
How to save feature extraction for fast and robust classification?

Jérôme Louradour

A2iA SA Paris

Christopher Kermorvant

The application that motivates this work is the classification of scanned documents, mainly for mailroom automation. Typically, document target classes are numerous and diverse: ID cards and other official papers, letters of complaint, subscription requests, etc. Some classes of documents (*e.g.* official papers) can be easily recognized using raw features, such as gray levels in image sub-resolution. On the other hand, others (*e.g.* handwritten letters with different topics) require a finer analysis, such as automatic text transcription followed by bag-of-words encoding. The task is an hard challenge (many classes, noisy features) and classification error rates of state-of-the-art systems are still too high to consider full automation. Hence classification confidence estimation must also be provided for each test sample, to be able to reach a good trade-off between accuracy and automation rate.

Combining all available features into a single classification model is not cumbersome given that some learners can find out optimal ways to select and combine features for multi-class discrimination. However, the brute-force approach is not satisfactory in terms of test computation time. An efficient classification system should be able to avoid extracting high-level features when it is not necessary *-i.e.* when reliable classification could be achieved by processing only low-level features if a test sample is "simple" enough.

We aim to perform sample-dependent feature extraction for efficient classification, for a given embedded *confidence-rated* classification model and some features extraction processes. The goal is to achieve classification accuracy similar to *-if not better than-* the baseline approach (classification after full extraction of features) with significantly less computational effort (mostly due to the extraction of high-level features).

Assuming that the embedded classifier can be more and more accurate as higher-level features are appended¹, we propose to build a cascade decision tree adapted to our goal. At each *decision node*, a new set of features are extracted and a node-specific classi-

fier *-trained with all types of features available at the current node-* suggests a best candidate class along with a confidence estimate. The decision can be either (1) to continue with higher-level features or (2) to trust the current classification and terminate. This choice is made by comparing the confidence to a threshold. An automated grid search approach is used to find the best thresholds for all nodes, with respect to a loss that we define as a linear combination of the probabilities of having chosen the correct class at other nodes. A critical point is that the whole system must provide a confidence estimate that is homogeneous for all possible *terminal nodes*. Provided that the embedded classifiers already compute "reliable" confidence estimates (trained on a big dataset), we propose to apply a node-specific correction transform to the *terminal node* classifier's confidence score. We experimented several forms for these correction transforms, and the best results were obtained by training a linear regression between each node classifier's scores and the scores of the last classifier in the cascade (the most robust one, trained on all possible features). It leads to significant improvement in comparison with no correction.

Experiments were carried out on two real databases for mail classification with respectively 16 and 127 classes, and baseline error rates around 9.5% and 28%. We compare the classification error rates at different automation levels, between a baseline AdaBoost approach and the proposed approach with a decision tree on AdaBoost classifiers outputs. For our approach, we varied two parameters values in the loss function used to choose the decision thresholds. These parameters monitor the tendency to refrain from extracting high-level features, and by this way the gain in test CPU time. Empirical results show that our approach permits to achieve good CPU time improvement (up to 40% off) while keeping the same classification accuracy as the baseline in a full automation scenario. However, the quality of the confidence score is more and more degraded as we force the system to rely on low-level features. A trade-off between gain of time and loss of accuracy at a given automation rate must be chosen by empirical validation. We plan to investigate other approaches to improve the quality of confidence scores.

¹This assumption is not warranted for all classification methods because of the *curse of dimensionality*, but it is true in our experiments with AdaBoost of decision stumps.