Classifier-as-a-Service: Online Query of Cascades and Operating Points

Brandyn A. White\textsuperscript{1}, Andrew E. Miller\textsuperscript{2}, Larry S. Davis\textsuperscript{1}

University of Maryland: College Park\textsuperscript{1}, University of Central Florida\textsuperscript{2}

\{bwhite, lsd\}@cs.umd.edu

1 Introduction

We introduce a classifier and parameter selection algorithm for Classifier-as-a-Service applications where there are many components (e.g., features, kernels, classifiers) available to construct classification algorithms. Queries specify varying requirements (i.e., quality and execution time), some of which may require combining classification algorithms to satisfy; each query may have a different set of quality and execution time requirements (e.g., fast and precise, slow and thorough) and the set of images to which the classifier is to be applied may be small (e.g., even a single image), necessitating a query resolution method that takes negligible time in comparison. When operating on large datasets, meeting design requirements automatically becomes essential to reducing costs associated with unnecessary computation and expert assistance. As queries specify requirements and not implementation details, additional components can be utilized naturally. Our query resolution method combines classifiers with complementary operating points (e.g., high recall algorithmic filter, followed by high precision human verification) in a rejection-chain configuration. Experiments are conducted on the SUN397\textsuperscript{[1]} dataset; we achieve state-of-the-art classification results and 1 m.s. query resolution times.

2 Proposed Approach

We describe a method for efficiently determining a combination of classifiers and thresholds sufficient to achieve a desired per-class quality and overall execution time specification, provided as a “query”. Query resolution involves combing classifiers into a rejection-chain cascade which satisfy quality and execution time requirements as close as possible. Our primary contribution is the development of an efficient method to determine the performance characteristics of these cascades offline, so that they can be queried efficiently online. Given pre-trained cascade stages $S$ (Sec. 2.1), we predict each class $l \in \mathcal{L}$ using each cascade stage $S \in S$ producing a confidence matrix $\vec{x}^S$ with $|\mathcal{L}|$ rows and $N$ columns, where $N$ is the size of a validation set. We illustrate our approach on the scene recognition task, where each image belongs to exactly one class and the validation groundtruth $\vec{g}$ is a matrix of the same shape as $\vec{x}^S$ with values...
If a cascade stage \( S \) the threshold selection (Sec. 2.2) produces a set of thresholds \( T^S_l \) that capture that cascade stage’s performance. The cascade selection produces \( C_l \subseteq \mathcal{P}(S) \) where \( \mathcal{P}(S) \) is the powerset of the set of cascade stages \( S \). Finally the cascade simulation (Sec. 2.3) simulates each cascade \( C \in C_l \) using the thresholds selected \( T^S_l \) where \( S \in C \). The simulation is an efficient method of evaluating the performance of the cascades and produces operating points (i.e., confusion matrices and times) that are then stored in the cascade database. This training phase operates on each class \( l \) independently and, for notational convenience, we let \( \vec{g}_l, \vec{x}^S = \vec{x}^S_l, T^S = T^S_l \), and \( C = C_l \).

### 2.1 Cascade Stages

A cascade stage \( S \) takes an image \( I \) and a set of classes \( \mathcal{L} \) from which it produces a (possibly sparse) confidence vector \( x^S \). For the single class case in Fig. 1(a), this is a single image feature \( f \) and a single binary classifier \( c_1 \) which produces the confidence vector \( x^S \) as \( x^S_1 \leftarrow c_1(f(I)) \). For the general form in Fig. 1(b), a single feature \( f \) is shared by many classifiers and \( x^S \) is produced as \( \forall l \in \mathcal{L}, x^S_l \leftarrow c_l(f(I)) \). We now provide a definition for a cascade stage \( S \) where \( I \) is an image and \( \mathcal{L} \) is the set of classes (represented as positive integers): (1) The cascade stage is defined for all \( \mathcal{L}, x^S \leftarrow S(I, \mathcal{L}) \) where \( S \) may be non-deterministic and \( x^S \) is a real-valued vector, (3) \( \forall l, m \in \mathcal{L}, \) confidence value \( x^S_l \) is independent of \( m \in \mathcal{L} \) or \( m \notin \mathcal{L} \) where \( l \neq m \), and (4) Larger values of \( x^S_l \) signify higher confidence that the input belongs to class \( l \).

### 2.2 Threshold Selection

Given a trained cascade stage \( S \) and a validation set, our task is to find a set of thresholds \( T^S \) that compactly represents its operating points. The threshold selection occurs independently for each class and cascade stage \( S \). We wish to minimize the number of thresholds \( |T^S| \) to reduce the cascade simulation complexity. We say a confidence value \( \vec{x}^S_l \) is positive with respect to \( t \in T^S \) if \( x^S_l \geq t \). The initial set of thresholds is \( T^S_u = \mathcal{X}^S \cup \{\infty\} \), where \( \mathcal{X}^S \) is the set of confidence values of \( \vec{x}^S \). Including infinity ensures that at least one threshold has no false positives. This set is sufficient as any other threshold produces a redundant partition; however, it is not necessary as thresholds which produce “worse” confusion matrices may be present (i.e., more FP or FN errors). We construct a subset of \( T^S_u \) that is both necessary and sufficient. Given a confidence value \( \vec{x}^S_l \) and ground truth label \( \vec{g}_l \) for each image \( i \) in the validation set, we sort them ascending by confidence value with positive ground truth instances listed before negative ones for the same confidence values. Observe that the secondary sorting eliminates ‘overestimated’ confusion matrices that can result from naïve generation of confusion matrices with the same confidence value but different ground truth polarities[2]. As this
process is independent of the stage $S$, we let $\bar{g} = \bar{g}^S$, $\bar{x} = \bar{x}^S$, and $X = X^S$. When operating on the vector $\bar{g}$, $\bar{g}_i$ represents the ground truth polarity at position $i$, with $\bar{x}_i$ as its associated confidence value and $\bar{g}_{i-1}$ as its neighbor in the descending direction.

We find the minimum number of thresholds $T_c$ required to exactly represent the performance characteristics of the cascade stage. The predicate $Keep(\bar{x}_i)$ is true when $\bar{g}_{i-1}$ is not positive and $\bar{g}_i$ is not negative; this clearly produces the desired set as $T_c = \{\bar{x}_i \in X : Keep(\bar{x}_i)\}$. More practically, we can relax our exact representation by allowing for a bounded absolute precision/recall difference, $z$, between the confusion matrices produced by $T_c$ and a subset $T_{a,z}$ of them, leading to $O(1)$ thresholds. This is accomplished by assigning each threshold in $T_c$ as a node in a graph with edges induced by $z$ and computing the dominating set, which correspond to thresholds in $T_{a,z}$.

2.3 Cascade Selection/Simulation

We evaluate two methods for selecting which candidate cascades of length $\ell$ should be used for simulation: dense ($D$) and sparse ($S$). The sparse method selects the $\beta$ stages with highest rank correlation to the ground truth and limits the branching factor to $\alpha$ stages with lowest minimum rank correlation with preceding stages. Given a set of cascades $C$ for a class with an associated set of thresholds $T^S$ where $S \subseteq C$ and $C \in C$, the goal is to efficiently compute a cascade database that contains the union of all cascades in $C$ that can be formed over all of their thresholds. This process is a simulation as we are not computing the cascade performance directly; rather, we find the confidence values for all stages independently and exactly compute their combined performance offline. The output for each set of thresholds for a cascade includes a binary confusion matrix, stage names, stage thresholds, and % of inputs evaluated at each stage (used to compute time).

A naïve approach would evaluate the validation set using every possible chain and its corresponding set of thresholds. There are a combinatorial number of operations in the length of the cascade if every possible ordering of a single cascade (e.g., $A \rightarrow B \rightarrow C$, $C \rightarrow B \rightarrow A$) were considered; however, the order does not affect the resulting binary confusion matrix. We order the stages by running time per image. When evaluating a cascade $A \rightarrow B \rightarrow C$, we have already computed $A$ and $A \rightarrow B$ which can be reused for longer cascades with common prefixes (e.g., $A \rightarrow B$ reused in $A \rightarrow B \rightarrow C$ and $A \rightarrow B \rightarrow D$). Stages with the same parent are all computed simultaneously. The computational complexity of this method is the sum of the complexity for each cascade from a leaf to a root node. The complexity of each path is bounded by $O(N \prod_{S \in C} |T^S|)$ where $N$ is the number of validation inputs (e.g., images for image classification). However, this worst case performance cannot occur since the number of validation inputs $N$ reduces from stage to stage, exactly as they do when passing through a rejection chain cascade.

3 Experiments

We show experimental results on the SUN397\cite{sun397} scene recognition dataset which consists of 39,700 images (50 train/test per class, using partition #1). Fig. 2(a) summarizes the features used along with their recognition rate for multi-class classification using linear and histogram intersection kernels (HIK) over all of the features. LAbhist uses
Figure 2: (left) Recognition rates of features/kernels. (right) Existing (#0-3), upper bound (#4-5), and (#6-15) method variations (threshold method, max cascade length $\ell$, cascade selection method). The threshold selection methods (Sec. 2.2) are $T_e$ and $T_a,z$. The cascade selection methods (Sec. 2.3) are $D$ and $S_\alpha$ with $\beta$ as half of the stages.

As we are evaluating binary classification performance, we use the mean area under the ROC. Our method produces a significant improvement (compared to Fig. 2(#0-3)) using similar features, classifiers, and kernels Fig. 2(#7). It is clear that this gain is from the cascade design as the features alone Fig. 2(#6) perform worse. We use 20% of the training set as a validation set to learn the cascade database with the classifiers trained on the other 80%. We are able to calculate an upper bound for our method by identifying the optimal cascades and thresholds on the test set Fig. 2(#4-5); the approach has near ideal performance. To gain further insight we show the number of classes with AUC better, same, or worse ($>$, $=$, $<$) than [1] for each result in Fig. 2 (our method uses 20% less training than [1] due to calibration).

To evaluate query performance, for all classes and for constraints of length 1 to 5 (i.e., precision, recall, time, accuracy, F-1) we compute 100 random example queries (e.g., House with $p > .4$, $r > .3$, and $t < 1$) with a cap of 100 returned cascades using the cascade database produced by Fig. 2(#14). The median query times (in ascending order from 1-5 constraints) are: 1.1, 3.5, 1.0, .1, and .09 in m.s.; The additional constraints reduce the search space dramatically. This is fast compared to the feature/classifier computation and is only performed once per query resolution, not per image as in[9]. Multi-class queries (i.e., quality per class, overall time) produce per-class candidates and the overall time is minimized. Computations were performed on a 2.2GHz Xeon.
References


