Classification odds&ends

CS 446
1. The perceptron algorithm
Recall the (soft/nonseparable) **primal** SVM problem:

\[
\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} [1 - y_i w^T x_i]^+. 
\]
Recall the (soft/nonseparable) primal SVM problem:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{n} [1 - y_i \mathbf{w}^T \mathbf{x}_i]^+.
\]

For convenience, let’s write it as regularized ERM:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{\lambda}{2} \| \mathbf{w} \|^2 + \frac{1}{n} \sum_{i=1}^{n} [1 - y_i \mathbf{w}^T \mathbf{x}_i]^+.
\]
Recall the (soft/nonseparable) **primal** SVM problem:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{n} [1 - y_i \mathbf{w}^T \mathbf{x}_i]^+.
\]

For convenience, let's write it as regularized ERM:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{\lambda}{2} \| \mathbf{w} \|^2 + \frac{1}{n} \sum_{i=1}^{n} [1 - y_i \mathbf{w}^T \mathbf{x}_i]^+.
\]

The (full batch) gradient descent update is
Recall the (soft/nonseparable) **primal** SVM problem:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} [1 - y_i \mathbf{w}^T \mathbf{x}_i]_+.
\]

For convenience, let's write it as regularized ERM:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} [1 - y_i \mathbf{w}^T \mathbf{x}_i]_+.
\]

The (full batch) gradient descent update is

\[
\mathbf{w}' := \mathbf{w} - \eta \left( \lambda \mathbf{w} - \frac{1}{n} \sum_{i=1}^{n} 1[1 - y_i \mathbf{w}^T \mathbf{x}_i \geq 0]y_i \mathbf{x}_i \right)
\]

\[
= (1 - \eta \lambda) \mathbf{w} + \eta \frac{1}{n} \sum_{i=1}^{n} 1[1 - y_i \mathbf{w}^T \mathbf{x}_i \geq 0]y_i \mathbf{x}_i.
\]

The stochastic gradient descent (SGD) update is
SVM primal recap

Recall the (soft/nonseparable) primal SVM problem:

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} [1 - y_i w^T x_i]^+. $$

For convenience, let’s write it as regularized ERM:

$$\min_{w \in \mathbb{R}^d} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} [1 - y_i w^T x_i]^+. $$

The (full batch) gradient descent update is

$$w' := w - \eta \left( \lambda w - \frac{1}{n} \sum_{i=1}^{n} 1[1 - y_i w^T x_i \geq 0] y_i x_i \right)$$

$$= (1 - \eta \lambda) w + \eta \frac{1}{n} \sum_{i=1}^{n} 1[1 - y_i w^T x_i \geq 0] y_i x_i.$$ 

The stochastic gradient descent (SGD) update is

$$w' := w - \eta \left( \lambda w - 1[1 - y w^T x \geq 0] y x \right)$$

$$= (1 - \eta \lambda) w + \eta 1[1 - y w^T x \geq 0] y x,$$

where \((x, y)\) is drawn uniformly at random from the training set.
Regularized ERM formulation:

$$\min_{w \in \mathbb{R}^d} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} [1 - y_i w^T x_i]_+.$$  

SGD update: sample \((x, y)\), and update

$$w' = (1 - \eta \lambda)w + \eta 1[1 - yw^T x \geq 0]yx.$$  

Question: what happens if \(\lambda \downarrow 0\)?
SVM primal SGD

Regularized ERM formulation:

\[
\min_{w \in \mathbb{R}^d} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} [1 - y_i w^T x_i]^+.
\]

SGD update: sample \((x, y)\), and update

\[
w' = (1 - \eta \lambda)w + \eta \mathbb{1}[1 - yw^T x \geq 0]yx.
\]

**Question**: what happens if \(\lambda \downarrow 0\)?

- Nothing keeps \(\|w\|\) small.
- \(\mathbb{1}[1 - yw^T x \geq 0]\) might as well be \(\mathbb{1}[-yw^T x \geq 0]\).

(Note: heuristic derivation; doesn’t correspond to \(C \uparrow \infty\).)
SVM primal SGD

Regularized ERM formulation:

$$\min_{w \in \mathbb{R}^d} \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} [1 - y_i w^T x_i]^+. \quad (1)$$

SGD update: sample \((x, y)\), and update

$$w' = (1 - \eta \lambda)w + \eta 1[yw^T x \geq 0]yx. \quad (2)$$

Question: what happens if \(\lambda \downarrow 0\)?

- Nothing keeps \(\|w\|\) small.
- \(1[yw^T x \geq 0]\) might as well be \(1[-yw^T x \geq 0]\).

(Note: heuristic derivation; doesn’t correspond to \(C \uparrow \infty\).)

Together, this gives the perceptron algorithm:
initialize with \(w := 0\), and thereafter set

$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$
The Perceptron Algorithm

Perceptron update (Rosenblatt ’58): initialize $w := 0$, and thereafter

$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$ 

Remarks.
The Perceptron Algorithm

Perceptron update (Rosenblatt ’58): initialize $w := 0$, and thereafter

$$w \leftarrow w + 1[yw^Tx \leq 0]yx.$$ 

Remarks.

- Can interpret algorithm as:
  
  *either we are correct with a margin ($yw^Tx > 0$) and we do nothing,*
  
  *or we are not and update $w \leftarrow w + yx$.*
The Perceptron Algorithm

Perceptron update (Rosenblatt '58): initialize \( w := 0 \), and thereafter

\[
\mathbf{w} \leftarrow \mathbf{w} + \mathbf{y} \mathbf{w}^T \mathbf{x} \leq 0 \mathbf{y} \mathbf{x}.
\]

Remarks.

- Can interpret algorithm as:
  
  *either* we are correct with a margin \((\mathbf{y} \mathbf{w}^T \mathbf{x} > 0)\) and we do nothing,

  *or* we are not and update \( \mathbf{w} \leftarrow \mathbf{w} + \mathbf{y} \mathbf{x} \).

- Therefore: if we update, we do so by rotating towards \( \mathbf{y} \mathbf{x} \).
Perceptron update (Rosenblatt ’58): initialize $\mathbf{w} := 0$, and thereafter
\[ \mathbf{w} \leftarrow \mathbf{w} + 1[y\mathbf{w}^\top \mathbf{x} \leq 0]yx. \]

Remarks.

- Can interpret algorithm as:
  *either* we are correct with a margin ($yw^\top x > 0$) and we do nothing,
  *or* we are not and update $\mathbf{w} \leftarrow \mathbf{w} + yx$.

- Therefore: if we update, we do so by rotating towards $yx$.

- This makes sense: $(\mathbf{w} + yx)^\top (yx) = \mathbf{w}^\top (yx) + \|x\|^2$;
  i.e., we increase $\mathbf{w}^\top (yx)$. 

Not obvious that Perceptron will eventually terminate! (We’ll return to this.)
The Perceptron Algorithm

Perceptron update (Rosenblatt ’58): initialize $\mathbf{w} := 0$, and thereafter

$$\mathbf{w} \leftarrow \mathbf{w} + 1[y \mathbf{w}^T \mathbf{x} \leq 0]y \mathbf{x}.$$ 

Remarks.

- Can interpret algorithm as:
  
  *either* we are correct with a margin ($y \mathbf{w}^T \mathbf{x} > 0$) and we do nothing,
  
  *or* we are not and update $\mathbf{w} \leftarrow \mathbf{w} + y \mathbf{x}$.

- Therefore: if we update, we do so by rotating towards $y \mathbf{x}$.

- This makes sense: $(\mathbf{w} + y \mathbf{x})^T (y \mathbf{x}) = \mathbf{w}^T (y \mathbf{x}) + ||\mathbf{x}||^2$; i.e., we increase $\mathbf{w}^T (y \mathbf{x})$.

Scenario 1

Current vector $\hat{\mathbf{w}}_t$ comparable to $\mathbf{x}_t$ in length.
The Perceptron Algorithm

Perceptron update (Rosenblatt '58): initialize $w := 0$, and thereafter
$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$

Remarks.

- Can interpret algorithm as:
  *either* we are correct with a margin ($yw^T x > 0$) and we do nothing,
  *or* we are not and update $w \leftarrow w + yx$.

- Therefore: if we update, we do so by rotating towards $yx$.

- This makes sense: $(w + yx)^T(yx) = w^T(yx) + \|x\|^2$; i.e., we increase $w^T(yx)$.

Scenario 1

Updated vector $\hat{w}_{t+1}$ now correctly classifies $(x_t, y_t)$. 

Not obvious that Perceptron will eventually terminate! (We'll return to this.)
The Perceptron Algorithm

Perceptron update (Rosenblatt '58): initialize \( \mathbf{w} := 0 \), and thereafter

\[
\mathbf{w} \leftarrow \mathbf{w} + \mathbf{1}[yw^T x \leq 0]yx.
\]

Remarks.

- Can interpret algorithm as:
  
either we are correct with a margin \((yw^T x > 0)\) and we do nothing,
  
or we are not and update \( \mathbf{w} \leftarrow \mathbf{w} + yx \).

- Therefore: if we update, we do so by rotating towards \( yx \).

- This makes sense: \((\mathbf{w} + yx)^T(yx) = \mathbf{w}^T(yx) + \|x\|^2\); i.e., we increase \( \mathbf{w}^T(yx) \).

Scenario 2

Current vector \( \hat{\mathbf{w}}_t \) much longer than \( \mathbf{x}_t \).
The Perceptron Algorithm

Perceptron update (Rosenblatt '58): initialize $w := 0$, and thereafter

$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$  

Remarks.

- Can interpret algorithm as:
  - either we are correct with a margin ($yw^T x > 0$) and we do nothing,
  - or we are not and update $w \leftarrow w + yx$.
- Therefore: if we update, we do so by rotating towards $yx$.
- This makes sense: $(w + yx)^T (yx) = w^T (yx) + \|x\|^2$; i.e., we increase $w^T (yx)$.

Scenario 2

Updated vector $\hat{w}_{t+1}$ does not correctly classify $(x_t, y_t)$.
The Perceptron Algorithm

Perceptron update (Rosenblatt ’58): initialize $\mathbf{w} := \mathbf{0}$, and thereafter

$$\mathbf{w} \leftarrow \mathbf{w} + 1[y\mathbf{w}^T\mathbf{x} \leq 0]y\mathbf{x}.$$  

Remarks.

- Can interpret algorithm as:  
  either we are correct with a margin ($y\mathbf{w}^T\mathbf{x} > 0$) and we do nothing,  
  or we are not and update $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$.
- Therefore: if we update, we do so by rotating towards $y\mathbf{x}$.
- This makes sense: $(\mathbf{w} + y\mathbf{x})^T(y\mathbf{x}) = \mathbf{w}^T(y\mathbf{x}) + ||\mathbf{x}||^2$;  
  i.e., we increase $\mathbf{w}^T(y\mathbf{x})$.

**Scenario 2**

Updated vector $\hat{\mathbf{w}}_{t+1}$ does not correctly classify $(\mathbf{x}_t, y_t)$.

Not obvious that Perceptron will eventually terminate! (We’ll return to this.)
2. **Kernelizing Perceptron**
Perceptron training: start with \( w_0 \leftarrow 0 \), and thereafter set

\[
    w_i \leftarrow w_{i-1} + \mathbb{1}[y_i w_{i-1}^T x_i \leq 0]y_i x_i.
\]

Define margin violation set \( V_i := \{ j \leq i : y_j w_j^T x_j \leq 0 \} \).

Then:

\[
    w_i \leftarrow \sum_{j \in V_i} x_j y_j.
\]

Can rewrite algorithm with \( V_i \):

▶ Predict \( \hat{y}_i := w_i^T x_i = \sum_{j \in V_i} y_j x_j^T \).

▶ Update \( V_i \leftarrow V_i - 1 \cup \{ i \} \) if margin violation \( \hat{y}_i y_i \leq 0 \), else \( V_i \leftarrow V_i - 1 \).

\( x \) only appears in form \( x^T x' \)!

Kernelized Perceptron:

▶ Predict \( \hat{y}_i := \sum_{j \in V_i - 1} y_j \phi(x_j)^T \phi(x_i) = \sum_{j \in V_i - 1} y_j k(x_j, x_i) \).

▶ Update \( V_i \leftarrow V_i - 1 \cup \{ i \} \) if margin violation \( \hat{y}_i y_i \leq 0 \), else \( V_i \leftarrow V_i - 1 \).
Perceptron training: start with $\mathbf{w}_0 \leftarrow 0$, and thereafter set

$$
\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \mathbb{1}[y_i \mathbf{w}_{i-1}^\top \mathbf{x}_i \leq 0] y_i \mathbf{x}_i.
$$

Define margin violation set $\mathcal{V}_i := \{ j \leq i : y_j \mathbf{w}_{j-1}^\top \mathbf{x}_j \leq 0 \}$. Then:

$$
\mathbf{w}_i := \sum_{j \in \mathcal{V}_i} \mathbf{x}_j y_j.
$$

Can rewrite algorithm with $\mathcal{V}_i$:

- **Predict** $\hat{y}_i := \mathbf{w}_{i-1}^\top \mathbf{x}_i = \sum_{j \in \mathcal{V}_{i-1}} y_j \mathbf{x}_j^\top \mathbf{x}_i$.

- **Update** $\mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \cup \{ i \}$ if margin violation $\hat{y}_i y_i \leq 0$, else $\mathcal{V}_i \leftarrow \mathcal{V}_{i-1}$.
Kernelizing Perceptron

**Perceptron training:** start with \( w_0 \leftarrow 0 \), and thereafter set

\[
    w_i \leftarrow w_{i-1} + 1[y_i w_{i-1}^T x_i \leq 0] y_i x_i.
\]

Define margin violation set \( \mathcal{V}_i := \{ j \leq i : y_j w_{j-1}^T x_j \leq 0 \} \).

Then:

\[
    w_i := \sum_{j \in \mathcal{V}_i} x_j y_j.
\]

Can rewrite algorithm with \( \mathcal{V}_i \):

- **Predict** \( \hat{y}_i := w_{i-1}^T x_i = \sum_{j \in \mathcal{V}_{i-1}} y_j x_j^T x_i \).
- **Update** \( \mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \cup \{ i \} \) if margin violation \( \hat{y}_i y_i \leq 0 \), else \( \mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \).

\( x \) only appears in form \( x^T x' \)! **Kernelized Perceptron:**

- **Predict** \( \hat{y}_i := \sum_{j \in \mathcal{V}_{i-1}} y_j \phi(x_j)^T \phi(x_i) = \sum_{j \in \mathcal{V}_{i-1}} y_j k(x_j, x_i) \).
- **Update** \( \mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \cup \{ i \} \) if margin violation \( \hat{y}_i y_i \leq 0 \), else \( \mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \).
Kernelized perceptron vs Kernelized SVM

Kernelized Perceptron:

- **Predict** \( \hat{y}_i := \sum_{j \in V_{i-1}} y_j \phi(x_j)^T \phi(x_i) = \sum_{j \in V_{i-1}} y_j k(x_j, x_i). \)
- **Update** \( V_i \leftarrow V_{i-1} \cup \{i\} \) if margin violation \( \hat{y}_i y_i \leq 0 \), else \( V_i \leftarrow V_{i-1} \).

Kernelized dual SVM. Dual objective

\[
\max_{\alpha \in [0,C]^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j).
\]
Kernelized perceptron vs Kernelized SVM

Kernelized Perceptron:

- **Predict** \( \hat{y}_i := \sum_{j \in V_{i-1}} y_j \phi(x_j)^T \phi(x_i) = \sum_{j \in V_{i-1}} y_j k(x_j, x_i) \).
- **Update** \( V_i \leftarrow V_{i-1} \cup \{i\} \) if margin violation \( \hat{y}_iy_i \leq 0 \), else \( V_i \leftarrow V_{i-1} \).

Kernelized dual SVM. Dual objective

\[
\max_{\alpha \in [0,C]^n} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j k(x_i, x_j).
\]

- **Perceptron** stops iterating when data is separated.
  SVM stops when it is separated with a large margin (unless \( C \) is small!).
Kernelized perceptron vs Kernelized SVM

Kernelized Perceptron:

- **Predict** \( \hat{y}_i := \sum_{j \in V_{i-1}} y_j \phi(x_j)^T \phi(x_i) = \sum_{j \in V_{i-1}} y_j k(x_j, x_i) \).
- **Update** \( V_i \leftarrow V_{i-1} \cup \{i\} \) if margin violation \( \hat{y}_i y_i \leq 0 \), else \( V_i \leftarrow V_{i-1} \).

Kernelized dual SVM. Dual objective

\[
\max_{\alpha \in [0, C]^n} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j k(x_i, x_j).
\]

- **Perceptron** stops iterating when data is separated.
- **SVM** stops when it is separated with a large margin (unless \( C \) is small!).
- \( V_i \) is fickle and depends on the order of data; permuting data might result in a different predictor. Meanwhile, **SVM** (primal) solution is unique.
Kernelized perceptron vs Kernelized SVM

Kernelized Perceptron:

- **Predict** $\hat{y}_i := \sum_{j \in \mathcal{V}_{i-1}} y_j \phi(x_j)^T \phi(x_i) = \sum_{j \in \mathcal{V}_{i-1}} y_j k(x_j, x_i)$.
- **Update** $\mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \cup \{i\}$ if margin violation $\hat{y}_i y_i \leq 0$, else $\mathcal{V}_i \leftarrow \mathcal{V}_{i-1}$.

Kernelized dual SVM. Dual objective

$$\max_{\alpha \in [0, C]^n} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j k(x_i, x_j).$$

- **Perceptron** stops iterating when data is separated.
  SVM stops when it is separated with a large margin (unless $C$ is small!).

- **$\mathcal{V}_i$** is fickle and depends on the order of data; permuting data might result in a different predictor. Meanwhile, SVM (primal) solution is unique.

- **$\mathcal{V}_i$** is *not* a set of support vectors.
3. Perceptron convergence and online learning
Theorem. Let examples \( ((x_i, y_i))_{i=1}^{t} \) be given, and assume \( \bar{u} \in \mathbb{R}^{d} \) with
\[
\min_{i} y_{i} x_{i}^{\top} \bar{u} = 1.
\]
Then \( |\mathcal{V}_{t}| \leq \| \bar{u} \|^{2}_{2} L^{2} \), where \( L := \max_{i} \| x_{i} \|_{2} \).
Theorem. Let examples \(((x_i, y_i))_{i=1}^{t}\) be given, and assume \(\bar{u} \in \mathbb{R}^d\) with

\[
\min_i y_i x_i^T \bar{u} = 1.
\]

Then \(|\mathcal{V}_t| \leq \|\bar{u}\|_2^2 L^2\), where \(L := \max_i \|x_i\|_2\).

Remarks.

- In other words, there are at most \(\|\bar{u}\|_2^2 L^2\) margin violations.

- Suppose we have a training set \(S := ((\tilde{x}_i, \tilde{y}_i))_{i=1}^{n}\), and run perceptron on epochs until one has no mistakes; there are at most \(1 + \|\bar{u}\|_2^2 L^2\) epochs!

- There need not be a fixed training set; we’ll get back to this!
Theorem. Let examples \( ((x_i, y_i))_{i=1}^{t} \) be given, and assume \( \bar{u} \in \mathbb{R}^d \) with
\[
\min_i y_i x_i^T \bar{u} = 1.
\]

Then \( |\mathcal{V}_t| \leq \|\bar{u}\|^2_2 L^2 \), where \( L := \max_i \|x_i\|_2 \).

Proof. Intuition: mistakes rotate \( w_i \) towards \( \bar{u} \). Therefore consider
\[
\frac{w_t^T \bar{u}}{\|w_t\| \|\bar{u}\|}.
\]
Theorem. Let examples \(((x_i, y_i))_{i=1}^{t}\) be given, and assume \(\vec{u} \in \mathbb{R}^d\) with
\[
\min_i y_i x_i^T \vec{u} = 1.
\]
Then \(|\mathcal{V}_t| \leq \|\vec{u}\|^2_2 L^2\), where \(L := \max_i \|x_i\|_2\).

Proof. Intuition: mistakes rotate \(w_i\) towards \(\vec{u}\). Therefore consider
\[
\frac{w_t^T \vec{u}}{\|w_t\| \|\vec{u}\|}.
\]

- **Numerator:** \(\vec{u}^T w_t \geq |\mathcal{V}_t|\).
Perceptron convergence theorem (Novikoff, ’62)

**Theorem.** Let examples \(((x_i, y_i))_{i=1}^t\) be given, and assume \(\bar{u} \in \mathbb{R}^d\) with

\[
\min_i y_i x_i^T \bar{u} = 1.
\]

Then \(|\mathcal{V}_t| \leq \|\bar{u}\|_2^2 L^2\), where \(L := \max_i \|x_i\|_2\).

**Proof.** Intuition: mistakes rotate \(w_i\) towards \(\bar{u}\). Therefore consider

\[
\frac{w_t^T \bar{u}}{\|w_t\| \|\bar{u}\|}.
\]

▶ **Numerator:** \(\bar{u}^T w_t \geq |\mathcal{V}_t|\).

\[
\bar{u}^T w_t = \bar{u}^T \sum_{i \in \mathcal{V}_t} x_i y_i = \sum_{i \in \mathcal{V}_t} \bar{u}^T x_i y_i \geq \sum_{i \in \mathcal{V}_t} 1 = |\mathcal{V}_t|.
\]
Theorem. Let examples \( ((x_i, y_i))_{i=1}^t \) be given, and assume \( \tilde{u} \in \mathbb{R}^d \) with

\[
\min_i y_i x_i^T \tilde{u} = 1.
\]

Then \( |\mathcal{V}_t| \leq \|\tilde{u}\|_2^2 L^2 \), where \( L := \max_i \|x_i\|_2 \).

Proof. Intuition: mistakes rotate \( w_i \) towards \( \tilde{u} \). Therefore consider

\[
\frac{w_t^T \tilde{u}}{\|w_t\| \|\tilde{u}\|}.
\]

▶ Numerator: \( \tilde{u}^T w_t \geq |\mathcal{V}_t| \).
**Theorem.** Let examples \(((x_i, y_i))_{i=1}^t\) be given, and assume \(\bar{u} \in \mathbb{R}^d\) with
\[
\min_i y_i x_i^T \bar{u} = 1.
\]
Then \(|\mathcal{V}_t| \leq \|\bar{u}\|_2^2 L^2\), where \(L := \max_i \|x_i\|_2\).

**Proof.** Intuition: mistakes rotate \(w_i\) towards \(\bar{u}\). Therefore consider
\[
\frac{w_t^T \bar{u}}{\|w_t\| \|\bar{u}\|}.
\]

- **Numerator:** \(\bar{u}^T w_t \geq |\mathcal{V}_t|\).
- **Denominator:** \(\|w_t\| \leq L \sqrt{|\mathcal{V}_t|}\).
Theorem. Let examples \(((x_i, y_i))_{i=1}^t\) be given, and assume \(\bar{u} \in \mathbb{R}^d\) with
\[
\min_i y_i x_i^T \bar{u} = 1.
\]
Then \(|V_t| \leq \|\bar{u}\|_2^2 L^2\), where \(L := \max_i \|x_i\|_2\).

Proof. Intuition: mistakes rotate \(w_i\) towards \(\bar{u}\). Therefore consider
\[
\frac{w_t^T \bar{u}}{\|w_t\| \|\bar{u}\|}.
\]

- **Numerator:** \(\bar{u}^T \bar{w} \geq |V_t|\).
- **Denominator:** \(\|w_t\| \leq L \sqrt{|V_t|}\).

\[
\|w_i\|^2 = \|w_{i-1} + x_i y_i 1[w_{i-1}^T x_i y_i \leq 0]\|^2 \\
= \|w_{i-1}\|^2 + 2 w_{i-1}^T x_i y_i 1[w_{i-1}^T x_i y_i \leq 0] + \|x_i y_i 1[w_{i-1}^T x_i y_i \leq 0]\|^2 \\
\leq \|w_{i-1}\|^2 + 0 + L^2 1[w_{i-1}^T x_i y_i \leq 0],
\]

which by induction implies \(\|w_t\|^2 \leq L^2 \sum_{i \leq t} 1[w_{i-1}^T x_i y_i \leq 0] = L^2 |V_t|\).
**Theorem.** Let examples $((x_i, y_i))_{i=1}^{t}$ be given, and assume $\bar{u} \in \mathbb{R}^d$ with

$$\min_i y_i x_i^T \bar{u} = 1.$$ 

Then $|\mathcal{V}_t| \leq \|\bar{u}\|_2^2 L^2$, where $L := \max_i \|x_i\|_2$.

**Proof.** Intuition: mistakes rotate $w_t$ towards $\bar{u}$. Therefore consider

$$\frac{w_t^T \bar{u}}{\|w_t\| \|\bar{u}\|}.$$ 

- **Numerator:** $\bar{u}^T w_t \geq |\mathcal{V}_t|$.
- **Denominator:** $\|w_t\| \leq L \sqrt{|\mathcal{V}_t|}$. 
**Theorem.** Let examples \(((x_i, y_i))_{i=1}^t\) be given, and assume \(\bar{u} \in \mathbb{R}^d\) with 

\[
\min_i y_i x_i^T \bar{u} = 1.
\]

Then \(|V_t| \leq \|\bar{u}\|_2^2 L^2\), where \(L := \max_i \|x_i\|_2\).

**Proof.** Intuition: mistakes rotate \(w_i\) towards \(\bar{u}\). Therefore consider 

\[
\frac{w_t^T \bar{u}}{\|w_t\| \|\bar{u}\|}.
\]

- **Numerator:** \(\bar{u}^T w_t \geq |V_t|\).
- **Denominator:** \(\|w_t\| \leq L \sqrt{|V_t|}\).

By Cauchy-Schwarz,

\[
1 \geq \frac{w_t^T \bar{u}}{\|w_t\| \|\bar{u}\|} \geq \frac{|V_t|}{L \|\bar{u}\| \sqrt{|V_t|}} = \frac{\sqrt{|V_t|}}{L \|\bar{u}\|},
\]

which implies \(|V_t| \leq L^2 \|\bar{u}\|^2\). \(\square\)
Theorem. Let examples $((x_i, y_i))_{i=1}^t$ be given, and assume $\bar{u} \in \mathbb{R}^d$ with

$$\min_{(x,y) \in S} yx^T \bar{u} = 1.$$ 

Then $|\mathcal{V}_t| \leq \|\bar{u}\|_2^2 L^2$, where $L := \max_i \|x_i\|_2$.

Remarks.

- There need not be a fixed training set; we’ll get back to this!
Theorem. Let examples \(((x_i, y_i))_{i=1}^t\) be given, and assume \(\bar{u} \in \mathbb{R}^d\) with
\[
\min_{(x,y) \in S} yx^\top \bar{u} = 1.
\]
Then \(|\mathcal{V}_t| \leq \|\bar{u}\|_2^2 L^2\), where \(L := \max_i \|x_i\|_2\).

Remarks.

▷ There need not be a fixed training set; we’ll get back to this!

▷ \(((x_i, y_i))_{i=1}^t\) may be chosen with knowledge of the algorithm; they can be adversarial inputs, aimed to break the algorithm!
Theorem. Let examples \(((x_i, y_i))_{i=1}^t\) be given, and assume \(\tilde{u} \in \mathbb{R}^d\) with
\[
\min_{(x, y) \in S} yx^\top \tilde{u} = 1.
\]
Then \(|\mathcal{V}_t| \leq \|\tilde{u}\|_2^2 L^2\), where \(L := \max_i \|x_i\|_2\).

Remarks.

▶ There need not be a fixed training set; we’ll get back to this!

▶ \(((x_i, y_i))_{i=1}^t\) may be chosen with knowledge of the algorithm; they can be adversarial inputs, aimed to break the algorithm!

▶ The guarantee is: for any sequence of points (with a valid \(\tilde{u}\)), the number of mistakes is at most \(L^2 \|\tilde{u}\|_2^2\)!
**Theorem.** Let examples $((x_i, y_i))_{i=1}^t$ be given, and assume $\bar{u} \in \mathbb{R}^d$ with

$$\min_{(x, y) \in S} y x^\top \bar{u} = 1.$$ 

Then $|V_t| \leq \|\bar{u}\|_2^2 L^2$, where $L := \max_i \|x_i\|_2$.

**Remarks.**

- There *need not be a fixed training set*; we’ll get back to this!
- $((x_i, y_i))_{i=1}^t$ may be chosen with knowledge of the algorithm; they can be adversarial inputs, aimed to break the algorithm!
- The guarantee is: for *any* sequence of points (with a valid $\bar{u}$), the number of mistakes is at most $L^2 \|\bar{u}\|_2^2$!
- This is the online/adversarial setting: the algorithm must be able to handle any inputs satisfying some constraint!
Theorem. Let examples \( ((x_i, y_i))_{i=1}^t \) be given, and assume \( \bar{u} \in \mathbb{R}^d \) with

\[
\min_{(x,y) \in S} y x^\top \bar{u} = 1.
\]

Then \( |\mathcal{V}_t| \leq \|\bar{u}\|_2^2 L^2 \), where \( L := \max_i \|x_i\|_2 \).

Remarks.

- There need not be a fixed training set; we’ll get back to this!
- \( ((x_i, y_i))_{i=1}^t \) may be chosen with knowledge of the algorithm; they can be adversarial inputs, aimed to break the algorithm!
- The guarantee is: for any sequence of points (with a valid \( \bar{u} \)), the number of mistakes is at most \( L^2 \|\bar{u}\|_2^2 \)!
- This is the online/adversarial setting: the algorithm must be able to handle any inputs satisfying some constraint!
- This topic has a huge literature; we’ll not cover it here. Consider though: are SPAM emails IID, or adversarial?
Example: OCR digits

- Binary classification problem: distinguish “9” from other digits.
- # training examples: 60000 (about 6000 are from class “9”).
- Test error rates using (variant of) Kernelized Perceptron.

<table>
<thead>
<tr>
<th></th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>0.037</td>
</tr>
<tr>
<td>degree 2 poly</td>
<td>0.009</td>
</tr>
<tr>
<td>degree 4 poly</td>
<td>0.006</td>
</tr>
</tbody>
</table>

Remark: why binary classification?
4. Classification objectives
(Regularized) ERM for binary classification

ERM setup: given a class of predictors $F$, minimize (regularized) empirical risk:

$$\min_{f \in F} \hat{R}(f) + P(f),$$

where

- $\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i f(x_i))$ is the empirical risk, with $\ell : \mathbb{R} \to \mathbb{R}$ a loss function;

- $P(f)$ is a regularizer/penalty;
  e.g., if $f(x) = w^T x$, one choice is $\frac{\lambda}{2} \|w\|^2$.
  (We’ll discuss this more in the upcoming statistical learning theory lectures.)
(Regularized) ERM for binary classification

ERM setup: given a class of predictors $\mathcal{F}$, minimize (regularized) empirical risk:

$$\min_{f \in \mathcal{F}} \hat{R}(f) + P(f),$$

where

- $\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i f(x_i))$ is the empirical risk, with $\ell : \mathbb{R} \to \mathbb{R}$ a loss function;

- $P(f)$ is a regularizer/penalty;
  e.g., if $f(x) = w^T x$, one choice is $\frac{1}{2} \|w\|^2$.
  (We’ll discuss this more in the upcoming statistical learning theory lectures.)

How to choose $\ell$?

- Our real goal: minimize $\ell_{0/1}(y\hat{y}) = 1[y\hat{y} \leq 0]$.

- One option: $\ell(y\hat{y}) \geq 1[y\hat{y} \leq 0]$;
  then minimizing $\hat{R}$ also minimizes misclassifications.

- There are other more complicated conditions;
  see e.g., logistic regression lectures,
  also “Statistical behavior and consistency of classification methods based
  on convex risk minimization” (Zhang, 2004).
  Overall, logistic or least squares suffices for most cases.
Zero-one loss and some surrogate losses

Some *surrogate losses* that upper-bound $\ell_{0/1}$:

<table>
<thead>
<tr>
<th>Loss</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>hinge</td>
<td>$\ell_{\text{hinge}}(z) = [1 - z]_+$</td>
</tr>
<tr>
<td>squared</td>
<td>$\ell_{\text{sq}}(z) = (1 - z)^2$</td>
</tr>
<tr>
<td>logistic’</td>
<td>$\ell_{\text{logistic}}(z) = \ln(1 + e^{-z})/\ln(2)$</td>
</tr>
<tr>
<td>exponential</td>
<td>$\ell_{\text{exp}}(z) = e^{-z}$</td>
</tr>
</tbody>
</table>
Zero-one loss and some surrogate losses

Some surrogate losses that upper-bound $\ell_{0/1}$:

<table>
<thead>
<tr>
<th>Loss</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>hinge</td>
<td>$\ell_{\text{hinge}}(z) = [1 - z]_+$</td>
</tr>
<tr>
<td>squared</td>
<td>$\ell_{\text{sq}}(z) = (1 - z)^2$</td>
</tr>
<tr>
<td>logistic'</td>
<td>$\ell_{\text{logistic}}(z) = \ln(1 + e^{-z})/\ln(2)$</td>
</tr>
<tr>
<td>exponential</td>
<td>$\ell_{\text{exp}}(z) = e^{-z}$</td>
</tr>
</tbody>
</table>

Note: when $y \in \{\pm 1\}$ and $\hat{y} \in \mathbb{R}$,

$$
\ell_{\text{sq}}(y\hat{y}) = (1 - y\hat{y})^2 = (y - \hat{y})^2.
$$

Also, logistic has been rescaled.
Some surrogate losses that upper-bound $\ell_{0/1}$:

<table>
<thead>
<tr>
<th>Loss</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>hinge</td>
<td>$\ell_{\text{hinge}}(z) = [1 - z]_+$</td>
</tr>
<tr>
<td>squared</td>
<td>$\ell_{\text{sq}}(z) = (1 - z)^2$</td>
</tr>
<tr>
<td>logistic'</td>
<td>$\ell_{\text{logistic}}(z) = \ln(1 + e^{-z})/\ln(2)$</td>
</tr>
<tr>
<td>exponential</td>
<td>$\ell_{\text{exp}}(z) = e^{-z}$</td>
</tr>
</tbody>
</table>

Note: when $y \in \{\pm 1\}$ and $\hat{y} \in \mathbb{R}$,

$$\ell_{\text{sq}}(y\hat{y}) = (1 - y\hat{y})^2 = (y - \hat{y})^2.$$ 
Also, logistic has been rescaled.

(How about Perceptron? It uses $\ell(z) = \max\{0, -z\}$ . . . )
Zero-one loss and some surrogate losses

Some surrogate losses that upper-bound $\ell_{0/1}$:

<table>
<thead>
<tr>
<th>Loss</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>hinge</td>
<td>$\ell_{\text{hinge}}(z) = [1 - z]_+$</td>
</tr>
<tr>
<td>squared</td>
<td>$\ell_{\text{sq}}(z) = (1 - z)^2$</td>
</tr>
<tr>
<td>logistic</td>
<td>$\ell_{\text{logistic}}(z) = \ln(1 + e^{-z})/\ln(2)$</td>
</tr>
<tr>
<td>exponential</td>
<td>$\ell_{\exp}(z) = e^{-z}$</td>
</tr>
</tbody>
</table>

Note: when $y \in \{\pm 1\}$ and $\hat{y} \in \mathbb{R}$,

$$\ell_{\text{sq}}(yy) = (1 - yy)^2 = (y - \hat{y})^2.$$  
Also, logistic has been rescaled.

(How about Perceptron? It uses $\ell(z) = \max\{0, -z\}$ . . . )

Question: Is zero-one $z \mapsto \mathbb{1}[z \leq 0]$ what we always want?
Confusion table (for binary classification, i.e., $\mathbb{Y} = \{0, 1\}$):

<table>
<thead>
<tr>
<th></th>
<th>$y = 1$</th>
<th>$y = -1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} \geq 0$</td>
<td># true pos.</td>
<td># false pos.</td>
</tr>
<tr>
<td>$\hat{y} &lt; 0$</td>
<td># false neg.</td>
<td># true neg.</td>
</tr>
</tbody>
</table>

Note: can also form $k \times k$ multiclass confusion matrix.
**Confusion table** (for binary classification, i.e., $\mathcal{Y} = \{0, 1\}$):

<table>
<thead>
<tr>
<th>$\hat{y}$</th>
<th>$y = 1$</th>
<th>$y = -1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} \geq 0$</td>
<td># true pos.</td>
<td># false pos.</td>
</tr>
<tr>
<td>$\hat{y} &lt; 0$</td>
<td># false neg.</td>
<td># true neg.</td>
</tr>
</tbody>
</table>

Can use numbers in table to estimate

- $\mathbb{P}(\hat{Y} \geq 0 \mid Y = 1)$ (**true positive rate**, a.k.a. **sensitivity/recall**),
- $\mathbb{P}(\hat{Y} \geq 0 \mid Y = -1)$ (**false positive rate**, a.k.a. $1 - \text{specificity}$),
- $\mathbb{P}(\hat{Y} < 0 \mid Y = -1)$ (**true negative rate / specificity**), $1 - \text{FPR}$,
- $\mathbb{P}(\hat{Y} < 0 \mid Y = 1)$ (**false negative rate**), $1 - \text{TPR}$,
- $\mathbb{P}(Y = 1 \mid \hat{Y} \geq 0)$ (**precision**).

... assuming numbers in table are from test data, not training data!
Confusion table (for binary classification, i.e., $\mathcal{Y} = \{0, 1\}$):

<table>
<thead>
<tr>
<th>$\hat{y}$</th>
<th>$y = 1$</th>
<th>$y = -1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} \geq 0$</td>
<td># true pos.</td>
<td># false pos.</td>
</tr>
<tr>
<td>$\hat{y} &lt; 0$</td>
<td># false neg.</td>
<td># true neg.</td>
</tr>
</tbody>
</table>

Can use numbers in table to estimate:

- $\mathbb{P}(\hat{Y} \geq 0 \mid Y = 1)$ (*true positive rate*, a.k.a. *sensitivity/recall*),
- $\mathbb{P}(\hat{Y} \geq 0 \mid Y = -1)$ (*false positive rate*, a.k.a. $1 - \text{specificity}$),
- $\mathbb{P}(\hat{Y} < 0 \mid Y = -1)$ (*true negative rate / specificity*), $1 - \text{FPR}$,
- $\mathbb{P}(\hat{Y} < 0 \mid Y = 1)$ (*false negative rate*, $1 - \text{TPR}$),
- $\mathbb{P}(Y = 1 \mid \hat{Y} \geq 0)$ (*precision*).

... assuming numbers in table are from test data, not training data!

**Note:** can also form $k \times k$ multic和平 confusion matrix.
Confusion table — changing threshold

**Confusion table** (for binary classification, i.e., $\mathcal{Y} = \{0, 1\}$):

<table>
<thead>
<tr>
<th></th>
<th>$y = 1$</th>
<th>$y = -1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} \geq 0$</td>
<td># true pos.</td>
<td># false pos.</td>
</tr>
<tr>
<td>$\hat{y} &lt; 0$</td>
<td># false neg.</td>
<td># true neg.</td>
</tr>
</tbody>
</table>

Question: can we easily increase the TPR?

Answer: yes, by prediction with $f(x) - \theta$ where $\theta > 0$. 

12 / 18
**Confusion table** (for binary classification, i.e., $\mathcal{Y} = \{0, 1\}$):

<table>
<thead>
<tr>
<th></th>
<th>$y = 1$</th>
<th>$y = -1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{y} \geq 0$</td>
<td># true pos.</td>
<td># false pos.</td>
</tr>
<tr>
<td>$\hat{y} &lt; 0$</td>
<td># false neg.</td>
<td># true neg.</td>
</tr>
</tbody>
</table>

**Question:** can we easily increase the TPR?
Confusion table — changing threshold

Confusion table (for binary classification, i.e., \( \mathcal{Y} = \{0, 1\} \)):

<table>
<thead>
<tr>
<th></th>
<th>( y = 1 )</th>
<th>( y = -1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{y} \geq 0 )</td>
<td># true pos.</td>
<td># false pos.</td>
</tr>
<tr>
<td>( \hat{y} &lt; 0 )</td>
<td># false neg.</td>
<td># true neg.</td>
</tr>
</tbody>
</table>

Question: can we easily increase the TPR?

Answer: yes, by prediction with \( f(x) - \theta \) where \( \theta > 0 \).
Receiver operating characteristic (ROC) curve

Predict with $f_\theta(x) := f(x) - \theta$; this increases TPR, but also FPR.
Receiver operating characteristic (ROC) curve

Predict with $f_\theta(x) := f(x) - \theta$; this increases TPR, but also FPR.

Each $\hat{f}_\theta$ has its own (FPR, TPR).
Receiver operating characteristic (ROC) curve

Predict with $f_{\theta}(x) := f(x) - \theta$; this increases TPR, but also FPR.
Each $\hat{f}_\theta$ has its own (FPR, TPR).

What is the achievable curve? (This is called the ROC curve.)
Receiver operating characteristic (ROC) curve

Predict with $f_\theta(x) := f(x) - \theta$; this increases TPR, but also FPR.

Each $\hat{f}_\theta$ has its own (FPR, TPR).

What is the achievable curve? (This is called the ROC curve.)

AUC (area under (ROC) curve) is another popular classification performance measure.
Predict with $f_\theta(x) := f(x) - \theta$; this increases TPR, but also FPR.

Each $\hat{f}_\theta$ has its own (FPR, TPR).

What is the achievable curve? (This is called the ROC curve.)

AUC (area under (ROC) curve) is another popular classification performance measure.

Yet another is F1: $\left(\frac{\text{precision}^{-1} + \text{recall}^{-1}}{2}\right)^{-1}$. 
Receiver operating characteristic (ROC) curve

Predict with $f_{\theta}(x) := f(x) - \theta$; this increases TPR, but also FPR.

Each $\hat{f}_{\theta}$ has its own (FPR, TPR).

What is the achievable curve? (This is called the ROC curve.)

AUC (area under (ROC) curve) is another popular classification performance measure.

Yet another is $F1$: $\left( \frac{\text{precision}^{-1} + \text{recall}^{-1}}{2} \right)^{-1}$.

Note: might be able to do better by retraining for specific FPR, rather than having one predictor and adjusting $\theta$. Can we optimize this?
Often have **different costs for different kinds of mistakes**. For $c \in [0, 1]$:

<table>
<thead>
<tr>
<th></th>
<th>$\hat{y} = -1$</th>
<th>$\hat{y} = +1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = -1$</td>
<td>0</td>
<td>$c$</td>
</tr>
<tr>
<td>$y = +1$</td>
<td>$1 - c$</td>
<td>0</td>
</tr>
</tbody>
</table>
Cost-sensitive classification

Often have **different costs for different kinds of mistakes**. For \( c \in [0, 1] \):

<table>
<thead>
<tr>
<th>( y )</th>
<th>( \hat{y} = -1 )</th>
<th>( \hat{y} = +1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = -1 )</td>
<td>0</td>
<td>( c )</td>
</tr>
<tr>
<td>( y = +1 )</td>
<td>( 1 - c )</td>
<td>0</td>
</tr>
</tbody>
</table>

(Why can we restrict attention to \( c \in [0, 1] \)?)
Often have **different costs for different kinds of mistakes**. For \( c \in [0, 1] \):

<table>
<thead>
<tr>
<th></th>
<th>( \hat{y} = -1 )</th>
<th>( \hat{y} = +1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = -1 )</td>
<td>0</td>
<td>( c )</td>
</tr>
<tr>
<td>( y = +1 )</td>
<td>( 1 - c )</td>
<td>0</td>
</tr>
</tbody>
</table>

(Why can we restrict attention to \( c \in [0, 1] \)?)

**Cost-sensitive zero-one loss**:

\[
\ell^{(c)}_{0/1}(y, p) = (\mathbb{1}\{y = 1\} \cdot (1 - c) + \mathbb{1}\{y = -1\} \cdot c) \cdot \ell_{0/1}(yp).
\]
Cost-sensitive classification

Often have **different costs for different kinds of mistakes**. For $c \in [0, 1]$:

<table>
<thead>
<tr>
<th></th>
<th>$\hat{y} = -1$</th>
<th>$\hat{y} = +1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = -1$</td>
<td>0</td>
<td>$c$</td>
</tr>
<tr>
<td>$y = +1$</td>
<td>$1 - c$</td>
<td>0</td>
</tr>
</tbody>
</table>

(Why can we restrict attention to $c \in [0, 1]$?)

**Cost-sensitive $\ell$-loss** (for loss $\ell : \mathbb{R} \rightarrow \mathbb{R}$):

$$
\ell^{(c)}(y, p) = (\mathbb{1}\{y = 1\} \cdot (1 - c) + \mathbb{1}\{y = -1\} \cdot c) \cdot \ell(yp).
$$
Cost-sensitive classification

Often have **different costs for different kinds of mistakes.** For \( c \in [0, 1] \):

<table>
<thead>
<tr>
<th></th>
<th>( \hat{y} = -1 )</th>
<th>( \hat{y} = +1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = -1 )</td>
<td>0</td>
<td>( c )</td>
</tr>
<tr>
<td>( y = +1 )</td>
<td>( 1 - c )</td>
<td>0</td>
</tr>
</tbody>
</table>

(Why can we restrict attention to \( c \in [0, 1] \)?)

**Cost-sensitive \( \ell \)-loss** (for loss \( \ell : \mathbb{R} \to \mathbb{R} \)):

\[
\ell^{(c)}(y, p) = \left( \mathbf{1}\{y = 1\} \cdot (1 - c) + \mathbf{1}\{y = -1\} \cdot c \right) \cdot \ell(yp).
\]

**Fact:** if \( \ell \) is convex, then so is \( \ell^{(c)}(y, \cdot) \) for each \( y \in \{\pm 1\} \).
Often have **different costs for different kinds of mistakes**. For \( c \in [0, 1] \):

<table>
<thead>
<tr>
<th>( y )</th>
<th>( \hat{y} = -1 )</th>
<th>( \hat{y} = +1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y = -1 )</td>
<td>0</td>
<td>( c )</td>
</tr>
<tr>
<td>( y = +1 )</td>
<td>( 1 - c )</td>
<td>0</td>
</tr>
</tbody>
</table>

(Why can we restrict attention to \( c \in [0, 1] \)?)

**Cost-sensitive \( \ell \)-loss** (for loss \( \ell : \mathbb{R} \to \mathbb{R} \)):

\[
\ell^{(c)}(y, p) = (\mathbb{1}\{y = 1\} \cdot (1 - c) + \mathbb{1}\{y = -1\} \cdot c) \cdot \ell(yp).
\]

**Fact**: if \( \ell \) is convex, then so is \( \ell^{(c)}(y, \cdot) \) for each \( y \in \{\pm 1\} \).

**Cost-sensitive (empirical) \( \ell \)-risk** of predictor \( h : \mathcal{X} \to \mathbb{R} \):

\[
\mathcal{R}^{(c)}(h) := \mathbb{E} \left[ \ell^{(c)}(Y, h(X)) \right] \quad \text{Our actual objective.}
\]

\[
\widehat{\mathcal{R}}^{(c)}(h) := \frac{1}{n} \sum_{i=1}^{n} \ell^{(c)}(Y_i, h(X_i)) \quad \text{What we can try to minimize.}
\]
Minimizer of cost-sensitive risk

What is the optimal classifier for cost-sensitive (zero-one loss) risk?

Let $\eta(x) := P(Y = 1 | X = x)$ for $x \in X$.

Conditionally on $X = x$, the minimizer of conditional cost-sensitive risk $\hat{y} \mapsto E[\ell(c)(Y, \hat{y}) | X = x]$ is $\hat{y} = \theta(x) \cdot (1 - c) \cdot 1\{\hat{y} = -1\} + (1 - \theta(x)) \cdot c \cdot 1\{\hat{y} = +1\}$.

Therefore, the classifier based on scoring function $h(x) := \theta(x) - c$, $x \in X$ has the smallest cost-sensitive risk $R(c)$.

Minimizer of cost-sensitive risk

What is the optimal classifier for cost-sensitive (zero-one loss) risk?
Let \( \eta(x) := \mathbb{P}(Y = 1 \mid X = x) \) for \( x \in \mathcal{X} \).
What is the optimal classifier for cost-sensitive (zero-one loss) risk? Let \( \eta(x) := \mathbb{P}(Y = 1 \mid X = x) \) for \( x \in \mathcal{X} \).

- Conditional on \( X = x \), the minimizer of conditional cost-sensitive risk

\[
\hat{y} \mapsto \mathbb{E} \left[ \ell^{(c)}(Y, \hat{y}) \mid X = x \right]
\]

\[
= \eta(x) \cdot (1 - c) \cdot 1\{\hat{y} = -1\} + (1 - \eta(x)) \cdot c \cdot 1\{\hat{y} = +1\}
\]

is

\[
\hat{y} := \begin{cases} 
+1 & \text{if } \eta(x) \cdot (1 - c) > (1 - \eta(x)) \cdot c, \\
-1 & \text{otherwise.}
\end{cases}
\]
What is the optimal classifier for cost-sensitive (zero-one loss) risk?

Let \( \eta(x) := \mathbb{P}(Y = 1 \mid X = x) \) for \( x \in \mathcal{X} \).

Conditional on \( X = x \), the minimizer of conditional cost-sensitive risk

\[
\hat{y} \mapsto \mathbb{E}\left[ \ell(c)(Y, \hat{y}) \mid X = x \right] = \eta(x) \cdot (1 - c) \cdot 1\{\hat{y} = -1\} + (1 - \eta(x)) \cdot c \cdot 1\{\hat{y} = +1\}
\]

is

\[
\hat{y} := \begin{cases} 
+1 & \text{if } \eta(x) > c, \\
-1 & \text{otherwise}.
\end{cases}
\]
Minimizer of cost-sensitive risk

What is the optimal classifier for cost-sensitive (zero-one loss) risk?
Let $\eta(x) := \mathbb{P}(Y = 1 \mid X = x)$ for $x \in \mathcal{X}$.

- Conditional on $X = x$, the minimizer of conditional cost-sensitive risk

$$
\hat{y} \mapsto \mathbb{E} \left[ \ell_c(Y, \hat{y}) \mid X = x \right]
= \eta(x) \cdot (1 - c) \cdot 1\{\hat{y} = -1\} + (1 - \eta(x)) \cdot c \cdot 1\{\hat{y} = +1\}
$$

is

$$
\hat{y} := \begin{cases} 
+1 & \text{if } \eta(x) > c, \\
-1 & \text{otherwise.}
\end{cases}
$$

- Therefore, the classifier based on scoring function

$$
h(x) := \eta(x) - c, \quad x \in \mathcal{X}
$$

has the smallest cost-sensitive risk $\mathcal{R}^{(c)}$. 
Example: tune $c$ for balanced error rate

Suppose you are care about *Balanced Error Rate (BER)*:

$$\text{BER} := \frac{1}{2} \cdot \text{False Negative Rate} + \frac{1}{2} \cdot \text{False Positive Rate}.$$ 

Which cost-sensitive risk should you (try to) minimize?
Example: tune $c$ for balanced error rate

Suppose you are care about *Balanced Error Rate (BER)*:

$$BER := \frac{1}{2} \cdot \text{False Negative Rate} + \frac{1}{2} \cdot \text{False Positive Rate}.$$  

Which cost-sensitive risk should you (try to) minimize?

$$2 \cdot BER = \underbrace{\Pr(h(X) \leq 0 \mid Y = +1)}_{\text{FNR}} + \underbrace{\Pr(h(X) > 0 \mid Y = -1)}_{\text{FPR}}$$
Example: tune $c$ for balanced error rate

Suppose you are care about *Balanced Error Rate (BER)*:

$$
\text{BER} := \frac{1}{2} \cdot \text{False Negative Rate} + \frac{1}{2} \cdot \text{False Positive Rate}.
$$

Which cost-sensitive risk should you (try to) minimize?

$$
2 \cdot \text{BER} = \mathbb{P}(h(X) \leq 0 \mid Y = +1) + \mathbb{P}(h(X) > 0 \mid Y = -1)
\begin{array}{ll}
\text{FNR} & \text{FPR} \\
\end{array}
$$

$$
= \frac{1}{\pi} \mathbb{P}(h(X) \leq 0 \land Y = +1) + \frac{1}{1 - \pi} \mathbb{P}(h(X) > 0 \land Y = -1)
$$

where $\pi := \mathbb{P}(Y = +1)$. 

Example: tune $c$ for balanced error rate

Suppose you are care about *Balanced Error Rate (BER)*:

$$BER := \frac{1}{2} \cdot \text{False Negative Rate} + \frac{1}{2} \cdot \text{False Positive Rate}.$$  

Which cost-sensitive risk should you (try to) minimize?

$$2 \cdot BER = \mathbb{P}(h(X) \leq 0 \mid Y = +1) + \mathbb{P}(h(X) > 0 \mid Y = -1)$$

$$= \frac{1}{\pi} \mathbb{P}(h(X) \leq 0 \land Y = +1) + \frac{1}{1 - \pi} \mathbb{P}(h(X) > 0 \land Y = -1)$$

where $\pi := \mathbb{P}(Y = +1)$. So use $\mathcal{R}^{(c)}$ with

$$c := \frac{1}{\frac{1}{1 - \pi} + \frac{1}{\pi}} = \pi = \mathbb{P}(Y = +1)$$

(which you can estimate).
Recall that kernel perceptron experiments were binary.
Multiclass classification

Recall that kernel perceptron experiments were binary.

▶ Some methods are naturally multiclass: $k$-nn, decision trees, logistic regression, deep networks, . . .
Multiclass classification

Recall that kernel perceptron experiments were binary.

- Some methods are *naturally multiclass*: $k$-nn, decision trees, logistic regression, deep networks, . . .
- Others, like kernel SVM and kernel perceptron, baked binary labels into their formulation.

One-vs-all reduction from multiclass to binary:

- Create $k$ binary datasets: $S_j := \{(x_i, 2 \cdot 1[y_i = j] - 1)\}_{i=1}^n$.
- Train $k$ binary classifiers ($f_j$)$_{j=1}^k$, where $f_j$ is trained on $S_j$.
- To predict, perform $x \mapsto \text{arg max}_j f_j(x)$.

When does it work well?

If $(f_1(x), ..., f_k(x)) \in \mathbb{R}^k$ is a conditional probability vector ($\Pr[Y = 1 | X = x], ..., \Pr[Y = k | X = x]$).

(Enforced on softmax deep networks. . .)

Why can it fail?

Nothing forces the predictors to coordinate; can have $f_1(x) = \infty$ for all $x$.

People have spent tons of effort on this (check out all the literature on multiclass SVM), but these days. . .
Multiclass classification

Recall that kernel perceptron experiments were binary.

- Some methods are naturally multiclass: $k$-nn, decision trees, logistic regression, deep networks, ... 
- Others, like kernel SVM and kernel perceptron, baked binary labels into their formulation.

One-vs-all reduction from multiclass to binary:

- Create $k$ binary datasets: $S_j := ((x_i, 2 \cdot 1[y_i = j] - 1))_{i=1}^n$.
- Train $k$ binary classifiers $(f_j)_{j=1}^k$, where $f_j$ is trained on $S_j$.
- To predict, perform $x \mapsto \arg \max_j f_j(x)$. 

When does it work well?

If $(f_1(x), ..., f_k(x)) \in \mathbb{R}^k$ is a conditional probability vector $(\Pr[Y = 1 | X = x], ..., \Pr[Y = k | X = x])$.

(Enforced on softmax deep networks...)

Why can it fail?

Nothing forces the predictors to coordinate; can have $f_1(x) = \infty$ for all $x$.

People have spent tons of effort on this (check out all the literature on multiclass SVM), but these days...
Multiclass classification

Recall that kernel perceptron experiments were binary.

- Some methods are naturally multiclass: \(k\)-nn, decision trees, logistic regression, deep networks, \ldots
- Others, like kernel SVM and kernel perceptron, baked binary labels into their formulation.

One-vs-all reduction from multiclass to binary:

- Create \(k\) binary datasets: 
  \[ S_j := ((x_i, 2 \cdot 1[y_i = j] - 1))_{i=1}^{n}. \]
- Train \(k\) binary classifiers \((f_j)_{j=1}^{k}\), where \(f_j\) is trained on \(S_j\).
- To predict, perform \(x \mapsto \arg \max_j f_j(x)\).

When does it work well? If \((f_1(x), \ldots, f_k(x)) \in \mathbb{R}^k\) is a conditional probability vector \((\Pr[Y = 1|X = x], \ldots, \Pr[Y = k|X = x])\).
Multiclass classification

Recall that kernel perceptron experiments were binary.

- Some methods are naturally multiclass: \(k\)-nn, decision trees, logistic regression, deep networks, . . .
- Others, like kernel SVM and kernel perceptron, baked binary labels into their formulation.

One-vs-all reduction from multiclass to binary:

- Create \(k\) binary datasets: \(S_j := (x_i, 2 \cdot 1[y_i = j] - 1))_{i=1}^n\).
- Train \(k\) binary classifiers \((f_j)_{j=1}^k\), where \(f_j\) is trained on \(S_j\).
- To predict, perform \(x \mapsto \arg \max_j f_j(x)\).

When does it work well? If \((f_1(x), \ldots, f_k(x)) \in \mathbb{R}^k\) is a conditional probability vector \((\Pr[Y = 1|X = x], \ldots, \Pr[Y = k|X = x])\). (Enforced on softmax deep networks . . .)
Multiclass classification

Recall that kernel perceptron experiments were binary.

- Some methods are naturally multiclass: \( k \)-nn, decision trees, logistic regression, deep networks, . . .
- Others, like kernel SVM and kernel perceptron, baked binary labels into their formulation.

**One-vs-all** reduction from multiclass to binary:

- Create \( k \) binary datasets: \( S_j := (((x_i, 2 \cdot 1[y_i = j] - 1))_{i=1}^n \).
- Train \( k \) binary classifiers \((f_j)_{j=1}^k\), where \( f_j \) is trained on \( S_j \).
- To predict, perform \( x \mapsto \arg \max_j f_j(x) \).

When does it work well? If \((f_1(x), \ldots, f_k(x)) \in \mathbb{R}^k\) is a conditional probability vector \((\Pr[Y = 1|X = x], \ldots, \Pr[Y = k|X = x])\). (Enforced on softmax deep networks . . .)

Why can it fail? Nothing forces the predictors to coordinate; can have \( f_1(x) = \infty \) for all \( x \).
Multiclass classification

Recall that kernel perceptron experiments were binary.

- Some methods are naturally multiclass: \(k\)-nn, decision trees, logistic regression, deep networks, \ldots
- Others, like kernel SVM and kernel perceptron, baked binary labels into their formulation.

One-vs-all reduction from multiclass to binary:

- Create \(k\) binary datasets: \(S_j := ((x_i, 2 \cdot 1[y_i = j] - 1))_{i=1}^n\).
- Train \(k\) binary classifiers \((f_j)_{j=1}^k\), where \(f_j\) is trained on \(S_j\).
- To predict, perform \(x \mapsto \arg\max_j f_j(x)\).

When does it work well? If \((f_1(x), \ldots, f_k(x)) \in \mathbb{R}^k\) is a conditional probability vector \((\Pr[Y = 1|X = x], \ldots, \Pr[Y = k|X = x])\). (Enforced on softmax deep networks\ldots)

Why can it fail? Nothing forces the predictors to coordinate; can have \(f_1(x) = \infty\) for all \(x\). People have spent tons of effort on this (check out all the literature on multiclass SVM), but these days\ldots
5. Summary
Summary

- Perceptron algorithm.
- Perceptron guarantee; online/adversarial learning.
- Kernel perceptron.
- Different losses.
- One-against-all, multiclass.