The Extreme Value Machine – Supplemental Material

June 30, 2017

1 Algorithm Details

In this section we provide the details of the algorithms presented in the main text of the paper. The process of training the EVM is provided in Alg. 1.

Algorithm 1 EVM Training

```
1: function TRAIN EVM($X$,$y$,$\tau$,$\varsigma$)    ▷ $X$ and $y$ are data and labels for all classes $\mathcal{C}$.
2:     for $\mathcal{C}_l \in \mathcal{C}$ do
3:         $\Psi_l \leftarrow$Fit $\Psi(X,y,\tau,C_l)$                 ▷ See Alg. 2.
4:         $I \leftarrow$ Reduce$(X_l, \Psi_l, \varsigma)$           ▷ See Algs. 3 and 4.
5:         $\Psi_l \leftarrow \Psi_l[I]$
6:         $X_l \leftarrow X_l[I]$
7:         $y_l \leftarrow y_l[I]$
8:     end for
9: end function
```

Figure 1: Algorithm for EVM training. The function calls are detailed in Algs. 2, 3, and 4.

For brevity, we adopt array notation. The function takes four arguments (cf. line 1): X and y correspond to all training data and labels respectively, τ is the tailsize, and ς is the coverage threshold. The training function then iterates through every class $\mathcal{C}_l \in \mathcal{C}$ (line 2), fitting a vector of Ψ-models (Ψ_l) for that class (line 3). The fitting function is presented in Alg. 2 and takes four arguments: X, y, τ, and the class identifier C_l. At each fit, only the Ψ-models for the points and labels corresponding to class C_l, *i.e.*, X_l and y_l are considered. The function returns Ψ_l – a vector of Weibull parameters. Next, the model reduction routine is called (line 4), which takes as arguments X_l, Ψ_l, and coverage threshold ς. This routine can correspond to Algs. 3 or 4. A vector, I, of indices is returned by this routine, and is then used to select Weibull parameters Ψ_l, points X_l, and labels y_l to keep (lines 5-7). Note that for conceptual clarity, we have presented Alg. 1 in an iterative fashion. However, it is possible to parallelize the loop in lines 2-8, fitting each class separately. The extent to which that is effective is an implementation-level consideration that depends on the hardware at hand, particularly since each Ψ_{li} in the fitting algorithm (Alg. 2) can also be parallelized as can pair-wise distance computations within all subroutines (Algs. 2-4).

Algorithm 2 Ψ-Model Fitting

```
1: function FIT $\Psi(X,y,\tau,C_l)$
2:     $D$ = cdist$(X_l,X_{\neg l})$     ▷ $N_l \times N_{\neg l}$ dimensional matrix.
3:     for $i = 1$ to $N_l$ do
4:         $\Psi_{li} \leftarrow$ Weibull_fit_low$(\frac{1}{2} \times$ sort$(D_i)[:\tau])$
5:     end for
6:     return $\Psi_l$
7: end function
```

Figure 2: Algorithm for fitting Ψ-Models. This is called from Alg. 1.

The fitting routine, depicted in Alg. 2, takes as arguments (cf. line 1) all sample points X with labels y, tailsize τ, and class identifier C_l. We use notation X_l to refer to points belonging to class C_l and $X_{\neg l}$ to refer to all other points. In line 2, the $N_l \times N_{\neg l}$ matrix of pairwise distances is computed, then each row of that matrix is sorted in ascending order and an MLE Weibull fit is performed on the nearest τ margins (line 4). Weibulls are then returned as Ψ_l (line 6).

The first model reduction routine, which approximates the solution to Set Cover is shown in Alg. 3. Employing this algorithm yields a non-pre-determined number of extreme vectors. The routine takes three arguments (line 1): Ψ_l, X_l, and ς – the coverage threshold. First, in line 2, pairwise distances between points are computed to yield an $N_l \times N_l$ matrix of distances, then for each of these points, respective probabilities of association with

Algorithm 3 Set Cover Model Reduction

```
1:  function SET COVER($\Psi_l$,$X_l$,$\varsigma$)
2:      $\widetilde{D}$ =pdist($X_l$)    ▷ An $N_l \times N_l$ matrix.    ▷ Subsets of indices.
3:      for $i = 1$ to $N_l$ do
4:          for $j = 1$ to $N_l$ do
5:              if ($\Psi_{li}(\widetilde{D}_{ij}) \geq \varsigma$) then
6:                  $\mathbb{S}_i$.insert($j$)
7:              end if
8:          end for
9:      end for
10:     $\mathbb{U} = \{1, ..., N_l\}$    ▷ Universe.
11:     $\mathbf{C} \leftarrow \{\}$    ▷ Covered elements.
12:     $I \leftarrow []$    ▷ Cover indices.
13:     while ($\mathbf{C} \neq \mathbb{U}$) do
14:         $ind \leftarrow \text{argmax}_i\ (\mathbb{S}_i - \mathbf{C})$
15:         $\mathbf{C} \leftarrow \mathbf{C} \cup \mathbb{S}_{ind}$
16:         $I$.append($ind$)
17:         $\mathbb{S}$.remove($\mathbb{S}_{ind}$)
18:     end while
19:     return $I$
20: end function
```

Figure 3: A fast algorithm for model reduction which leverages an approximation to Set Cover. Using this yields a variable number of extreme vectors.

Algorithm 4 Model Reduction

```
1:  function FIXED-SIZE REDUCTION($\Psi_l$,$X_l$,$N_{max}$,$\delta_P$)
2:      $\varsigma_{min} \leftarrow 0.0$
3:      $\varsigma_{old} \leftarrow \varsigma_{max} \leftarrow 1.0$
4:      $I_{old} \leftarrow [1, .., N_l]$
5:      while true do    ▷ Infinite while loop.
6:          $\varsigma \leftarrow (\varsigma_{min} + \varsigma_{max})/2$
7:          $I \leftarrow$ Set_Cover($\Psi_l$,$X_l$,$\varsigma$)    ▷ Call to Alg. 3.
8:          $M \leftarrow$ length($I$)
9:          $C_1 \leftarrow (N - N_{max} \geq M - N_{max} > 0)$
10:         $C_2 \leftarrow (N - N_{max} \geq M - N_{max} < 0)$
11:         $C_3 \leftarrow (N - N_{max} < M - N_{max})$
12:         $C_4 \leftarrow ((M = N_{max})$ or $abs(\varsigma - \varsigma_{old}) \leq \delta_P)$
13:         if $C_1$ then
14:             $\varsigma_{max} \leftarrow \varsigma$
15:         else
16:             if $C_2$ or $C_3$ then
17:                 $\varsigma_{min} \leftarrow \varsigma$
18:             end if
19:         end if
20:         if $C_4$ then
21:             return $I[: N_{max}]$    ▷ First $N_{max}$ elements of $I$
22:         end if
23:         $I \leftarrow I_{old}$; $\varsigma_{old} \leftarrow \varsigma$
24:     end while
25: end function
```

Figure 4: Algorithm for fixed-size model reduction. The appropriate value of ς is arrived at to yield a model of a fixed size via a binary search on the output of the Set Cover algorithm. While a fixed number of extreme vectors can be returned, this algorithm takes longer to compute, as it must call Alg. 3 several times.

the point governing the corresponding row are computed (line 5). For each point-probability in the ith row, if it is above ς, it is added to the respective set of covered points associated with the ith extreme vector, *i.e.*, $\mathbb{S}_i$. In lines 13-18, values i of the greatest cardinality $\mathbb{S}_i$ are greedily selected without replacement until all points within class $\mathcal{C}_l$ have been covered, and the respective indices of these points are appended to vector I. Once all points in the class are covered I corresponds to the indices of points, Ψ-models, and labels to retain as extreme values. This vector is returned to the calling routine in line 19.

An extension of the Set Cover model reduction algorithm in Alg. 3 is the fixed-size reduction algorithm in Alg. 4, in which a bisection is performed on ς to attain a fixed number of extreme vectors (N_{max}). The first two arguments to this algorithm (Ψ_l and X_l) are the same as for Alg. 3. N_{max} corresponds to the maximum number of extreme vectors and δ_P corresponds to a tolerance on the difference between chosen ς values over iterations at which to terminate the bisection.

The bisection is carried out in lines 5-24. Upon the completion of the search, the value of ς that most closely, within δP yields N_{max} extreme vectors, with at least N_{max} extreme vectors, is selected and out of these extreme vectors, the N_{max} of highest coverage are returned.

Since Alg. 4 allows a principled mechanism for the EVM to use a fixed subset of Extreme Vectors, it is technically possible, by extension, to have Vector Ratio (discussed in the main text) as a hyperparameter of the EVM rather than ς. However, this does come at an increased computational cost, since multiple calls to Alg. 3 have to be performed. Examining the ramifications of using Vector Ratio as a hyperparameter is a topic which we leave for future work.

2 Parameter Distributions

While the EVM *significantly* outperforms NNO in terms of both accuracy and F1-measure (see Sec. V. of the main text), and is easily parallelized as the Weibull fits do not depend on one another, computing class means is still noticeably more efficient than computing maximum likelihood estimates of λ and κ over all extreme vectors. For very large datasets or for training with limited resources, it is natural to ask: can we make the EVM more efficient by choosing principled parameter values and avoiding fits altogether, or using better initializations on parameter values to speed up MLE convergence?

While formulating such a classifier is largely beyond the scope of the main text, as a preliminary exploration of this question, we illustrate the distribution of shape and scale parameters in Fig. 5 for both of the datasets that we evaluate in the main text. For both ImageNet and Letter, scale parameters were very close to one. This is interesting because the datasets are quite different, both in terms of characteristics and in terms of dimensionality. Moreover, different metrics were chosen over which to compute the margins. However, the distributions behave quite similarly, with a slightly longer tail for the Letter dataset. The approximately equal mean scale of Ψ-models between datasets is not surprising, due to the normalization of input data during pre-processing. What is more striking is the similar behavior of the distributions.

Shape parameter distributions assume a much wider range for both ImageNet and Letter and the distributions look little alike between the two datasets. Thus, it is not immediately clear what would constitute a principled choice for κ. In order to ascertain the dependence of the model on κ, we selected values of $\kappa = 1$ and $\kappa = 2$, and attempted to run the open set protocol on OLETTER. However, this resulted in very poor open set recognition performance.

3 EVM-SVM Analogy

The Extreme Value Machine has some components that are analagous to those of Support Vector Machines, but the respective models and components behave in fundamentally different manners. While Ψ-models are kernel-like radial basis functions, they do not obey Mercer's Theorem and are therefore not Mercer kernels for values of $\kappa > 2$. As can be seen in the figures from the previous section, $\kappa > 2$ occurs quite frequently, so the EVM cannot be interpreted as a kernel machine.

Further, the EVM optimization objective of selecting the smallest number of extreme vectors to cover all points within the class within some probability differs from the SVM objective of selecting support vectors that yield maximum soft margin. SVM support vectors carry little information about class inclusion/structure over all space but considerable information about structure of the decision boundary, while EVMs carry information with respect to probability of inclusion over each entire class.

While one can draw an analogy between ς and the C parameter in an SVM, it is a very loose analogy – these hyperparameters serve a fundamentally different purpose: C controls the softness of the margin for an SVM, whereas ς controls the number of extreme vectors. The loose analogy is that C and ς control what support vectors or extreme vectors constitute the model, but they do so in much different ways. The primary objective of C is to tune the softness of the margin, whereas the primary objective of ς is to tune the size of the model. Certain choices of ς may have regularizing effects too, but assessing them is a topic that we leave to future work. A better analogy to the SVM's C parameter might be τ, as it does control the softness of the margin, but again, what constitutes margin is fundamentally different – the EVM's soft margin is a result of the EVM's point-wise margin distribution. Note also that modelsize is dependent on both τ and ς, because larger values of τ result in more coverage and thus the same value of ς will yield fewer extreme vectors.

With respect to the W-SVM (referenced in the main text of the paper), it is also a fundamentally different model from the EVM: It consists a one-class SVM model and a multi-class SVM model, but the EVT fitting is done in a post-hoc manner, at the score level, and after the SVMs have been trained. The EVM, by contrast, performs EVT fitting directly during the optimization process, computed in feature space – not at the score level.

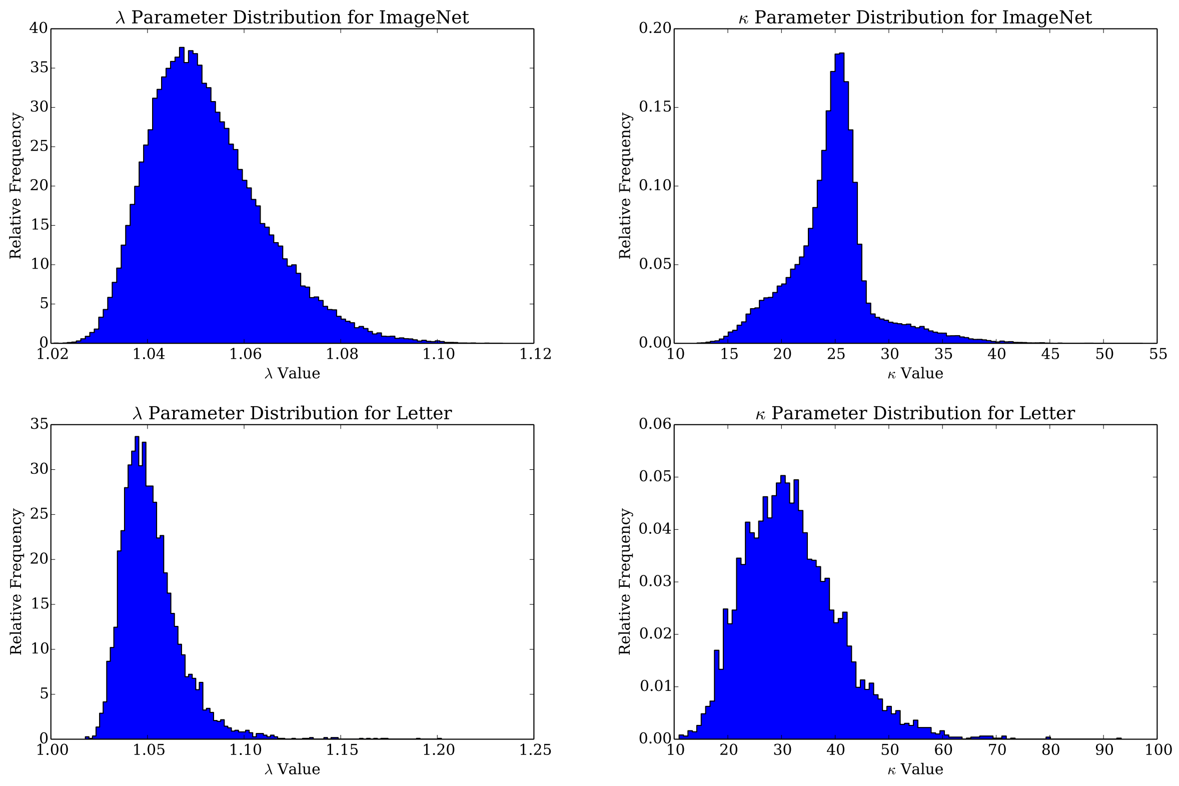

Figure 5: Distributions of Ψ-model scale (λ) and shape (κ) parameters over the first 200 classes of the ImageNet training set and over all classes of the Letter training set.