Multi-Attribute Queries: To Merge or Not to Merge?

Mohammad Rastegari  Ali Diba  Devi Parikh  Ali Farhadi

Goal: Finding a best combination of keywords (attributes) in an image search query.

Visual Phrase: [Gong et al. CVPR11]

• Intuition: In a multi-attribute image search, some combinations of attributes can be learned jointly, resulting in a better classifier.

Why not:
– Learning individual detector for each attribute? It may not be effective due to significant difference in appearance (Joint attribute may have similar appearance across images)
– Learning one detector by merging all attributes? It may not be powerful due to the lack of jointly labeled training data.

✓ Efficient  × Inaccurate

Naive Solution (Upper Bound):
Learn and evaluate all possible combinations on a validation set, pick the best.

× Inefficient  ✓ Accurate

Our Approach:
Estimate “learnability” of each combination efficiently without training, pick the best.

✓ Efficient  ✓ Accurate

Learnability Function:
Set of attributes: \( A = \{a_1, a_2, \ldots, a_k\} \)

A component: \( c_i \in S = P(A) \quad S = \{c_1, c_2, \ldots, c_m\} \), \( m = 2^k \)

A combination: \( C \subseteq S \)
Margin: \( \mathcal{K}(c_1, c_2) \) Average pair wise distance between two sets of instances labeled by \( c_1 \) and \( c_2 \)

Diagonal: \( D(c) \) Average distance in a set of instances labeled by \( c \)

\[
\mathcal{L}(C) = \sum_{c \in C} \left( \sum_{c' \in C, c' \neq c} \mathcal{K}(c, c') \right) + \sum_{a \in C} \mathcal{K}(c, c \setminus a) - D(c)
\]

Complexity for computing the Margin \( \mathcal{K} \) between two sets with \( n_1 \) and \( n_2 \) elements is \( O(n_1 n_2) \)

Complexity for computing the Diagonal \( D \) of a set with \( n \) elements is \( O(n^2) \)

In Binary feature space:
Margin \( \rightarrow O(n^2) \)
Diagonal \( \rightarrow O(n) \)

Recent binary code methods are very accurate: DBC [Rastegari et al. ECCV12] and ITQ [Gong et al CVPR11]

Optimization:

\[
\max_x \mathcal{L}(S \odot x) - \lambda |x|
\]

\( Z^T x \geq 1 \quad x \in \{0,1\}^m \)

NP-Hard!!

Gain Function:
\( G(a_i, a_j) = \mathcal{K}(a_i, a_j) + \mathcal{K}(a_i, a_j) - D(a_i, a_j) \)

• The higher \( G(a, a) \) the higher is the reward for merging \( a \) and \( b \)

Greedy Algorithm:
• For every pairs of attributes compute \( G \)
• Pick the pair with maximum \( G \)
• If the maximum \( G > 0 \) then:
  1. Merge the two corresponding attributes
  2. Add the new merged-attribute
  3. Remove the two independent attribute

Reducing the search space drastically

Lemma 1. If attributes \( a_i \) and \( a_j \) are merged because \( G(a_i, a_j) > 0 \) then for any other attribute \( a_k \), \( G(a_k, a_k) \geq G(a_i, a_k) + G(a_i, a_j) \)

Proof. It’s simple to show that if \( A \subseteq B \) then \( P(A) \subseteq P(B) \) and if \( C \subseteq D \) then \( K(A, C) \leq K(B, D) \).

We can show that \( G(a_i, a_j) = K(a_i, a_k) + K(a_i, a_j) \) and \( G(a_i, a_j) \geq G(a_i, a_k) + G(a_i, a_j) \)

\( G(a_i, a_j) = K(a_i, a_k) + K(a_i, a_j) \) and \( G(a_i, a_j) \geq G(a_i, a_k) + G(a_i, a_j) \)

Lemmas hold for \( \mathcal{K}(a_i, a_k) \) and \( D(a_i, a_k) \). The same holds for \( \mathcal{K}(a_i, a_j) \) and \( D(a_i, a_j) \).

\( O(k^3) \) vs. \( O(2^k) \)

Our method is robust across the binary code methods

Experiments:
Datasets:
1. aPascal [Farhadi et al. 2009]
2. Caltech-UCSD Bird200 [Welinder et al. 2010]

Efficient Sum of Pairwise Hamming Distances

Algorithm 1 Efficient Sum of Pairwise Hamming Distances

- Input: \( S \) sets of Hamming distances between all pairs of rows in \( P \)
- Output: \( S \) sets of Hamming distances between all pairs of rows in \( P \)
- for \( A = 1 \ldots |S| \) do
- \( G(i) = \sum_{j=1}^{|S|} \mathcal{K}(i,j) \) // Counting Number of zeros in 1st dimension of \( G \)
- \( G(A) = \sum_{i=1}^{|S|} \mathcal{K}(i,j) \) // Counting Number of ones in 2nd dimension of \( G \)
- end for
- for \( B = 1 \ldots |S| \) do
- if \( |G(A) - G(B)| > 0 \) then
- \( \mathcal{K}(S,A,B) = |G(A) - G(B)| \)
- end if
- end for
- end for

Comment: Save all elements in \( P \)