CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

Feb 5, 2021

Outline

Outline

Logistics

- HW 1 is due today, and HW 2 will be assigned.
- Please form the groups for the project, we'll have groups of 3 students working together. Use piazza to find group members.

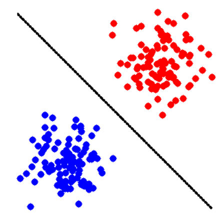
Outline

Summary

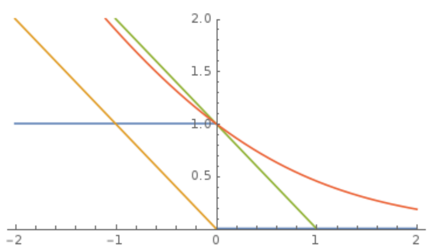
Linear models for **binary** classification:

Step 1. Model is the set of **separating hyperplanes**

$$\mathcal{F} = \{f(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}\}$$



Step 2. Pick the **surrogate loss**



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1-z\}$(used in SVM and many others)
- logistic loss $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z))$ (used in logistic regression)

Step 3. Find empirical risk minimizer (ERM):

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^{\mathsf{D}}} F(\boldsymbol{w}) = \operatorname*{argmin}_{\boldsymbol{w}\in\mathbb{R}^{\mathsf{D}}} \frac{1}{N}\sum_{n=1}^{N} \ell(y_n \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$$

using

- **GD:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\nabla F(\boldsymbol{w})$
- **SGD:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\tilde{\nabla} F(\boldsymbol{w})$
- **Newton:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \left(\nabla^2 F(\boldsymbol{w})\right)^{-1}\nabla F(\boldsymbol{w})$

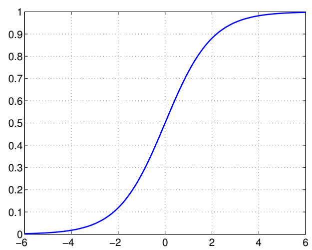
A Probabilistic view of logistic regression

Minimizing logistic loss = MLE for the sigmoid model

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}} \sum_{n=1}^{N} \ell_{\text{logistic}}(y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) = \operatorname*{argmax}_{\boldsymbol{w}} \prod_{n=1}^{N} \mathbb{P}(y_n \mid \boldsymbol{x}_n; \boldsymbol{w})$$

where

$$\mathbb{P}(y \mid \boldsymbol{x}; \boldsymbol{w}) = \sigma(y \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}) = \frac{1}{1 + e^{-y \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}}}$$



Outline

Classification

Recall the setup:

- input (feature vector): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- goal: learn a mapping $f : \mathbb{R}^{\mathsf{D}} \rightarrow [\mathsf{C}]$

Examples:

- recognizing digits ($\mathsf{C} = 10$) or letters ($\mathsf{C} = 26$ or 52)
- predicting weather: sunny, cloudy, rainy, etc
- predicting image category: ImageNet dataset ($\mathsf{C} \approx 20K$)

Nearest Neighbor Classifier naturally works for arbitrary C.

Linear models: from binary to multiclass

Step 1: *What should a linear model look like for multiclass tasks?*

Note: a linear model for binary tasks (switching from $\{-1, +1\}$ to $\{1, 2\}$)

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} \geq 0 \\ 2 & \text{if } \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} < 0 \end{cases}$$

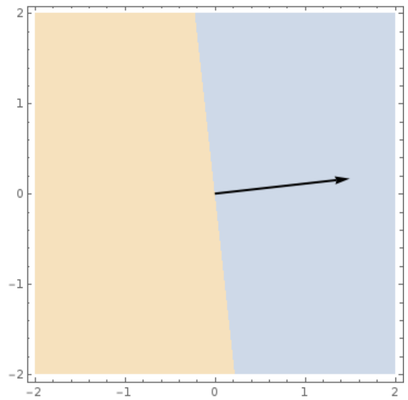
can be written as

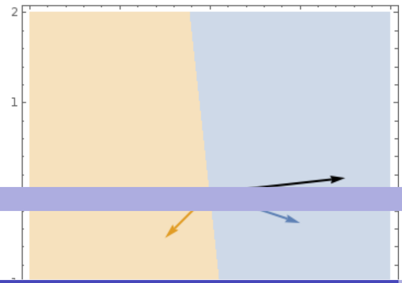
$$\begin{aligned} f(\boldsymbol{x}) &= \begin{cases} 1 & \text{if } \boldsymbol{w}_1^{\mathrm{T}} \boldsymbol{x} \geq \boldsymbol{w}_2^{\mathrm{T}} \boldsymbol{x} \\ 2 & \text{if } \boldsymbol{w}_2^{\mathrm{T}} \boldsymbol{x} > \boldsymbol{w}_1^{\mathrm{T}} \boldsymbol{x} \end{cases} \\ &= \operatorname*{argmax}_{k \in \{1,2\}} \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} \end{aligned}$$

for any $\boldsymbol{w}_1, \boldsymbol{w}_2$ s.t. $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$

Think of $\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}$ as **a score for class** k.

Linear models: from binary to multiclass

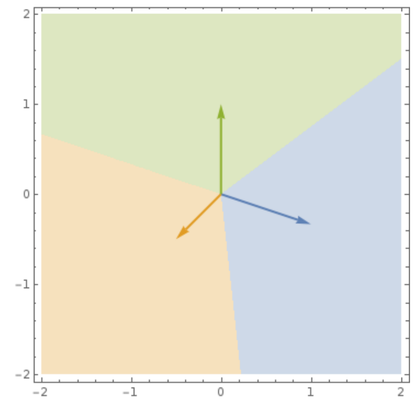




$\boldsymbol{w} = (\frac{3}{2}, \frac{1}{6}) = \boldsymbol{w}_1 - \boldsymbol{w}_2$
$\boldsymbol{w}_1 = (1, -\frac{1}{3})$
$\boldsymbol{w}_2 = (-\frac{1}{2}, -\frac{1}{2})$

- Blue class:
  $\{\boldsymbol{x} : 1 = \text{argmax}_k \, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$

Linear models: from binary to multiclass



$\boldsymbol{w}_1 = (1, -\frac{1}{3})$
$\boldsymbol{w}_2 = (-\frac{1}{2}, -\frac{1}{2})$
$\boldsymbol{w}_3 = (0, 1)$

- Blue class:
  $\{\boldsymbol{x} : 1 = \text{argmax}_k \, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Orange class:
  $\{\boldsymbol{x} : 2 = \text{argmax}_k \, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Green class:
  $\{\boldsymbol{x} : 3 = \text{argmax}_k \, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$

Linear models for multiclass classification

$$\mathcal{F} = \left\{ f(\boldsymbol{x}) = \operatorname*{argmax}_{k \in [\mathsf{C}]} \; \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} \mid \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{\mathsf{C}} \in \mathbb{R}^{\mathsf{D}} \right\}$$
$$= \left\{ f(\boldsymbol{x}) = \operatorname*{argmax}_{k \in [\mathsf{C}]} \; (\boldsymbol{W}\boldsymbol{x})_k \mid \boldsymbol{W} \in \mathbb{R}^{\mathsf{C} \times \mathsf{D}} \right\}$$

Step 2: *How do we generalize perceptron/hinge/logistic loss?*

This lecture: focus on the more popular **logistic loss**

Multinomial logistic regression: a probabilistic view

Observe: for binary logistic regression, with $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$:

$$\mathbb{P}(y = 1 \mid \boldsymbol{x}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}} = \frac{e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}}}{e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}} + e^{\boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x}}} \propto e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}}$$

Naturally, for multiclass:

$$\mathbb{P}(y = k \mid \boldsymbol{x}; \boldsymbol{W}) = \frac{e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k' \in [\mathsf{C}]} e^{\boldsymbol{w}_{k'}^{\mathrm{T}}\boldsymbol{x}}} \propto e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}$$

This is called the *softmax function*.

Applying MLE again

Maximize probability of seeing labels $y_1, \ldots, y_{\mathsf{N}}$ given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{\mathsf{N}}$

$$P(\boldsymbol{W}) = \prod_{n=1}^{\mathsf{N}} \mathbb{P}(y_n \mid \boldsymbol{x}_n; \boldsymbol{W}) = \prod_{n=1}^{\mathsf{N}} \frac{e^{\boldsymbol{w}_{y_n}^{\mathrm{T}} \boldsymbol{x}_n}}{\sum_{k \in [\mathsf{C}]} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_n}}$$

By taking **negative log**, this is equivalent to minimizing

$$F(\boldsymbol{W}) = \sum_{n=1}^{\mathsf{N}} \ln \left(\frac{\sum_{k \in [\mathsf{C}]} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_n}}{e^{\boldsymbol{w}_{y_n}^{\mathrm{T}} \boldsymbol{x}_n}} \right) = \sum_{n=1}^{\mathsf{N}} \ln \left(1 + \sum_{k \neq y_n} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_n})^{\mathrm{T}} \boldsymbol{x}_n} \right)$$

This is the *multiclass logistic loss*, a.k.a *cross-entropy loss*.

When $\mathsf{C} = 2$, this is the same as binary logistic loss.

Step 3: Optimization

Apply **SGD**: what is the gradient of

$$g(\boldsymbol{W}) = \ln \left(1 + \sum_{k' \neq y_n} e^{(\boldsymbol{w}_{k'} - \boldsymbol{w}_{y_n})^{\mathrm{T}} \boldsymbol{x}_n} \right)?$$

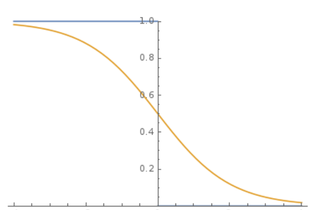
SGD for Binary Classification case (last lecture)

Recall that $\ell_{\text{logistic}}(z) = ln(1 + \exp(-z))$

$$\begin{aligned}
\boldsymbol{w} &\leftarrow \boldsymbol{w} - \eta \tilde{\nabla} F(\boldsymbol{w}) \\
&= \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \ell_{\text{logistic}}(y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) \qquad (n \in [N] \text{ is drawn u.a.r.}) \\
&= \boldsymbol{w} - \eta \left(\left. \frac{\partial \ell_{\text{logistic}}(z)}{\partial z} \right|_{z = y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n} \right) y_n \boldsymbol{x}_n \\
&= \boldsymbol{w} - \eta \left(\left. \frac{-e^{-z}}{1 + e^{-z}} \right|_{z = y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n} \right) y_n \boldsymbol{x}_n \\
&= \boldsymbol{w} + \eta \sigma(-y_n \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) y_n \boldsymbol{x}_n \\
&= \boldsymbol{w} + \eta \mathbb{P}(-y_n \mid \boldsymbol{x}_n; \boldsymbol{w}) y_n \boldsymbol{x}_n
\end{aligned}$$

This is a *soft version of Perceptron!*

$\mathbb{P}(-y_n | \boldsymbol{x}_n; \boldsymbol{w})$ versus $\mathbb{I}[y_n \neq \mathrm{sgn}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]$



Step 3: Optimization

Apply **SGD**: what is the gradient of

$$g(\boldsymbol{W}) = \ln \left(1 + \sum_{k' \neq y_n} e^{(\boldsymbol{w}_{k'} - \boldsymbol{w}_{y_n})^{\mathrm{T}} \boldsymbol{x}_n} \right)?$$

It's a $\mathsf{C} \times \mathsf{D}$ matrix. Let's focus on the k-th row:

If $k \neq y_n$:

$$\nabla_{\boldsymbol{w}_k} g(\boldsymbol{W}) = \frac{e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_n})^{\mathrm{T}} \boldsymbol{x}_n}}{1 + \sum_{k' \neq y_n} e^{(\boldsymbol{w}_{k'} - \boldsymbol{w}_{y_n})^{\mathrm{T}} \boldsymbol{x}_n}} \boldsymbol{x}_n^{\mathrm{T}} = \mathbb{P}(k \mid \boldsymbol{x}_n; \boldsymbol{W}) \boldsymbol{x}_n^{\mathrm{T}}$$

else:

$$\nabla_{\boldsymbol{w}_k} g(\boldsymbol{W}) = \frac{-\left(\sum_{k' \neq y_n} e^{(\boldsymbol{w}_{k'} - \boldsymbol{w}_{y_n})^{\mathrm{T}} \boldsymbol{x}_n} \right)}{1 + \sum_{k' \neq y_n} e^{(\boldsymbol{w}_{k'} - \boldsymbol{w}_{y_n})^{\mathrm{T}} \boldsymbol{x}_n}} \boldsymbol{x}_n^{\mathrm{T}} = \left(\mathbb{P}(y_n \mid \boldsymbol{x}_n; \boldsymbol{W}) - 1 \right) \boldsymbol{x}_n^{\mathrm{T}}$$

SGD for multinomial logistic regression

Initialize $\boldsymbol{W} = \boldsymbol{0}$ (or randomly). Repeat:

1. pick $n \in [\mathsf{N}]$ uniformly at random
2. update the parameters

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \begin{pmatrix} \mathbb{P}(y = 1 \mid \boldsymbol{x}_n; \boldsymbol{W}) \\ \vdots \\ \mathbb{P}(y = y_n \mid \boldsymbol{x}_n; \boldsymbol{W}) - 1 \\ \vdots \\ \mathbb{P}(y = \mathsf{C} \mid \boldsymbol{x}_n; \boldsymbol{W}) \end{pmatrix} \boldsymbol{x}_n^{\mathrm{T}}$$

Think about why the algorithm makes sense intuitively.

A note on prediction

Having learned $\boldsymbol{W}$, we can either

- make a *deterministic* prediction $\operatorname{argmax}_{k \in [\mathsf{C}]} \ \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}$
- make a *randomized* prediction according to $\mathbb{P}(k \mid \boldsymbol{x}; \boldsymbol{W}) \propto e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}}$

In either case, **(expected) mistake is bounded by logistic loss**

- deterministic

$$\mathbb{I}[f(\boldsymbol{x}) \neq y] \leq \ln \left(1 + \sum_{k \neq y} e^{(\boldsymbol{w}_k - \boldsymbol{w}_y)^{\mathrm{T}} \boldsymbol{x}}\right)$$

- randomized

$$\mathbb{E}\left[\mathbb{I}[f(\boldsymbol{x}) \neq y]\right] = 1 - \mathbb{P}(y \mid \boldsymbol{x}; \boldsymbol{W}) \leq -\ln \mathbb{P}(y \mid \boldsymbol{x}; \boldsymbol{W})$$

Reduce multiclass to binary

Is there an *even more general and simpler approach* to derive multiclass classification algorithms?

Given a binary classification algorithm (*any one*, not just linear methods), can we turn it to a multiclass algorithm, *in a black-box manner*?

Yes, there are in fact many ways to do it.

- **one-versus-all** (one-versus-rest, one-against-all, etc)
- **one-versus-one** (all-versus-all, etc)
- **Error-Correcting Output Codes** (ECOC)
- **tree-based reduction**

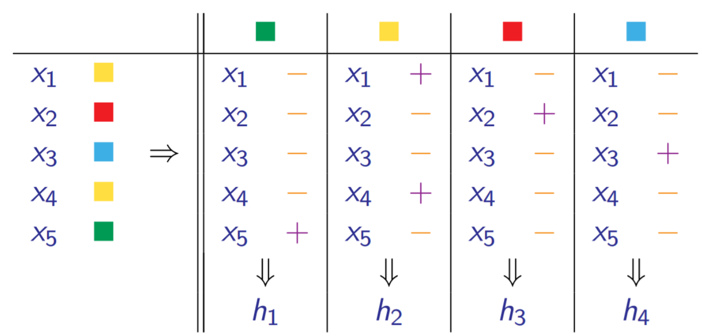
One-versus-all (OvA)

(picture credit: link)

Idea: train C binary classifiers to learn "**is class k or not?**" for each k.

Training: for each class $k \in [\mathsf{C}]$,

- re-label examples with class k as $+1$, and all others as -1
- train a binary classifier h_k using this new dataset



One-versus-all (OvA)

Prediction: for a new example $\boldsymbol{x}$

- ask each h_k: **does this belong to class k?** (i.e. $h_k(\boldsymbol{x})$)
- randomly pick among all k's s.t. $h_k(\boldsymbol{x}) = +1$.

Issue: will (probably) make a mistake *as long as one of h_k errs*.

One-versus-one (OvO) (picture credit: link)

Idea: train $\binom{\mathsf{C}}{2}$ binary classifiers to learn "**is class k or k'?**".

Training: for each pair (k, k'),

- re-label class k examples as $+1$ and class k' examples as -1
- *discard all other examples*
- train a binary classifier $h_{(k,k')}$ using this new dataset

			🟩 vs. 🟨	🟩 vs. 🟥	🟥 vs. 🟦	🟦 vs. 🟨	🟩 vs. 🟦	🟥 vs. 🟨
x_1	🟨		x_1 −			x_1 −		x_1 −
x_2	🟥			x_2 −	x_2 +			x_2 +
x_3	🟦	⇒			x_3 −	x_3 +	x_3 −	
x_4	🟨		x_4 −			x_4 −		x_4 −
x_5	🟩		x_5 +	x_5 +			x_5 +	
			⇓	⇓	⇓	⇓	⇓	⇓
			$h_{(1,2)}$	$h_{(1,3)}$	$h_{(3,4)}$	$h_{(4,2)}$	$h_{(1,4)}$	$h_{(3,2)}$

One-versus-one (OvO)

Prediction: for a new example $\boldsymbol{x}$

- ask each classifier $h_{(k,k')}$ to **vote for either class k or k'**
- predict the class with the most votes (break tie in some way)

More robust than one-versus-all, but *slower* in prediction.

Error-correcting output codes (ECOC) (picture credit: link)

Idea: based on a code $\boldsymbol{M} \in \{-1, +1\}^{\mathsf{C}\times\mathsf{L}}$, train L binary classifiers to learn "**is bit b on or off**".

Training: for each bit $b \in [\mathsf{L}]$

- re-label example x_n as $M_{y_n,b}$
- train a binary classifier h_b using this new dataset.

M	1	2	3	4	5
🟩	+	−	+	−	+
🟨	−	−	+	+	+
🟥	+	+	−	−	−
🟦	+	+	+	+	−

			1	2	3	4	5
x_1	🟨		x_1 −	x_1 −	x_1 +	x_1 +	x_1 +
x_2	🟥		x_2 +	x_2 +	x_2 −	x_2 −	x_2 −
x_3	🟦	⇒	x_3 +	x_3 +	x_3 +	x_3 +	x_3 −
x_4	🟨		x_4 −	x_4 −	x_4 +	x_4 +	x_4 +
x_5	🟩		x_5 +	x_5 −	x_5 +	x_5 −	x_5 +
			⇓	⇓	⇓	⇓	⇓
			h_1	h_2	h_3	h_4	h_5

Error-correcting output codes (ECOC)

Prediction: for a new example $\boldsymbol{x}$

- compute the **predicted code** $\boldsymbol{c} = (h_1(\boldsymbol{x}), \ldots, h_\mathsf{L}(\boldsymbol{x}))^\mathrm{T}$
- predict the class with the **most similar code**: $k = \mathrm{argmax}_k (\boldsymbol{Mc})_k$

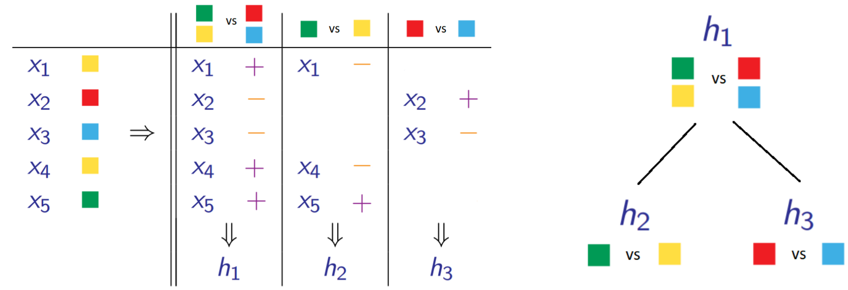
How to design the code $\boldsymbol{M}$?

- the more *dissimilar* the codes, the more robust
  - if any two codes are d bits away, then prediction can tolerate about $d/2$ errors
- *random code* is often a good choice

Tree based method

Idea: train $\approx$ C binary classifiers to learn "**belongs to which half?**".

Training: see pictures



Prediction is also natural, *but is very fast!* (think ImageNet where $C \approx 20K$)

Comparisons

In big O notation,

Reduction	**#training points**	**test time**	**remark**
OvA	CN	C	not robust
OvO	CN	C^2	can achieve very small training error
ECOC	LN	L	need diversity when designing code
Tree	$(\log_2 C)N$	$\log_2 C$	good for "extreme classification"