Dynamic Feature Selection for Classification on a Budget

Sergey Karayev
UC Berkeley

Mario Fritz
MPI Informatics

Trevor Darrell
UC Berkeley

Abstract

The economics of real-world implementations of predictive systems require power consumption (cpu-time) to be budgeted. A predictive system extracts features, which vary in cost, and combines them to output a regression or classification score. With a computational budget at test time, it is often not possible to extract all features for every instance. If different instances benefit from different features, selection of the best subset should be *dynamic*—a one-size-fits-all *static* policy is suboptimal. We address the challenge of minimizing *Anytime* loss given an arbitrary test-time budget and fully-observed training data. Our algorithm jointly learns a **feature selection** policy that stays within the cost budget and **feature combination** that minimizes loss. We take exact cost budget as input; the algorithm does not require tuning parameters to find an acceptable operating point. Evaluating on an illustrative synthetic problem, a visual scene recognition task (Scene-15), and a hierarchically-structured classification task (ImageNet), we show that our dynamic, non-myopic algorithm works better than several baselines. On the hierarhical dataset, we additionally show that maximally *specific* answers can be given for any desired cost budget and level of accuracy.

1 Introduction

Multi-class recognition has achieved levels of performance that allow useful real-world implementation. However, state-of-the-art methods tend to be computationally expensive. As these methods are applied at scale, managing their resource consumption (power or cpu-time) cost becomes crucial.

For most state-of-the-art classification, different features are extracted from an instance at different costs, and contribute differently to lowering the overall error of the classifier. More features mean lower error, but high accuracy can be achieved with only a small subset of features for some instances—and different instances benefit from different subsets of features. For example, simple binary features are sufficient to quickly detect faces [18] but not more varied visual objects, while the features most useful for separating landscapes from indoor scenes [19] are different from those most useful for recognizing fine distinctions between bird species [9].

Due to the cost, computing all features for all images is not feasible in a real world deployment, as each feature brings a significant computational burden. If an additional feature takes only half a second to compute, a system that processes a hundred million images a day will require an additional 1.5 years of cpu time per day.

A way to deal with this problem is to set an explicit cost *budget*, specified in terms of wall or cpu time or total power expended or other desiderata. Additionally, we may desire *Anytime* performance—the ability to terminate the classifier even before the cost budget is depleted, and still obtain the best answer. In this paper, we address the problem of selecting and combining a subset of features under an *Anytime* cost budget.

To exploit the fact that different instances benefit from different subsets of features, our approach to feature selection is a sequential policy. To learn the policy parameters, we formulate the problem as a Markov Decision Process (MDP) and use reinforcement learning methods. With different settings of parameters, we can find the best policy from **Static, Myopic**—greedy selection not relying on any observed feature values, to **Dynamic, Non-myopic**—selection that relies on observed feature values and considers future actions.

Since test-time efficiency is our motivation, our methods must not increase the computational burden. For this reason, our models are based on linear evaluations, not nearest-neighbor or graphical model methods. Because different features can be selected for different instances, and because our system may be called upon to give an answer at any point during its execution, the feature combination method needs to be robust to a large number of different observed-feature subsets. To this end, we present a novel method for learning several classifiers for different clusters of observed-feature subsets.

We evaluate our method on multi-class recognition tasks. We first demonstrate on synthetic data that our algorithm learns to pick features that are most useful to compute for the specific test instance. We demonstrate the advantage of non-myopic over greedy, and of dynamic over static on this and the Scene-15 visual classification dataset. Then we show results on a subset of the hierarchical ImageNet dataset, where we additionally learn to provide the most specific answers for any desired cost budget and accuracy level.

2 Related Work

Static selection A well-known method to evaluate features sequentially is the cascaded boosted classifier of Viola & Jones [18] (updated by Bourdev & Brandt [3] to a soft threshold model), which is able to quit evaluating an instance before all features are computed—but feature cost was not considered. The cost-sensitive cascade of Chen et al. [4] cyclically optimizes stage order and thresholds to jointly minimize classification error and feature computation cost.

Xu et al. [21] and Grubb & Bagnell [12] separately develop a variant of gradient boosting for training cost-sensitive classifiers; the latter prove near-optimality of their greedy algorithm with submodularity results. Their methods are tightly coupled to the stage-wise regression algorithm.

Dynamic selection The above methods learn an efficient but *fixed* order for evaluating features given a test instance, which is not able to use different features for different instances.

Gao & Koller [11] propose a method for *active classification*: myopically selecting the next feature based on expected information gain given the values of the already selected features. The method is based on locally weighted regression, highly costly at test time. Ji & Carin [14] also formulate cost-sensitive feature selection generatively, as an HMM conditioned on actions, but select actions myopically, again at cost at test time. Karayev at al. [15] propose a reinforcement learning approach for selecting object detectors; they rely on expensive test-time inference in a graphical model to combine observations.

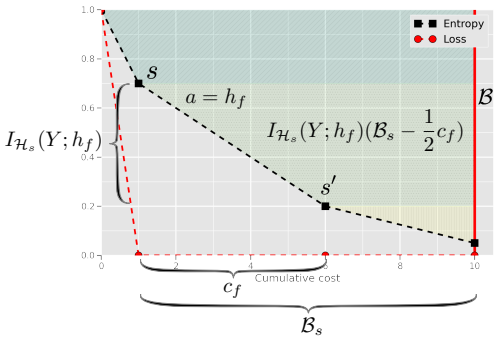
Dulac-Arnold et al. [8] present another MDP-based solution to "datum-wise classification", with an action space comprised of all features and labels. He He et al. [13] also formulate an MDP with features and a single classification step as actions, but solve it via imitation learning of a greedy policy. Benbouzid et al. [1] formulate an MDP that simply extends the traditional sequential boosted classifier with an additional *skip* action.

Label trees also guide an instance through a tree of classifiers; their structure is determined by the confusion matrix [2] or learned jointly with weights [7]. Xu et al. [20] learn a cost-sensitive binary tree of weak learners using an approach similar to the cyclic optimization of [4].

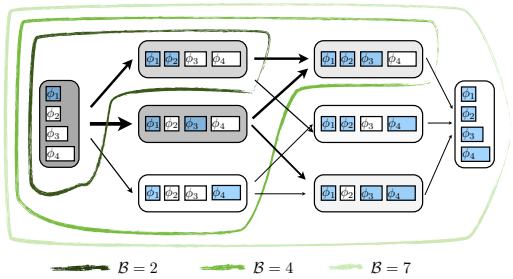
All but [15] achieve the tradeoff between accuracy and cost by tuning a unit-less parameter, and none but [15] target *Anytime* performance. However, its inference procedure is prohibitively expensive for test-time use in a general classification task. In contrast, our fast linear method allows direct specification of the Anytime cost budget.

3 Cost-sensitive Dynamic Feature Selection

We first formally define the problem and our notation.



(a) Definition of the reward function.



(b) Illustration of the state space.

Figure 1: (a) We seek to maximize the total area above the entropy vs. cost curve from 0 to $\mathcal{B}$, and so define the reward of an individual action as the area of the slice of the total area that it contributes. From state s, action h leads to state s' with cost c_f. The information gain of the action $a = h_f$ is $I_{\mathcal{H}_s}(Y; h_f) = H(Y; \mathcal{H}_s) - H(Y; \mathcal{H}_s \cup h_f)$.
(b) The action space $\mathcal{A}$ of the MDP is the the set of features $\mathcal{H}$, represented by the ϕ boxes. The primary discretization of the state space can be visualized by the possible feature subsets (larger boxes); selected features are colored in the diagram. The feature selection policy π induces a distribution over feature subsets, for a dataset, which is represented by the shading of the larger boxes. Not all states are reachable for a given budget $\mathcal{B}$. In the figure, we show three "budget cuts" of the state space.

Definition 1. ***The test-time efficient multi-class classification problem*** *consists of*

- *N instances labeled with one of K labels: $\mathcal{D} = \{x_n \in \mathcal{X}, y_n \in \mathcal{Y} = \{1, \ldots, K\}\}_{n=1}^{N}$.*
- *F features $\mathcal{H} = \{h_f : \mathcal{X} \mapsto \mathbb{R}^{d_f}\}_{f=1}^{F}$, with associated costs c_f.*
- *Budget-sensitive loss $\mathcal{L}_\mathcal{B}$, composed of cost budget $\mathcal{B}$ and loss function $\ell(\hat{y}, y) \mapsto \mathbb{R}$.*

*The goal is to find a **feature selection** policy $\pi(x) : \mathcal{X} \mapsto 2^\mathcal{H}$ and a **feature combination** classifier $g(\mathcal{H}_\pi) : 2^\mathcal{H} \mapsto \mathcal{Y}$ such that such that the total budget-sensitive loss $\sum \mathcal{L}_\mathcal{B}(g(\pi(x_n)), y_n)$ is minimized.*

The cost of a selected feature subset $\mathcal{H}_{\pi(x)}$ is $C_{\mathcal{H}_\pi(x)}$. The budget-sensitive loss $\mathcal{L}_\mathcal{B}$ presents a **hard budget constraint** by only accepting answers with $C_\mathcal{H} \leq \mathcal{B}$. Additionally, $\mathcal{L}_\mathcal{B}$ can be **cost-sensitive**: answers given with less cost are more valuable than costlier answers. The motivation for the latter property is *Anytime* performance; we should be able to stop our algorithm's execution at any time and have the best possible answer.

Feature costs c_f can be specified flexibly, with options including theoretical analysis, number of flops, wall clock runtime, total CPU time, or exact power expenditure. We believe that a deployment in a modern datacenter is most likely to optimize for power expenditure. In the absence of reliable ways to measure power, we use total CPU time to define the cost: if an operation is performed in parallel on multiple cores, its cost is considered to be the total cpu time on all cores.

At training time, our computation is unbudgeted, and we can compute all features to have *fully-observed* training instances. At test time, there is a budget and so the instances we classify will only be *partially-observed*, as determined by the feature selection policy.

We defer discussion of learning the **feature combination** classifier $g(\mathcal{H}_\pi) : 2^\mathcal{H} \mapsto \mathcal{Y}$ to Section 3.4. For now, we assume that g can combine an arbitrary subset of features and provide a distribution $P(Y = y)$. For example, g could be a Naive Bayes (NB) model trained on the fully-observed data.

3.1 Feature selection as an MDP.

To model the **feature selection** policy $\pi(x) : \mathcal{X} \mapsto 2^\mathcal{H}$, we introduce the Markov Decision Process (MDP), which defines a single *episode* of selecting features for some instance x.

Definition 2. *The **feature selection MDP** consists of the tuple* $(\mathcal{S}, \mathcal{A}, T(\cdot), R(\cdot), \gamma)$*:*

- ***State*** $s \in \mathcal{S}$ *stores the selected feature subset* $\mathcal{H}_{\pi(x)}$ *and their values and total cost* $C_{\mathcal{H}_{\pi(x)}}$*.*
- *The set of **actions*** $\mathcal{A}$ *is exactly the set of features* $\mathcal{H}$*.*
- *The (stochastic) **state transition** distribution* $T(s' \mid s, a)$ *can depend on the instance* x*.*
- *The **reward** function* $R(s, a, s') \mapsto \mathbb{R}$ *is manually specified, and depends on the classifier* g *and the instance* x*.*
- *The discount* γ *determines amount of **lookahead** in selecting actions: if 0, actions are selected greedily based on their immediate reward; if 1, the reward accrued by subsequent actions is given just as much weight as the reward of the current action.*

Running the MDP on a given instance x gives a trajectory $\xi = (s_0, a_0, s_1, r_1, \ldots, a_{I-1}, s_I, r_I)$, where I is the total number of actions taken (and therefore features selected), s_0 is the initial state, $a_i \sim \pi(a \mid s_i)$ is chosen by the *policy* $\pi(a \mid s)$, and $s_{i+1} \sim T(s \mid s_i, a_i)$, which can depend on x. The total expected reward of an MDP episode—its *value*—is written as

$$V_\pi(s_0) = \mathbb{E}_{\xi \sim \{\pi, x\}} r(\xi) = \mathbb{E}_{\xi \sim \{\pi, x\}} \left[\sum_{i=0}^{I} \gamma^i r_i \right] \tag{1}$$

Gathering such trajectories forms the basis of our policy learning method.

3.2 Defining the reward.

Recall that the budget-sensitive loss $\mathcal{L}_\mathcal{B}$ enforces *Anytime* performance by valuing early gains more than later gains. To formalize this, consider Figure 1, which shows the entropy and the 0-1 loss of g at every point in a sequential feature selection episode for some instance x. For the best *Anytime* performance, we want to capture the most area above the loss vs. cost curve, up to max budget $\mathcal{B}$ [15].

Recall from (1) that the value of an episode ξ is defined as the sum of obtained rewards. If the reward of a single action is defined as the area above the curve that is captured as a direct result, then the value of the whole episode exactly corresponds to $\mathcal{L}_\mathcal{B}$.

However, there is a problem with using loss directly: only the first action to "tip the scale" toward the correct prediction gets a direct reward (in the figure, it is the first action). If the classifier g can give a full distribution $P(Y = y \mid \mathcal{H}_{\pi(x)})$ and not just a prediction $\hat{y} \in \mathcal{Y}$, we can maximize the *information gain* of the selected subset instead of directly minimizing the loss of $g(\pi(x))$:

$$I(Y; \mathcal{H}_{\pi(x)}) = H(Y) - H(Y|\mathcal{H}_{\pi(x)}) = \sum_{y \in Y} P(y) \log P(y) - \sum_{y, \mathcal{H}_{\pi(x)}} P(y, \mathcal{H}_{\pi(x)}) \log P(y \mid \mathcal{H}_{\pi(x)})$$

To the extent that g is unbiased, maximizing information gain corresponds to minimizing loss, and ensures that we not only make the right classification decision but also become maximally certain. Therefore, as graphically presented in Figure 1, we define the reward of selecting feature h_s with cost c_f with the set $\mathcal{H}_s$ computed to be $I_{\mathcal{H}_s}(Y; h_f)(\mathcal{B}_s - \frac{1}{2} c_f)$.

Although we do not evaluate in this regime, note that this definition easily incorporates a **setup cost** in addition to **deadline cost** by only counting the area in between setup and deadline costs.

3.3 Parametrizing and learning the policy.

Space constraints prohibit a full exposition of reinforcement learning techniques; consult [17] for a thorough review. In brief: we seek π that maximizes the expected value of the MDP (1). Therefore, actions must be selected according to their expected *value*:

$$\arg\max_a \pi(a \mid s) = \arg\max_a Q^*(s, a)$$

where $Q^*(s, a)$ is the optimal *action-value function*—the expected value of taking action a in state s and then acting optimally to the end of the episode.

Because the state represents an exponential number of subsets and associated real values, we cannot represent $Q(s, a)$ exactly. Instead, we use feature approximation and write $Q(s, a) = \theta^T \phi(s, a)$,

where $\phi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^{d_s}$ is the state featurization function, d_s is the dimensionality of the state feature vector, and θ is a vector of weights that defines the policy.

Specifically, the policy is defined as

$$\pi(a \mid s) = \frac{1}{Z} \exp\left(\frac{1}{\tau}\theta^T \phi(s,a)\right) \tag{2}$$

where Z is the appropriate normalization and τ is a temperature parameter that controls the level of exploration vs. exploitation in the policy. As $\tau \to 0$, $\pi(a \mid s)$ becomes highly peaked at $\arg\max_a Q(s,a)$; it becomes uniform as $\tau \to \infty$.

As commonly done, we learn the θ by *policy iteration*. First, we gather (s, a, r, s') samples by running episodes (to completion) with the current policy parameters θ_i. From these samples, $\hat{Q}(s,a)$ values are computed, and θ_{i+1} are given by L_2-regularized least squares solution to $\hat{Q}(s,a) = \theta^T \phi(s,a)$, on all states that we have seen in training.

During training, we gather samples starting from either a random feasible state, with probability ϵ, or from the initial empty state otherwise. Both ϵ and τ parameters decay exponentially with the number of training iterations. Training is terminated if $\pi_{\theta_{i+1}}$ returns the exact same sequence of episodes ξ on a validation set as π_{θ_i}.

Static vs. Dynamic state-action feature vector. The featurization function $\phi(s)$ extracts the following features from the state:

- Bit vector **m** of length F: initially all bits are 1 and are set to 0 when the corresponding feature is computed.
- For each h_f, a vector of size d_f representing the values; 0 until observed.
- Cost feature $c \in [0, 1]$, for fraction of the budget spent.
- Bias feature 1.

These features define the **dynamic** state, presenting enough information to have a *closed-loop* (dynamic) policy that may select different features for different test instances. The **static** state has all of the above features except for the observed feature values. This enables only an *open-loop* (static) policy, which is exactly the same for all instances. Policy learned with the static state is used as a baseline in experiments.

The state-action feature function $\phi(s,a)$ effectively block-codes these features: it is 0 everywhere except the block corresponding to the action considered. In implementation, we train F separate regressions with a tied regularization parameter, which is K-fold cross-validated.

Effect of γ. Note that solving the MDP with these features and with $\gamma = 0$ finds a **Static, greedy** policy: the value of taking an action in a state is exactly the expected reward to be obtained. When $\gamma = 1$, the value of taking an action is the entire area above the curve as defined in Figure 1, and we learn the **Static, non-myopic** policy—another baseline.

3.4 Learning the classifier.

Input: $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$; $\mathcal{L}_\mathcal{B}$
Result: Trained π, g

$\pi_0 \leftarrow$ random;
for $i \leftarrow 1$ **to** *max_iterations* **do**
  States, Actions, Costs, Labels $\leftarrow$ `GatherSamples(`$\mathcal{D}$, π_{i-1}`)`;
  $g_i \leftarrow$ `UpdateClassifier(`*States, Labels*`)`;
  Rewards $\leftarrow$ `ComputeRewards(`*States, Costs, Labels,* $g_i, \mathcal{L}_\mathcal{B}, \gamma$`)`;
  $\pi_i \leftarrow$ `UpdatePolicy(`*States, Actions, Rewards*`)`;
end

Algorithm 1: Because reward computation depends on the classifier, and the distribution of states depends on the policy, g and π are trained iteratively.

We have so far assumed that g can combine an arbitrary subset of features and provide a distribution $P(Y = y)$—for example, a Gaussian Naive Bayes (NB) model trained on the fully-observed data. However, a Naive Bayes classifier suffers from its restrictive independence assumptions.

Discriminative classifiers are better able to learn the most from available data. For this reason, we use a **logistic regression** classifier, which presents new challenges. At test time, some feature values are missing and need to be imputed. If the classifier is trained exclusively on fully-observed data, then the feature value statistics at test time will not match, resulting in poor performance. Therefore, we need to learn classifier weights on a distribution of data that exhibits the pattern of missing features induces by the policy π. At the same time, learning the policy depends on the classifier g, used in the computation of the rewards. For this reason, the policy and classifier need to be learned jointly: algorithm 1 shows the iterative procedure.

Unobserved value imputation. Unlike the Naive Bayes classifier, the logistic regression classifier is not able to use an arbitrary subset of features $\mathcal{H}_\pi$, but instead operates on feature vectors of a fixed size. To represent the feature vector of a fully observed instance, we write $\mathbf{x} = [h_1(x), \ldots, h_f(x)]$. In case that $\mathcal{H}_\pi \subset \mathcal{H}$, we need to fill in unobserved feature values in the vector.

A basic strategy is **mean imputation**: filling in with the mean value of the feature:

$$\mathbf{x}_\pi = \left[h_i(x) : \begin{cases} h_i(x) & \text{if } h_i \in \mathcal{H}_{\pi(x)} \\ \bar{\mathbf{h}}_i & \text{otherwise} \end{cases} \right] \tag{3}$$

If we assume that $\mathbf{x}$ is distributed according to a multivariate Gaussian $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$, where Σ is the sample covariance $X^T X$ and the data is standardized to have zero mean, then it is possible to do **Gaussian imputation**. Given a feature subset $\mathcal{H}_\pi$, we write:

$$\mathbf{x}_\pi = \begin{bmatrix} \mathbf{x}^{\text{o}} \\ \mathbf{x}^{\text{u}} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^T & \mathbf{B} \end{bmatrix} \right) \tag{4}$$

where $\mathbf{x}^{\text{o}}$ and $\mathbf{x}^{\text{u}}$ represent the respectively observed and unobserved parts of the full feature vector $\mathbf{x}$. In this case, the distribution over unobserved variables conditioned on the observed variables is given as $\mathbf{x}^{\text{u}} \mid \mathbf{x}^{\text{o}} \sim \mathcal{N}\left(\mathbf{C}^T \mathbf{A}^{-1} \mathbf{x}^{\text{o}}, \mathbf{B} - \mathbf{C}^T \mathbf{A}^{-1} \mathbf{C}\right)$.

Learning more than one classifier. As illustrated in Figure 1b, the policy π selects some feature subsets more frequently than others. Instead of learning only one classifier g that must be robust to all observed feature subsets, we can learn several classifiers, one for each of the most frequent subsets. This is done by maintaining a distribution over encountered feature subsets during training. For each of the K most frequent subsets, a separate classifier is trained, using data that is closest by Hamming distance on the selected-feature bit vector.

Each classifier is trained with the LIBLINEAR implementation of logistic regression, with L_2 regularization parameter K-fold cross-validated at each iteration.

4 Evaluation

We evaluate the following sequential selection baselines:

- **Static, greedy**: corresponds to best performance of a policy that does not observe feature values and selects actions greedily ($\gamma = 0$).
- **Static, non-myopic**: policy that does not observe feature values but uses the MDP machinery with $\gamma = 1$ to consider future action rewards.
- **Dynamic, greedy**: policy that observed feature values, but selects actions greedily.

Our method is the **Dynamic, non-myopic** policy: observed feature values, and full lookahead.

In preliminary experiments, Logistic Regression always performed better than the Gaussian Naive Bayes classifier, and so only the former is used in evaluation experiments. As described, we evaluated classification with **Gaussian** vs. **Mean** imputation, and with different number of classifiers (1, 3, and 6) clustered by feature subsets. We found that mean imputation performed better than Gaussian imputation, and although increased number of classifiers sometimes increased performance, it also made our method more prone to overfitting, so $K = 1$ classifiers worked best. For reason of space, we report only the best achieved performance in the following evaluation results.

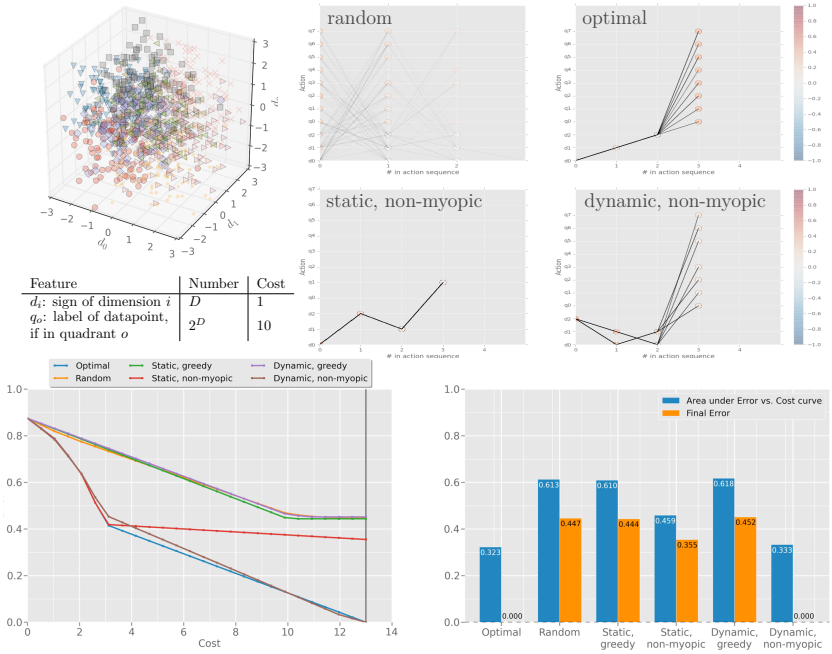


Figure 2: Evaluation on the 3-dimensional synthetic example (best viewed in color). The data is shown at top left; the sample feature trajectories of four different policies at top right. The plots in the bottom half report results for baselines and our Dynamic, Non-Myopic approach. Note that the static, non-myopic policy correctly learns to select the cheap features first, but does not have the necessary information to correctly branch, while the dynamic, non-myopic policy matches the optimal performance with branching.

4.1 Synthetic Experiment.

Following [20], we first show that the policy works as advertised in a challenging synthetic example. In D-dimensional space, the data has a label for each of the 2^D orthants, and is generated by a unit-variance Gaussian in that orthant (See top left of Figure 2 for the 3D case). There are D cheap features that simply return the sign of the data point's coordinate for the corresponding dimension. For each orthant, there is also an expensive feature that returns the data point's label if the point is located in the corresponding orthant, and random noise otherwise.

Clearly, the optimal policy is to determine the orthant with cheap features, and then take the corresponding expensive action. Figure 2 shows the results of this optimal policy, a random policy, and of different baselines and our method. Since there is clearly a correct minimal budget, we only evaluate at that budget. Note that both dynamic features and non-myopic learning are crucial to the optimal policy, which is successfully found by our approach.

4.2 Scenes.

The Scene-15 dataset [16] contains 4485 images from 15 visual scene classes. The task is to identify classify images according to scene. Following [19], we extracted 14 different visual features (GIST, HOG, TinyImages, LBP, SIFT, Line Histograms, Self-Similarity, Textons, Color Histograms, and variations). The features vary in cost from 0.3 seconds to 8 seconds, and in single-feature accuracy from 0.32 (TinyImages) to .82 (HOG). Separate multi-class linear SVMs were trained on each feature channel, using a random 100 positive example images per class for training. We used the liblinear implementation, and K-fold cross-validated the penalty parameter C.

The trained SVMs were evaluated on the images not used for training, resulting in a dataset of 2238 vectors of 210 confidence values: 15 classes for each of the 14 feature channels. This dataset was split 60-40 into training and test sets for our experiments.

Figure 3 shows the areas under the error vs. cost curve—Anytime performance—of the methods. For all evaluated budgets, our method significantly outperforms all others. Our results on this dataset match the reported results of Active Classification [11] (Figure 2) and exceed the reported results of Greedy Miser [21] (Figure 3), although both methods use an additional powerful feature channel (ObjectBank).

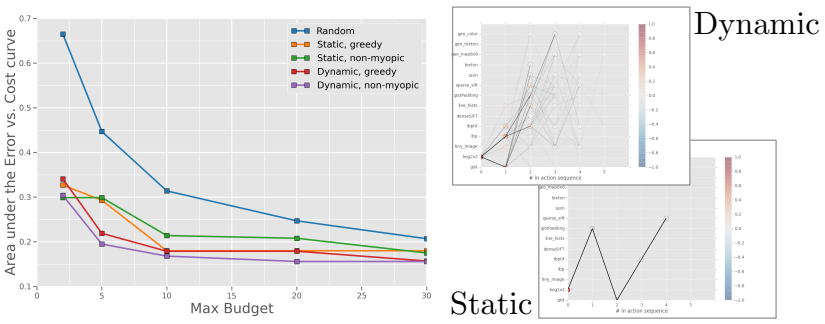


Figure 3: Results on Scenes-15 dataset.

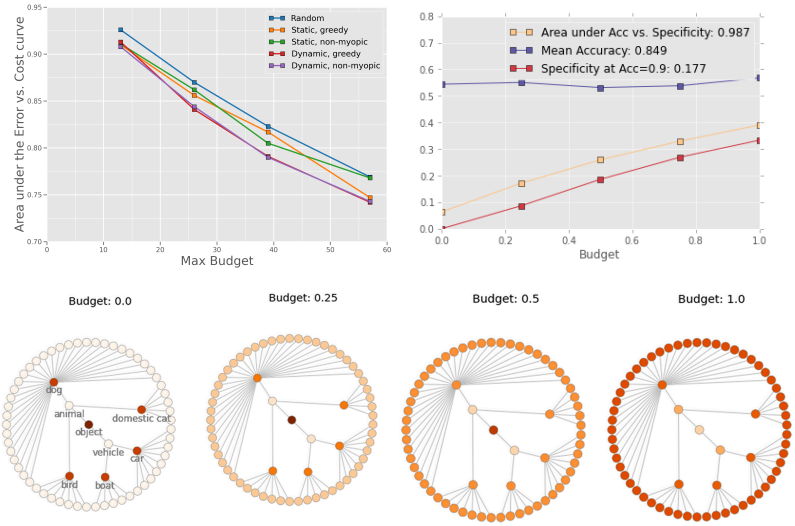


Figure 4: Results on the Imagenet 65-class subset. Note that when our method is combined with Hedging Your Bets [6], a constant accuracy can be achieved, with *specificity* of predictions increasing with the budget, as in human visual perception. (Color saturation corresponds to percentage of predictions at node.)

4.3 ImageNet and maximizing specificity.

The full ImageNet dataset has over 10K categories and over a million images [5]. The classes are organized in a hierarchical structure, which can be exploited for novel recognition methods. We evaluate on a 65-class subset used in [6]. In this evaluation, we consider the situation where the initial feature computation has already happened, and the task is to find a path through existing one-vs-all classifiers. Figure 4 shows a graph representation of the 65-class hierarchy. Features correspond to Platt-scaled SVM confidences of leaf-node classifiers, and each has cost 1. Accuracy is defined on all nodes; inner node confidences are given by summing the probabilities of the descendant nodes.

We combine our sequential feature selection with the Hedging Your Bets [6] method for finding the level of the hierarchy to maintain a guaranteed accuracy to give maximally specific answers given a cost budget. This is seen in Figure 4: as the available budget increases, the *specificity* (defined by normalized information gain in the hierarchy) of our predictions also increases, while accuracy remains constant. This suggests that our policy may be used as a model of human visual perception, which is known to be iteratively refined [10].

References

[1] D. Benbouzid, R. Busa-Fekete, and B. Kegl. Fast classification using sparse decision DAGs. In *ICML*, 2012. 2

[2] S. Bengio, J. Weston, and D. Grangier. Label Embedding Trees for Large Multi-Class Tasks. In *NIPS*, number 1, 2010. 2

[3] L. Bourdev and J. Brandt. Robust Object Detection via Soft Cascade. In *CVPR*, 2005. 2

[4] M. Chen, Z. Xu, K. Q. Weinberger, O. Chapelle, and D. Kedem. Classifier Cascade for Minimizing Feature Evaluation Cost. In *AISTATS*, 2012. 2

[5] J. Deng, A. C. Berg, K. Li, and L. Fei-fei. What Does Classifying More Than 10,000 Image Categories Tell Us ? In *ECCV*, pages 71–84, 2010. 8

[6] J. Deng, J. Krause, A. C. Berg, and L. Fei-fei. Hedging Your Bets: Optimizing Accuracy-Specificity Trade-offs in Large Scale Visual Recognition. In *CVPR*, 2012. 8

[7] J. Deng, S. Satheesh, A. C. Berg, and L. Fei-fei. Fast and Balanced: Efficient Label Tree Learning for Large Scale Object Recognition. In *NIPS*, number 1, pages 1–9, 2011. 2

[8] G. Dulac-Arnold, L. Denoyer, P. Preux, and P. Gallinari. Sequential approaches for learning datum-wise sparse representations. *Machine Learning*, 89(1-2):87–122, Aug. 2012. 2

[9] R. Farrell, O. Oza, V. I. Morariu, T. Darrell, and L. S. Davis. Birdlets: Subordinate categorization using volumetric primitives and pose-normalized appearance. In *ICCV*, Nov. 2011. 1

[10] L. Fei-Fei, A. Iyer, C. Koch, and P. Perona. What do we perceive in a glance of a real-world scene? *Journal of vision*, Jan. 2007. 8

[11] T. Gao and D. Koller. Active Classification based on Value of Classifier. In *NIPS*, 2011. 2, 8

[12] A. Grubb and J. A. Bagnell. SpeedBoost: Anytime Prediction with Uniform Near-Optimality. In *AISTATS*, 2012. 2

[13] H. He, D. Hal III, and J. Eisner. Cost-sensitive Dynamic Feature Selection. In *ICML-W*, 2012. 2

[14] S. Ji and L. Carin. Cost-Sensitive Feature Acquisition and Classification. *Pattern Recognition*, 2007. 2

[15] S. Karayev, T. Baumgartner, M. Fritz, and T. Darrell. Timely Object Recognition. In *NIPS*, 2012. 2, 4

[16] S. Lazebnik, C. Schmid, and J. Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *CVPR*, volume 2, pages 2169–2178. Ieee, 2006. 7

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. 4

[18] P. Viola, O. M. Way, and M. J. Jones. Robust Real-Time Face Detection. *IJCV*, 57(2):137–154, 2004. 1, 2

[19] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 1, 7

[20] Z. Xu, M. J. Kusner, K. Q. Weinberger, and M. Chen. Cost-Sensitive Tree of Classifiers. In *ICML*, 2013. 2, 7

[21] Z. Xu, K. Q. Weinberger, and O. Chapelle. The Greedy Miser: Learning under Test-time Budgets. In *ICML*, 2012. 2, 8