t'th frame. There are 15 joints: head, neck, left shoulder, left elbow, left hand, right shoulder, right elbow, right hand, torso, left hip, left knee, left foot, right hip, right knee, right foot. Skeletal tracking is performed using OpenNI open-source skeletal tracking software. The next step is to perform a second layer of feature extraction on the joint locations J_t^k. The second layer features are defined by the angles between the user's body parts, which are defined by the segments connecting the joints of the user. For instance, let J_t^1, J_t^2, J_t^3 be the locations of the user's neck, left shoulder, and left elbow, respectively, for the t'th frame. Then, we can see that the vector

$$v_1 = J_t^1 - J_t^2 \tag{71}$$

runs from the user's left shoulder to the neck. Likewise, we can see that the vector

$$v_2 = J_t^3 - J_t^2 \tag{72}$$

runs from the user's left shoulder to the left elbow. Now, we can determine the angle between v_1 and v_2 using the law of cosines:

$$\theta_1 = \arccos\left(\frac{v_1^T v_2}{\|v_1\|\|v_2\|}\right) \tag{73}$$

We can then form a vector $P_t \in \mathbb{R}^{11}$ which defines the pose of the user for the t'th frame using angles between the user's body parts. At this point, a pose code-book is learned in order to reduce the dimensionality of the pose feature space. In [12], k-means is used to learn a C cluster codebook. The pose code-book is then used in a bag-of-words framework in order to form a histogram of poses for a given test video. In other words, given a test video D_t, we represent the video as $U \in \mathbb{R}^C$ such that:

$$U(k) = \frac{1}{N}|\{P_t : P_t \in R_k, 1 \leq t \leq N\}| \tag{74}$$

where R_k is the k'th partition induced by the clustering operation. Finally, an SVM classifier is used to classify the resulting histogram U. The histogram intersection kernel, given by

$$K_{hist}(U^1, U^2) = \sum_{i=0}^{C} min(U^1(i), U^2(i)) \tag{75}$$

is a natural choice for this algorithm since each video is ultimately represented by a single histogram. Fig. 6 shows a high level overview of the algorithm.

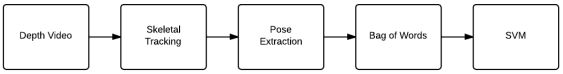

Fig. 6. Overview of activity recognition algorithm

For my purposes, the most interesting aspect of this algorithm is the code-book generation process. This algorithm does not fit into the framework shown in Fig. 1 because training the classifier requires clustering, so we cannot operate under the assumption that our classifier has been trained on perfect, unquantized data. Nevertheless, we can see if using supervised clustering can improve the activity recognition algorithm from both the accuracy and computational perspectives.

V. Diabetes Dataset Results

To test the algorithm, the first step was to ensure that my gradient descent implementation functioned properly. Fig. 15 shows the convergence of the gradient descent algorithm for various modes of the algorithm. For the fixed β flavor of the algorithm, the iteration index represents the iteration index of the gradient descent algorithm. For the annealing β strategy, the iteration number is the index of the β value used, where iterative gradient descent is performed every time a new β is set. We can see that for each variant of the algorithm, we have monotonic convergence of the gradient descent, which indicates that the line search algorithm functions properly and performs well for various types of posterior distributions.

Next, I experimented with different values of K for KNN posterior density estimation with the diabetes dataset. Fig. 17 shows various values of K as a function of iteration index for a 10-fold validation scheme. τ_1 and τ_2 are the tuning

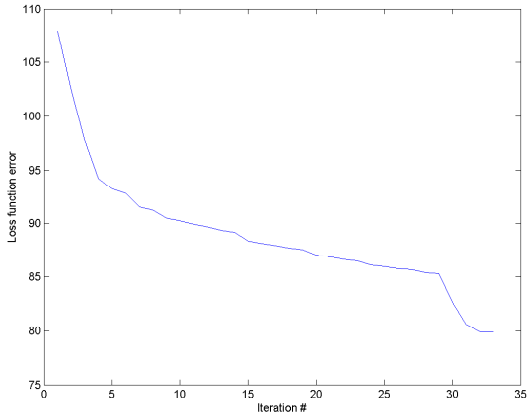

Fig. 7. Convergence of gradient descent for KNN posterior estimation using fixed β

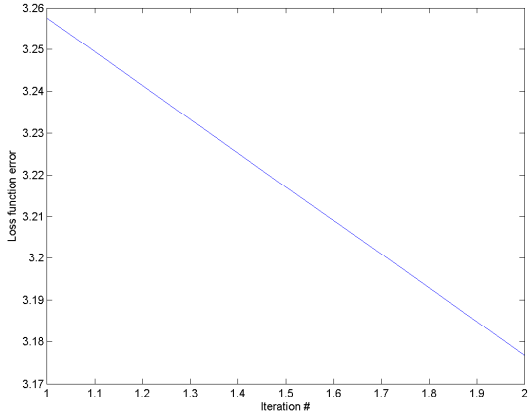

Fig. 8. Convergence of gradient descent for Parzen window posterior estimation using fixed β

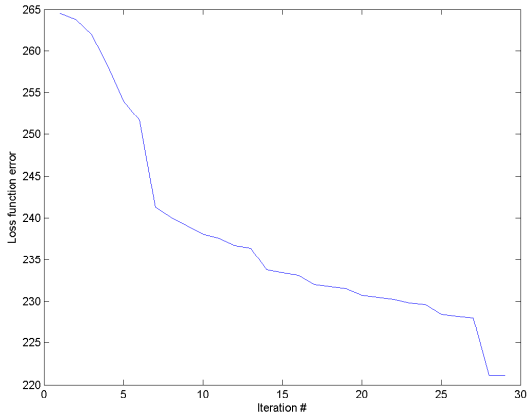

Fig. 9. Convergence of gradient descent for GMM posterior estimation using fixed β

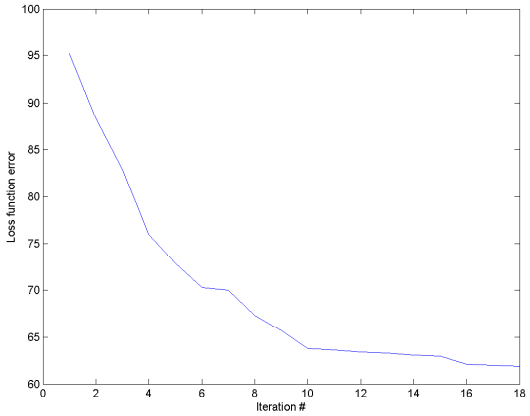

Fig. 10. Convergence of gradient descent for cross entropy posterior estimation using fixed β

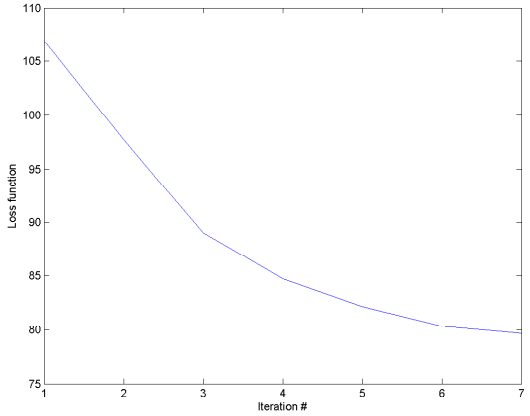

Fig. 11. Convergence of gradient descent for KNN posterior estimation using annealing β strategy

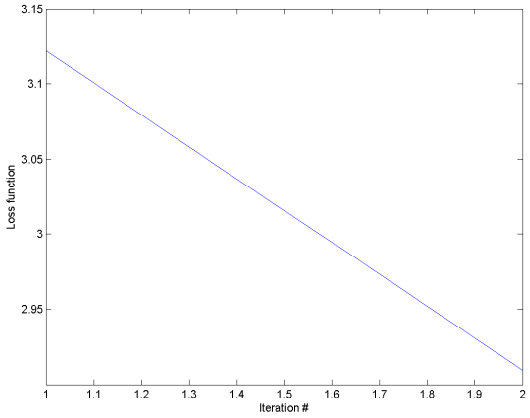

Fig. 12. Convergence of gradient descent for Parzen window posterior estimation using annelaing β strategy

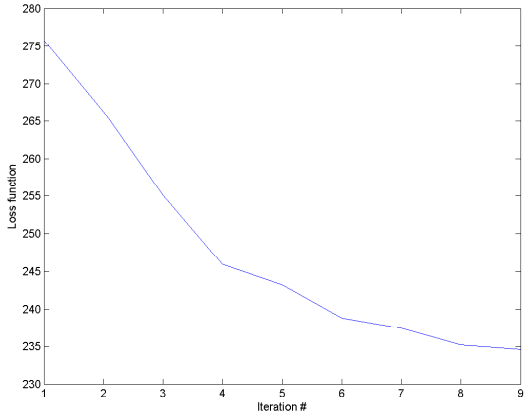

Fig. 13. Convergence of gradient descent for GMM posterior estimation using annealing β strategy

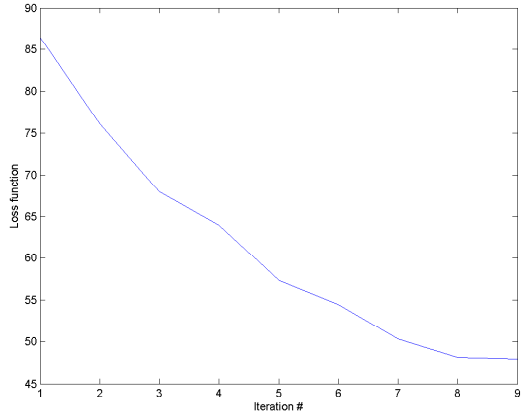

Fig. 14. Convergence of gradient descent for cross entropy posterior estimation using annealing β strategy

parameters for the gradient descent and line search algorithms, respectively. The gradient descent stops when

$$L_3^{(t-1)} - L_3^{(t)} < \tau_1 \tag{76}$$

where t is the iteration index. The line search algorithm stops when the bracket size becomes smaller than τ_2.

Fig. 16 through Fig. 19 show the performance of the clustering algorithms with the linear and NN classifiers, as compared to unclassified data and the k-means algorithm. The horizontal axis represents the number of clusters on a log scale. Although it is difficult to make any conclusive arguments about the influence of C on clustering with such a low dimensional dataset, we can make some observations. First, we can see that annealing does not have much of an effect on the overall effectiveness of the clustering, and in fact sometimes leads to worse results. Second, there seems to be a threshold for the cluster number, after which performance doesn't improve by increasing the cluster number. Third, none of the posterior pdf techniques seem to be best for all values of C, although for higher values of C KNN outperforms the other techniques across all experiments. Fourth, there is an interesting valley in classification performance for $C = 4$ across all of the

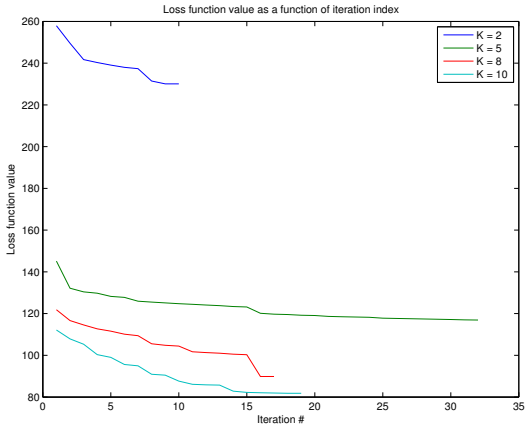

Fig. 15. Influence of K on loss function minimization with 10-fold cross-validation scheme and fixed β. $C = 10, \tau_1 = 0.1, \tau_2 = 0.005, \beta = \beta_0$

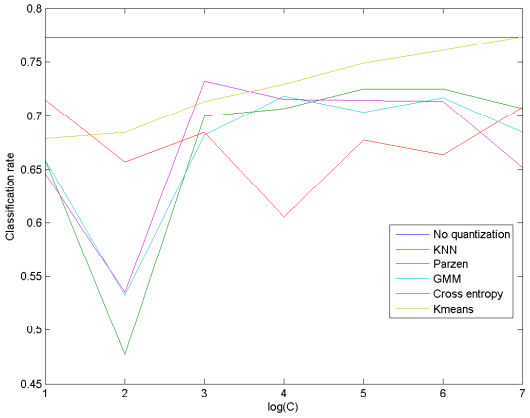

Fig. 16. Linear classifier results for 10-fold cross-validation scheme and fixed β strategy. $C = 10, \tau_1 = 0.1, \tau_2 = 0.005, \beta = \beta_0$

experiments. For this case, the classifier accuracy falls to 50%, which is equivalent to simply guessing one of the classes (for a uniform prior for the class density). One explanation for this behavior may be that for the case of $C = 4$, each class is separated into two clusters and the clusters happen to fall on opposite sides of the decision boundary, leading to this poor behavior. Finally, it seems that unsupervised clustering actually performs better than unsupervised clustering for the linear and NN classifiers. This may be a result of the fact that Kmeans is the optimal clustering strategy if we seek to preserve the structure of the original input space, χ, and the classifiers used in my experiments perform better when data is clustered in an l_2 optimal sense. For instance, we know that the linear classifier can actually penalize correct predictions due to the form of its error function, so clustering points that are far apart, as minimization of information loss sometimes does, can lead to unsatisfactory results.

VI. HAR Results

I compared the performance of the human activity recognition algorithm described above using supervised and unsu-

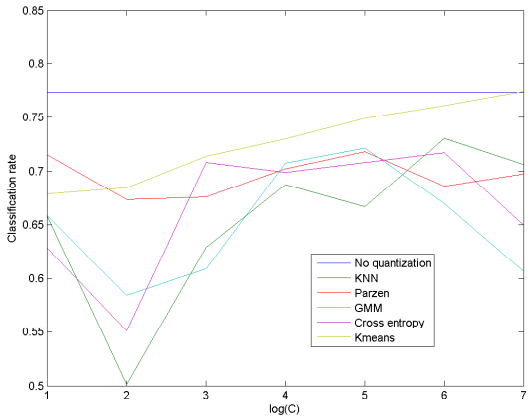

Fig. 17. Linear classifier results for 10-fold cross-validation scheme and annealing β strategy. $C = 10, \tau_1 = 0.1, \tau_2 = 0.005, \beta = \beta_n$

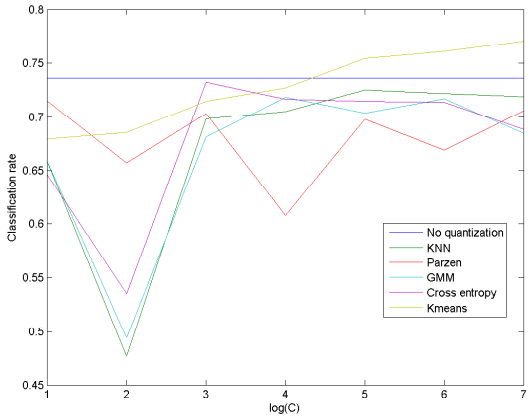

Fig. 18. NN classifier results for 10-fold cross-validation scheme and fixed β strategy. $C = 10, \tau_1 = 0.1, \tau_2 = 0.005, \beta = \beta_0$

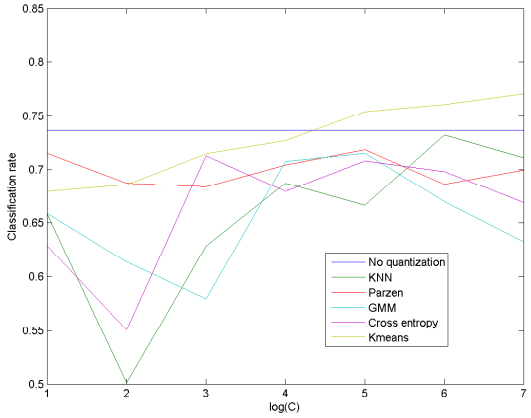

Fig. 19. NN classifier results for 10-fold cross-validation scheme and annealing β strategy. $C = 10, \tau_1 = 0.1, \tau_2 = 0.005, \beta = \beta_n$

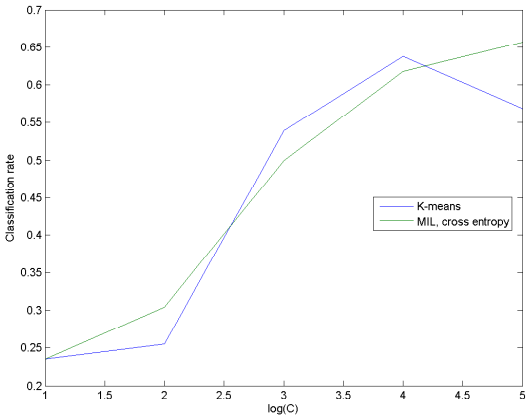

Fig. 20. Activity recognition classification rate using supervised and unsupervised clustering ($h = 10$)

pervised clustering to learn the pose codebook. The results are presented in Fig. 20. The dataset consisted of 13 actions. We can see that the difference between the two clustering algorithms is marginal for low cluster numbers, but supervised clustering begins to perform better for the higher cluster amounts. This result may stem from the fact that the performance of Kmeans is highly dependent on the initialization of the algorithm and it is only guarantee to converge to a local minimum, even though that local minimum may be a poor one. Minimization of information loss, on the other hand, is a convex problem with a global minimum. Therefore, I suspect that the superior performance of minimization of information loss over Kmeans for higher cluster numbers is due to the fact that the clustering achieved by minimization of information loss improves with higher cluster numbers, from an information theoretic point of view, while the performance of Kmeans decreases with increasing cluster numbers because the number of sub-optimal local minima increases as C grows.

I should note that I was only able to get reasonable clustering performance using the neural network posterior learning technique (with $h = 10$). This may be due to the versatility of the neural network approach to learn arbitrary distributions.

VII. Conclusion

In this report, I experimented with the minimization of information loss clustering technique, using several posterior pdf estimation techniques and several validation methodologies. The results show that when the number of classes is low, Kmeans outperforms minimization of information loss. On the other hand, if the number of classes is high and the input space is high dimensional, we can see some improvement in the classification performance of data clustered using minimization of information loss compared to Kmeans.

References

[1] S. Lazebnik and M. Raginsky, "Supervised Learning of Quantizer Codebooks by Information Loss Minimization," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 31, No. 7, July 2009.

[2] I.Dhillon, S. Mallela, and R. Kumar, "A Divisive Information Theoretic Feature Clustering Algorithm for Text Classification,"*J. Machine Learning Research*, vol. 3, pp. 1265-1287, 2003.

[3] B. Ni, G. Wang, P. Moulin, "RGBD-HuDaAct: A Color-Depth Video Database For Human Daily Activity Recognition." *Consumer Depth Cameras for Computer Vision.* Springer London, 2013. 193-208.

[4] N. Slonim and N. Tishby,"Document clustering using word clusters via the information bottleneck method," *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2000.

[5] N. Slonim, "The information bottleneck: Theory and Applications." Dissertation. Hebrew University of Jerusalem, 2002.

[6] S.D. Chen and P. Moulin, "A Two-Part Predictive Coder for Multitask Signal Compression," In preparation.

[7] T.M. Cover and J.A. Thomas, "Elements of Information Theory," 2006.

[8] C.M. Bishop, "Neural Networks for Pattern Recognition."

[9] *Lecture 7: Minimization or maximization of functions (Recipes Chapter10)*. Available: http://www.pha.jhu.edu/~neufeld/numerical/lecturenotes7.pdf

[10] R. Gutierez-Osuna. *Lecture 10: Density estimation II*. Available: http://courses.cs.tamu.edu/rgutier/cs790_w02/l10.pdf

[11] K. Rose, "Deterministic Annealing for Clustering, Compressions, Classification, Regression, and Related Optimization Problems," *Proceedings of the IEEE*, Vol. 86, No. 11, Nov. 1998.

[12] V. Escorica, M. A. Davila, M. Golparvar-Fard, J.C. Niebles, "Automated Vision-based Recognition of Construction Worker Actions for Building Interior Construction Operations Using RGBD Cameras," *CRC* 2012.