COMS 4771 Lecture 13

1. Beyond binary classification
2. Beyond prediction error
3. Cross-validation

Beyond binary classification

Reductions

Suppose we have a good algorithm $\mathcal{A}$ for learning binary classifiers.

Question: Can we use $\mathcal{A}$ to solve other machine learning problems?

Reductions

Suppose we have a good algorithm $\mathcal{A}$ for learning binary classifiers.

Question: Can we use $\mathcal{A}$ to solve other machine learning problems?

A **reduction** for a given machine lerning problem to binary classification is specified by two procedures:

- Instance mapping procedure: transforms samples for the given problem into samples for a binary classification problem
- Solution mapping procedure: transforms classifier f' for binary classification problem computed using $\mathcal{A}$ into a classifier f for the given problem.

Reductions

Suppose we have a good algorithm $\mathcal{A}$ for learning binary classifiers.

Question: Can we use $\mathcal{A}$ to solve other machine learning problems?

A **reduction** for a given machine lerning problem to binary classification is specified by two procedures:

- Instance mapping procedure: transforms samples for the given problem into samples for a binary classification problem
- Solution mapping procedure: transforms classifier f' for binary classification problem computed using $\mathcal{A}$ into a classifier f for the given problem.

Desired property:
If $\mathcal{A}$ finds a good binary classifier f', then f is a good classifier for the given problem.

Examples

1. **Problem**: finding high accuracy classifiers
   - **Reduction**: boosting
     (Reduces problem to finding weak classifiers.)
2. **Problem**: importance-weighted classifiation
   - **Reduction**: rejection sampling
     (Reduces problem to unweighted classification.)
3. **Problem**: multi-class classification
   - **Reduction**: One-Against-All
     (Reduces problem to binary classification.)

Importance-weighted classification

Problem:

- **Setting**: Random triple $(X, Y, W) \sim P$ for some probability distribution P over $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}_+$.
  $W =$ **importance weight** for labeled example (X, Y).
- **Goal**: Function $f : \mathcal{X} \to \mathcal{Y}$ with small **importance-weighted error**:

$$\mathbb{E}\Big[W \cdot \mathbb{1}\{f(X) \neq Y\}\Big].$$

Importance-weighted classification

Problem:

- **Setting**: Random triple $(X, Y, W) \sim P$ for some probability distribution P over $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}_+$.
  $W =$ **importance weight** for labeled example (X, Y).
- **Goal**: Function $f : \mathcal{X} \to \mathcal{Y}$ with small **importance-weighted error**:

$$\mathbb{E}\Big[W \cdot \mathbb{1}\{f(X) \neq Y\}\Big].$$

Importance-weighted classification

Problem:

- **Setting**: Random triple $(X, Y, W) \sim P$ for some probability distribution P over $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}_+$.
  $W =$ **importance weight** for labeled example (X, Y).
- **Goal**: Function $f : \mathcal{X} \to \mathcal{Y}$ with small **importance-weighted error**:

$$\mathbb{E}\Big[W \cdot \mathbb{1}\{f(X) \neq Y\}\Big].$$

Where it comes up:

- *Class-specific weights*: e.g., $W = 100 \Leftrightarrow Y = 0$ (and $W = 1$ otherwise).
- *Covariate-specific weights*: e.g., $W = 100 \Leftrightarrow X \in \mathcal{X}_0$ (and $W = 1$ o.w.).
- *Boosting*, *domain adaptation*, *causal inference*, ...

(Note: many learning algorithms natively handle importance weights.)

Importance-weighted classification

Problem:

- **Setting**: Random triple $(X, Y, W) \sim P$ for some probability distribution P over $\mathcal{X} \times \mathcal{Y} \times \mathbb{R}_+$.
  $W =$ **importance weight** for labeled example (X, Y).
- **Goal**: Function $f : \mathcal{X} \to \mathcal{Y}$ with small **importance-weighted error**:

$$\mathbb{E}\Big[W \cdot \mathbb{1}\{f(X) \neq Y\}\Big].$$

Where it comes up:

- *Class-specific weights*: e.g., $W = 100 \Leftrightarrow Y = 0$ (and $W = 1$ otherwise).
- *Covariate-specific weights*: e.g., $W = 100 \Leftrightarrow X \in \mathcal{X}_0$ (and $W = 1$ o.w.).
- *Boosting*, *domain adaptation*, *causal inference*, ...

(Note: many learning algorithms natively handle importance weights.)

Would like to reduce to (unweighted) classification.

The rejection sampling reduction

Main idea: Transform samples from P so they appear to be drawn from a distribution P', where

$$\mathbb{E}_{(X,Y,W)\sim P}\Big[W \cdot \mathbb{1}\{f(X) \neq Y\}\Big] \;\propto\; \mathbb{E}_{(X',Y')\sim P'}\Big[\mathbb{1}\{f(X') \neq Y'\}\Big].$$

The rejection sampling reduction

Main idea: Transform samples from P so they appear to be drawn from a distribution P', where

$$\mathbb{E}_{(X,Y,W)\sim P}\Big[W \cdot \mathbb{1}\{f(X) \neq Y\}\Big] \;\propto\; \mathbb{E}_{(X',Y')\sim P'}\Big[\mathbb{1}\{f(X') \neq Y'\}\Big].$$

Instance mapping procedure

Input Samples drawn from P

1: Let $w_{\max}$ be the maximum possible weight
2: **while** true **do**
3: Sample (X, Y, W) from P
4: Toss a coin with heads bias $\frac{W}{w_{\max}}$.
5: If heads, break from while loop
6: If tails, discard example.
7: **end while**
8: **return** (X, Y)

The rejection sampling reduction

Main idea: Transform samples from P so they appear to be drawn from a distribution P', where

$$\mathbb{E}_{(X,Y,W)\sim P}\Big[W \cdot \mathbb{1}\{f(X) \neq Y\}\Big] \;\propto\; \mathbb{E}_{(X',Y')\sim P'}\Big[\mathbb{1}\{f(X') \neq Y'\}\Big].$$

Instance mapping procedure

Input Samples drawn from P

1: Let $w_{\max}$ be the maximum possible weight
2: **while** true **do**
3: Sample (X, Y, W) from P
4: Toss a coin with heads bias $\frac{W}{w_{\max}}$.
5: If heads, break from while loop
6: If tails, discard example.
7: **end while**
8: **return** (X, Y)

Solution mapping procedure: identity map

Multi-class classification

Problem:

- **Setting**: Random pair $(X, Y) \sim P$ for some probability distribution P over $\mathcal{X} \times \{1, 2, \ldots, K\}$.
- **Goal**: Function $f : \mathcal{X} \to \mathcal{Y}$ with small prediction error $\Pr(f(X) \neq Y)$.

Multi-class classification

Problem:

- **Setting**: Random pair $(X, Y) \sim P$ for some probability distribution P over $\mathcal{X} \times \{1, 2, \ldots, K\}$.
- **Goal**: Function $f : \mathcal{X} \to \mathcal{Y}$ with small prediction error $\Pr(f(X) \neq Y)$.

Would like to reduce to binary classification.

One-Against-All reduction

Main idea: Create K binary classification problems

given $x \in \mathcal{X}$, predict whether or not $y = i$.

From each sample (x, y) drawn from P, create K samples, one for each binary classification problem:

$$(x, y) \longrightarrow \begin{cases} (x, \mathbb{1}\{y = 1\}) & \longrightarrow & \text{Binary classification problem } 1 \\ (x, \mathbb{1}\{y = 2\}) & \longrightarrow & \text{Binary classification problem } 2 \\ \vdots & \vdots & \vdots \\ (x, \mathbb{1}\{y = K\}) & \longrightarrow & \text{Binary classification problem } K \end{cases}$$

One-Against-All reduction

Instance mapping procedure

Input Sample $(X, Y) \in \mathcal{X} \times \{1, 2, \ldots, K\}$ drawn from P

1: **for** each $i = 1, 2, \ldots, K$ **do**
2: Generate sample $(X, \mathbb{1}\{Y = i\}) \in \mathcal{X} \times \{0, 1\}$ for binary classification problem i
3: **end for**

One-Against-All reduction

Instance mapping procedure

Input Sample $(X, Y) \in \mathcal{X} \times \{1, 2, \ldots, K\}$ drawn from P

1: **for** each $i = 1, 2, \ldots, K$ **do**
2: Generate sample $(X, \mathbb{1}\{Y = i\}) \in \mathcal{X} \times \{0, 1\}$ for binary classification problem i
3: **end for**

Solution mapping procedure

Input K binary predictors $f'_1, f'_2, \ldots, f'_K : \mathcal{X} \to \{0, 1\}$.

return Function $f : \mathcal{X} \to \{1, 2, \ldots, K\}$ where

$$f(x) = \operatorname*{arg\,max}_{i \in \{1,2,\ldots,K\}} f'_i(x) \quad \text{(breaking ties arbitrarily)}.$$

Recap

- **Reductions**: reuse existing technology to solve new problems.
  - Multi-class
  - Multi-label prediction
  - Ranking
  - Sequence prediction
  - ...
- **Lots of different problems and objectives** beyond binary classificaiton and prediction error—can be application-/domain-specific.

Beyond prediction error

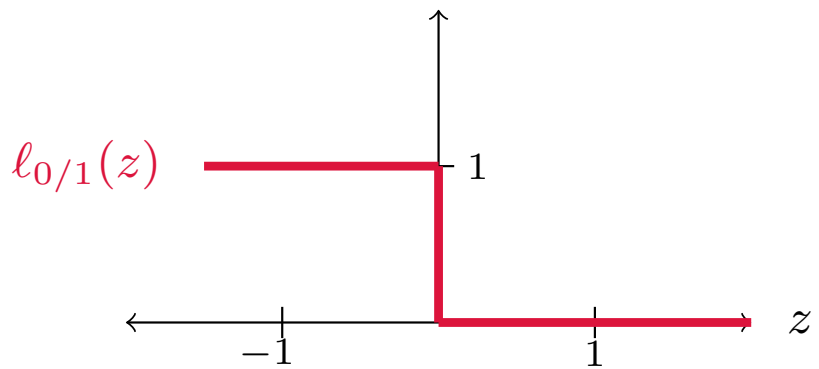
Prediction error / zero-one loss

P is a distribution over $\mathcal{X} \times \{-1, +1\}$, and $(X, Y) \sim P$.

For any classifier $f : \mathcal{X} \to \{-1, +1\}$,

$$\mathrm{err}(f) \;=\; \Pr\Big[f(X) \neq Y\Big] \;=\; \mathbb{E}\Big[\ell_{0/1}(Y f(X))\Big].$$

$$\ell_{0/1}(x) = (1 - \mathrm{sign}(x))/2$$



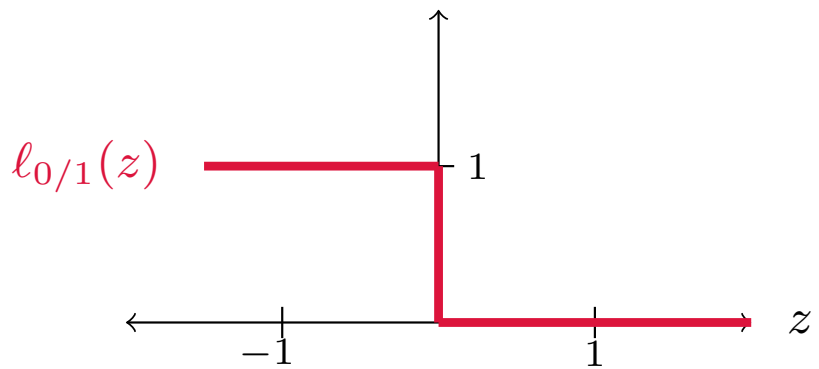
Prediction error / zero-one loss

P is a distribution over $\mathcal{X} \times \{-1, +1\}$, and $(X, Y) \sim P$.

For any classifier $f : \mathcal{X} \to \{-1, +1\}$,

$$\mathrm{err}(f) \;=\; \Pr\Big[f(X) \neq Y\Big] \;=\; \mathbb{E}\Big[\ell_{0/1}(Y f(X))\Big].$$

$$\ell_{0/1}(x) = (1 - \mathrm{sign}(x))/2$$



Also works with **real-valued predictors** $f : \mathcal{X} \to \mathbb{R}$; where the prediction is $\mathrm{sign}(f(\boldsymbol{x}))$, for example:

- **Linear classifiers**: $\boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle - \theta$.
- **(Binary) plug-in classifiers**: $x \mapsto \widehat{\Pr}[Y = +1 \,|\, X = x] - 1/2$.

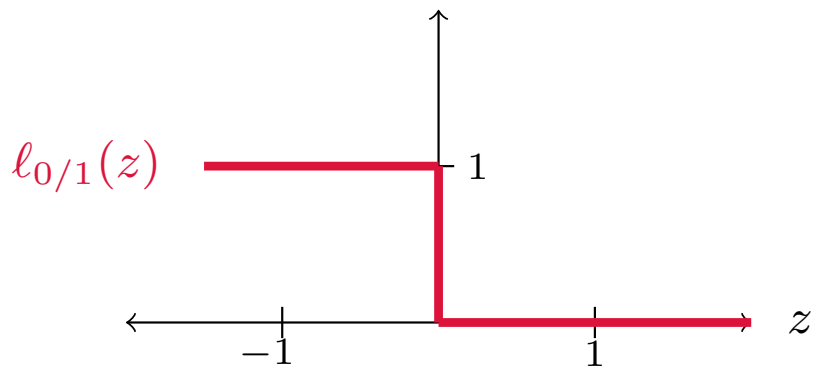
Prediction error / zero-one loss

P is a distribution over $\mathcal{X} \times \{-1, +1\}$, and $(X, Y) \sim P$.

For any classifier $f : \mathcal{X} \to \{-1, +1\}$,

$$\mathrm{err}(f) \;=\; \Pr\Big[f(X) \neq Y\Big] \;=\; \mathbb{E}\Big[\ell_{0/1}(Y f(X))\Big].$$

$$\ell_{0/1}(x) = (1 - \mathrm{sign}(x))/2$$



Also works with **real-valued predictors** $f : \mathcal{X} \to \mathbb{R}$; where the prediction is $\mathrm{sign}(f(\boldsymbol{x}))$, for example:

- **Linear classifiers**: $\boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle - \theta$.
- **(Binary) plug-in classifiers**: $x \mapsto \widehat{\Pr}[Y = +1 \,|\, X = x] - 1/2$.

Often useful to adjust threshold (e.g., θ and $1/2$ above).

Adjusting Thresholds

Often have **different costs for different kinds of mistakes**:

	$f(X) \leq t$	$f(X) > t$
$Y = -1$	0	a
$Y = +1$	b	0

Adjusting Thresholds

Often have **different costs for different kinds of mistakes**:

	$f(X) \leq t$	$f(X) > t$
$Y = -1$	0	a
$Y = +1$	b	0

If $b \gg a$, might want to decrease t.

Adjusting Thresholds

Often have **different costs for different kinds of mistakes**:

	$f(X) \leq t$	$f(X) > t$
$Y = -1$	0	a
$Y = +1$	b	0

If $b \gg a$, might want to decrease t.

If $a \gg b$, might want to increase t.

Adjusting Thresholds

Also, often interested in **different performance criteria**: given test set T,

- **Precision**:
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, y) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}$$
- **Recall** (a.k.a. **Sensitivity**, **True Positive Rate**):
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, 1) \in T\}|}$$
- **False Positive Rate**:
$$\frac{|\{(\boldsymbol{x}, -1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, -1) \in T\}|}$$

Adjusting Thresholds

Also, often interested in **different performance criteria**: given test set T,

- **Precision**:
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, y) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}$$
- **Recall** (a.k.a. **Sensitivity**, **True Positive Rate**):
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, 1) \in T\}|}$$
- **False Positive Rate**:
$$\frac{|\{(\boldsymbol{x}, -1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, -1) \in T\}|}$$

Precision generally improves if we increase t.

Adjusting Thresholds

Also, often interested in **different performance criteria**: given test set T,

- **Precision**:
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, y) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}$$
- **Recall** (a.k.a. **Sensitivity**, **True Positive Rate**):
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, 1) \in T\}|}$$
- **False Positive Rate**:
$$\frac{|\{(\boldsymbol{x}, -1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, -1) \in T\}|}$$

Precision generally improves if we increase t.

Recall improves if we decrease t.

Adjusting Thresholds

Also, often interested in **different performance criteria**: given test set T,

- **Precision**:
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, y) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}$$
- **Recall** (a.k.a. **Sensitivity**, **True Positive Rate**):
$$\frac{|\{(\boldsymbol{x}, 1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, 1) \in T\}|}$$
- **False Positive Rate**:
$$\frac{|\{(\boldsymbol{x}, -1) \in T \text{ s.t. } f(\boldsymbol{x}) > t\}|}{|\{(\boldsymbol{x}, -1) \in T\}|}$$

Precision generally improves if we increase t.

Recall improves if we decrease t.

False positive rate improves if we increase t.

Conditional probability estimation

Sometimes would like real-valued predictor f to be related to the **conditional probability function** η:

$$\eta(x) := \Pr[Y = +1 \,|\, X = x].$$

Conditional probability estimation

Sometimes would like real-valued predictor f to be related to the **conditional probability function** η:

$$\eta(x) := \Pr[Y = +1 \,|\, X = x].$$

Option #1: Can use **generative models** to construct **plug-in classifier**, but could fail if class conditional distributions are not accurate.

Conditional probability estimation

Sometimes would like real-valued predictor f to be related to the **conditional probability function** η:

$$\eta(x) := \Pr[Y = +1 \,|\, X = x].$$

Option #1: Can use **generative models** to construct **plug-in classifier**, but could fail if class conditional distributions are not accurate.

Option #2: Can use a loss function that is minimized by η (or some invertible transformation thereof).

Conditional probability estimation

Sometimes would like real-valued predictor f to be related to the **conditional probability function** η:

$$\eta(x) := \Pr[Y = +1 \,|\, X = x].$$

Option #1: Can use **generative models** to construct **plug-in classifier**, but could fail if class conditional distributions are not accurate.

Option #2: Can use a loss function that is minimized by η (or some invertible transformation thereof).

I.e. find a function $f : \mathcal{X} \to \mathbb{R}$ that minimizes

$$\frac{1}{n} \sum_{i=1}^{n} \ell(y_i f(\boldsymbol{x}_i))$$

for some function ℓ that is different from $\ell_{0/1}$.

Eliciting conditional probabilities

Goal: loss function that is minimized by (some invertible transformation of) the conditional probability function

$$\eta(x) = \Pr[Y = +1 \mid X = x].$$

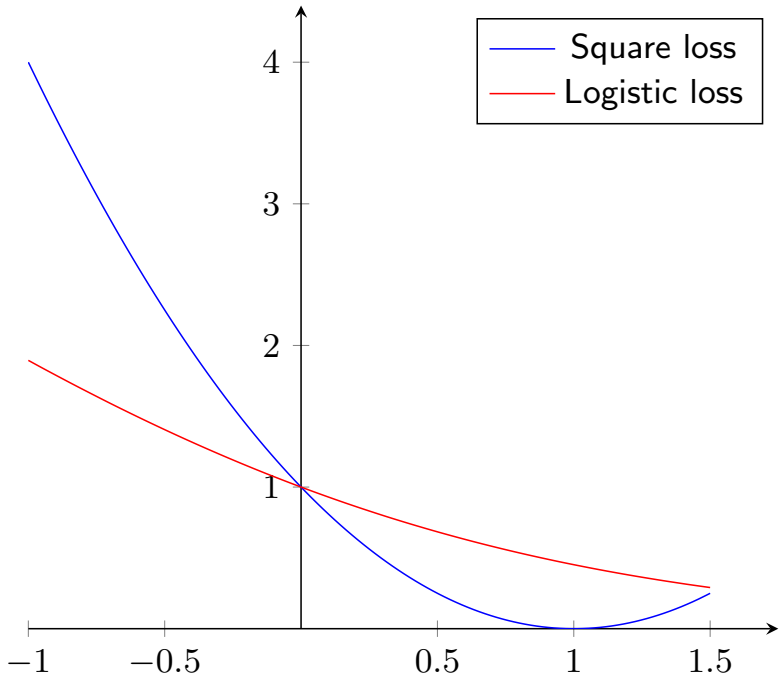
(Y takes values in $\{-1, +1\}$.)

Eliciting conditional probabilities

Goal: loss function that is minimized by (some invertible transformation of) the conditional probability function

$$\eta(x) = \Pr[Y = +1 \mid X = x].$$

(Y takes values in $\{-1, +1\}$.)



- **Square loss**: $\ell_{\mathrm{sq}}(z) = (1 - z)^2$

  $\mathbb{E}[\ell_{\mathrm{sq}}(Y f(x)) | X = x]$ is minimized by $f(x) = 2\eta(x) - 1$.

- **Logistic loss**: $\ell_{\log}(z) = \ln(1 + \exp(-z))$

  $\mathbb{E}[\ell_{\log}(Y f(x)) | X = x]$ is minimized by $f(x) = \ln\left(\frac{\eta(x)}{1-\eta(x)}\right)$.

Eliciting conditional probabilities

Suppose we assume a linear functional form for f, i.e. $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$

(assume bias incorporated into $\boldsymbol{w}$ by adding a constant 1 feature to examples if necessary)

Using loss functions: if loss function $\ell : \mathbb{R} \to \mathbb{R}$ is convex, then ERM (given below) is a convex problem:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)$$

Eliciting conditional probabilities

Suppose we assume a linear functional form for f, i.e. $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$

(assume bias incorporated into $\boldsymbol{w}$ by adding a constant 1 feature to examples if necessary)

Using loss functions: if loss function $\ell : \mathbb{R} \to \mathbb{R}$ is convex, then ERM (given below) is a convex problem:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)$$

Good news: Both ℓ_{sq} and $\ell_{\log}$ are convex!

Eliciting conditional probabilities

Suppose we assume a linear functional form for f, i.e. $f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$

(assume bias incorporated into $\boldsymbol{w}$ by adding a constant 1 feature to examples if necessary)

Using loss functions: if loss function $\ell : \mathbb{R} \to \mathbb{R}$ is convex, then ERM (given below) is a convex problem:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)$$

Good news: Both ℓ_{sq} and ℓ_{log} are convex!

In practice, usually incorporate (convex) regularization function R (e.g. $R(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|^2$) to avoid overfitting:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \quad R(\boldsymbol{w}) + \frac{1}{n} \sum_{i=1}^{n} \ell(y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)$$

Eliciting conditional probabilities

A non-example:

- **Hinge loss**: $\ell_{\mathrm{hl}}(z) = \max\{0, 1 - z\}$

  $\mathbb{E}[\ell_{\mathrm{hl}}(Y f(x))|X = x]$ is minimized by $f(x) = \mathrm{sign}(2\eta(x) - 1)$.

  Can't recover $\eta(x)$ from $f(x)$.

Eliciting conditional probabilities

A non-example:

- **Hinge loss**: $\ell_{\mathrm{hl}}(z) = \max\{0, 1 - z\}$

  $\mathbb{E}[\ell_{\mathrm{hl}}(Y f(x))|X = x]$ is minimized by $f(x) = \mathrm{sign}(2\eta(x) - 1)$.

  Can't recover $\eta(x)$ from $f(x)$.

Caveat: Might not be possible to represent

$$\boldsymbol{x} \mapsto 2\eta(\boldsymbol{x}) - 1 \quad \text{or} \quad \boldsymbol{x} \mapsto \ln\left(\frac{\eta(\boldsymbol{x})}{1 - \eta(\boldsymbol{x})}\right)$$

as (say) a linear function $\boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle$.

Eliciting conditional probabilities

A non-example:

- **Hinge loss**: $\ell_{\mathrm{hl}}(z) = \max\{0, 1 - z\}$

  $\mathbb{E}[\ell_{\mathrm{hl}}(Y f(x))|X = x]$ is minimized by $f(x) = \mathrm{sign}(2\eta(x) - 1)$.

  Can't recover $\eta(x)$ from $f(x)$.

Caveat: Might not be possible to represent

$$\boldsymbol{x} \mapsto 2\eta(\boldsymbol{x}) - 1 \quad \text{or} \quad \boldsymbol{x} \mapsto \ln\left(\frac{\eta(\boldsymbol{x})}{1 - \eta(\boldsymbol{x})}\right)$$

as (say) a linear function $\boldsymbol{x} \mapsto \langle \boldsymbol{w}, \boldsymbol{x} \rangle$.

Common solutions: enhance the feature space via feature expansion or kernels, or use more flexible models (e.g., tree models).

Cross validation

Model selection

Objective

- Often necessary to consider many different models for a given problem (e.g., class conditional distributions in generative model classifiers, features in linear classifiers, kernel in kernelized classifiers).
- Sometimes "model" simply means particular setting of **hyper-parameters** (e.g., k in k-NN, λ in soft-margin SVM, number of nodes in decision tree).

Terminology

The problem of choosing a good model is called **model selection**.

Example: SVM with Gaussian kernel

Soft-margin SVM with Gaussian kernel

- Models indexed by regularization parameter λ and Gaussian kernel bandwidth $h > 0$:

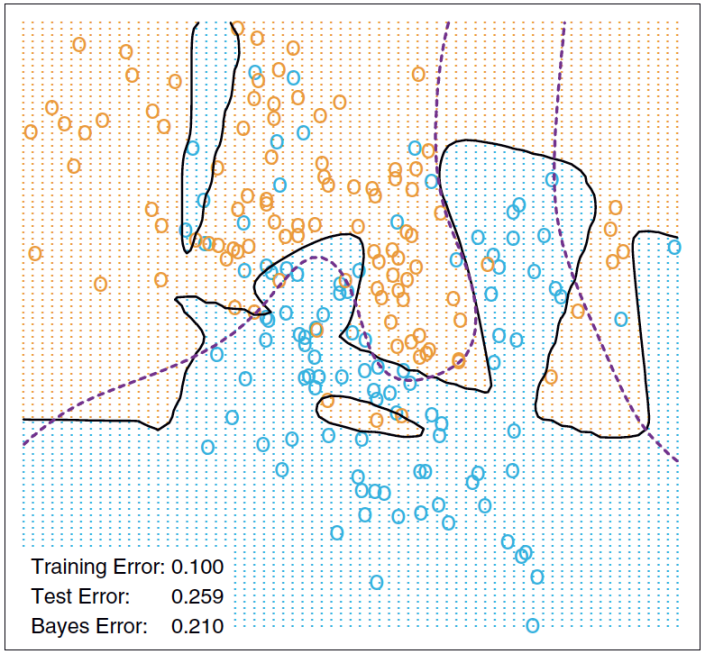
$$K(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \exp\left(-\frac{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2}{2h}\right).$$

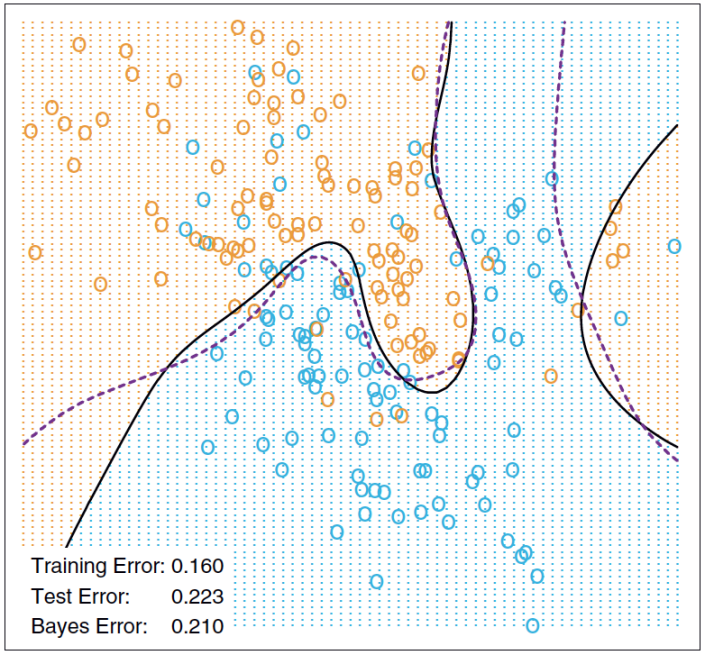
- Goal is to find setting of (λ, h) for which we can expect small true (generalization) error.

Naïve approach

- Also minimize over (λ, h) in SVM optimization problem.
- Leads to **overfitting**: resulting SVM classifier adapts too closely to specific properties of the training data, rather than underlying distribution.

Overfitting: illustration





- Classifier in this example has bandwidth parameter σ (similar to Gaussian kernel bandwidth).
- Small $\sigma \rightarrow$ permits curve with sharp bends
- Large $\sigma \rightarrow$ smoother curve.

Model selection by hold-out validation

(Henceforth, use $\boldsymbol{h}$ to denote particular setting of hyper-parameters / model choice.)

Hold-out validation

Model selection:

1. Randomly split data into three sets: training, validation, and test data.

Training	Validation	Test

2. Train classifier $\hat{f}_{\boldsymbol{h}}$ on Training data for different values of $\boldsymbol{h}$.
3. Compute Validation ("hold-out") error for each $\hat{f}_{\boldsymbol{h}}$: $\mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Validation})$.
4. Selection: $\hat{\boldsymbol{h}} =$ value of $\boldsymbol{h}$ with lowest Validation error.
5. Train classifier $\hat{f}$ using $\hat{\boldsymbol{h}}$ with Training + Validation data.

Model assessment:

6. Finally: estimate the error of $\hat{f}$ using test data.

Main idea behind hold-out validation

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Validation})$.

Training + Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training + Validation data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Test})$.

Main idea behind hold-out validation

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Validation})$.

Training + Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training + Validation data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Test})$.

The hope is that these quantities are similar!

Main idea behind hold-out validation

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Validation})$.

Training + Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training + Validation data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Test})$.

The hope is that these quantities are similar!

(Making this rigorous is actually rather tricky.)

Beyond simple hold-out validation

Standard hold-out:

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Validation})$.

Beyond simple hold-out validation

Standard hold-out:

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Validation})$.

Could also:

- train $\hat{f}_{\boldsymbol{h}}$ using Validation data, and
- evaluate $\hat{f}_{\boldsymbol{h}}$ using Training data.

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Validation data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Training})$.

Beyond simple hold-out validation

Standard hold-out:

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Training data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Validation})$.

Could also:

- train $\hat{f}_{\boldsymbol{h}}$ using Validation data, and
- evaluate $\hat{f}_{\boldsymbol{h}}$ using Training data.

Training	Validation	Test

Classifier $\hat{f}_{\boldsymbol{h}}$ trained on Validation data $\longrightarrow \mathrm{err}(\hat{f}_{\boldsymbol{h}}, \text{Training})$.

Idea: *Do both*, and average results as overall validation error for $\boldsymbol{h}$.

Model selection by K-fold cross validation

Model selection:

1. Set aside some test data.
2. Of remaining data, split into K parts ("folds", usually 5 or 10) $S_1, S_2, \ldots, S_K$.
3. For each value of $\boldsymbol{h}$:
   - For each $k \in \{1, 2, \ldots, K\}$:
     - Train classifier $\hat{f}_{\boldsymbol{h},k}$ using all S_i except S_k.
     - Evaluate classifier $\hat{f}_{\boldsymbol{h},k}$ using S_k: $\mathrm{err}(\hat{f}_{\boldsymbol{h},k}, S_k)$

     Example: $K = 5$ and $k = 4$

Training	Training	Training	Validation	Training

   - Cross-validation error for $\boldsymbol{h}$: $\frac{1}{K} \sum_{k=1}^{K} \mathrm{err}(\hat{f}_{\boldsymbol{h},k}, S_k)$.
4. Select the value $\hat{\boldsymbol{h}}$ with lowest cross-validation error.
5. Train classifier $\hat{f}$ using selected $\hat{\boldsymbol{h}}$ with all $S_1, S_2, \ldots, S_K$.

Model assessment:

6. Finally: estimate the error of $\hat{f}$ using test data.

Recap

- **Model selection**: goal is to pick best model (e.g., features, kernels, hyper-parameter settings) to achieve low true error.

Recap

- **Model selection**: goal is to pick best model (e.g., features, kernels, hyper-parameter settings) to achieve low true error.
- **Two common methods**: hold-out validation and K-fold cross validation (with $K = 5$ or $K = 10$).

Recap

- **Model selection**: goal is to pick best model (e.g., features, kernels, hyper-parameter settings) to achieve low true error.
- **Two common methods**: hold-out validation and K-fold cross validation (with $K = 5$ or $K = 10$).
- **Caution**: considering *too many* different models can lead to overfitting, even with hold-out / cross-validation.