Unbalanced Data (Binary Classification)

Balanced data: \( p(Y = +1) \approx p(Y = -1) \approx \frac{1}{2} \).

Examples where the fraction of positive examples is tiny: fraud detection, web page relevance
Unbalanced Data (Binary Classification)

Balanced data: \( p(Y = +1) \approx p(Y = -1) \approx \frac{1}{2} \).

Examples where the fraction of positive examples is tiny: fraud detection, web page relevance

Some solutions:
Unbalanced Data (Binary Classification)

Balanced data: \( p(Y = +1) \approx p(Y = -1) \approx \frac{1}{2} \).

Examples where the fraction of positive examples is tiny: fraud detection, web page relevance

Some solutions:

1. Throw out negative examples until you achieve balance.
Unbalanced Data (Binary Classification)

Balanced data: \( p(Y = +1) \approx p(Y = -1) \approx \frac{1}{2} \).

Examples where the fraction of positive examples is tiny: fraud detection, web page relevance

Some solutions:

1. Throw out negative examples until you achieve balance.
2. Down-weight negative examples until you achieve balance.
Unbalanced Data (Binary Classification)

Balanced data: \( p(Y = +1) \approx p(Y = -1) \approx \frac{1}{2} \).

Examples where the fraction of positive examples is tiny: fraud detection, web page relevance

Some solutions:

1. Throw out negative examples until you achieve balance.
2. Down-weight negative examples until you achieve balance. For example,

\[
L^{(\text{new})}(x, y, \text{parameters}) \leftarrow \alpha[y=+1] \cdot L^{(\text{old})}(x, y, \text{parameters})
\]

A similar effect can be achieved in SGD by sampling non-uniformly; assign \( \frac{1}{2N_+} \) to positive examples and \( \frac{1}{2N_-} \) to negative examples.
Unbalanced Data (Binary Classification)

Balanced data: \( p(Y = +1) \approx p(Y = -1) \approx \frac{1}{2} \).

Examples where the fraction of positive examples is tiny: fraud detection, web page relevance

Some solutions:

1. Throw out negative examples until you achieve balance.
2. Down-weight negative examples until you achieve balance.
3. Modification to the hinge loss:

\[
L_n^{(\text{hinge})}(w, b) = \max\{0, \underbrace{\text{cost}(y_n) - y_n \cdot (w \cdot x_n + b)}_{\text{formerly } 1}\}
\]

\[
\text{cost}(y_n) = \begin{cases} 
\alpha & \text{if } y_n = -1 \text{ (false positive)} \\
\beta & \text{if } y_n = +1 \text{ (false negative)} 
\end{cases}
\]
Multiclass Classification

Suppose you have a set of classes, $\mathcal{Y}$, such that $|\mathcal{Y}| > 2$. 

1. See A5 for generalizations of familiar loss functions.

2. One-versus-all training: train $|\mathcal{Y}|$ binary classifiers, letting each $y \in \mathcal{Y}$ take a turn as the positive class. Let $a(y)$ be the activation function for the classifier where 

$$
\begin{align*}
    y &\rightarrow +1, \\
    \mathcal{Y}\setminus\{y\} &\rightarrow -1
\end{align*}
$$

Then define the classifier $f: \mathcal{X} \rightarrow \mathcal{Y}$ as:

$$f(x) = \text{argmax}_{y \in \mathcal{Y}} a(y)(x)$$

Theorem: error rate is at most $(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$, where $\bar{\epsilon}$ is the average error rate among the binary classifiers.

3. All-versus-all (“tournament”): build $\binom{|\mathcal{Y}|}{2}$ classifiers, pairing every $y, y' \in \mathcal{Y}$.

Theorem: error rate is at most $2(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$.

4. Tree-structured tournament. Theorem: error rate is at most $\lceil \log_2 |\mathcal{Y}| \rceil \cdot \bar{\epsilon}$. 

Multiclass Classification

Suppose you have a set of classes, \( \mathcal{Y} \), such that \(|\mathcal{Y}| > 2\).

1. See A5 for generalizations of familiar loss functions.
Multiclass Classification

Suppose you have a set of classes, $\mathcal{Y}$, such that $|\mathcal{Y}| > 2$.

1. See A5 for generalizations of familiar loss functions.

2. One-versus-all training: train $|\mathcal{Y}|$ binary classifiers, letting each $y \in \mathcal{Y}$ take a turn as the positive class. Let $a^{(y)}$ be the activation function for the classifier where $\{y \rightarrow +1, \mathcal{Y} \setminus \{y\} \rightarrow -1\}$. Then define the classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$ as:

   $$f(x) = \arg\max_{y \in \mathcal{Y}} a^{(y)}(x)$$

   Theorem: error rate is at most $(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$, where $\bar{\epsilon}$ is the average error rate among the binary classifiers.

3. All-versus-all ("tournament"): build $\binom{|\mathcal{Y}|}{2}$ classifiers, pairing every $y, y' \in \mathcal{Y}$. Theorem: error rate is at most $2(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$.

4. Tree-structured tournament. Theorem: error rate is at most $\lceil \log_2 |\mathcal{Y}| \rceil \cdot \bar{\epsilon}$. 


Multiclass Classification

Suppose you have a set of classes, $\mathcal{Y}$, such that $|\mathcal{Y}| > 2$.

1. See A5 for generalizations of familiar loss functions.

2. One-versus-all training: train $|\mathcal{Y}|$ binary classifiers, letting each $y \in \mathcal{Y}$ take a turn as the positive class. Let $a^{(y)}$ be the activation function for the classifier where \(\{y \rightarrow +1, \mathcal{Y} \setminus \{y\} \rightarrow -1\}\). Then define the classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$ as:

$$f(x) = \arg\max_{y \in \mathcal{Y}} a^{(y)}(x)$$

Theorem: error rate is at most $(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$, where $\bar{\epsilon}$ is the average error rate among the binary classifiers. One bad classifier can ruin $f$; in particular, watch out for the more rare labels, and be sure to tune hyperparameters separately.

3. All-versus-all ("tournament"): build $\left(\frac{|\mathcal{Y}|}{2}\right)$ classifiers, pairing every $y, y' \in \mathcal{Y}$.

Theorem: error rate is at most $2(\frac{|\mathcal{Y}|}{2} - 1) \cdot \bar{\epsilon}$.

4. Tree-structured tournament. Theorem: error rate is at most $\lceil \log_2 |\mathcal{Y}| \rceil \cdot \bar{\epsilon}$. 
Multiclass Classification

Suppose you have a set of classes, $\mathcal{Y}$, such that $|\mathcal{Y}| > 2$.

1. See A5 for generalizations of familiar loss functions.

2. One-versus-all training: train $|\mathcal{Y}|$ binary classifiers, letting each $y \in \mathcal{Y}$ take a turn as the positive class. Let $a^{(y)}$ be the activation function for the classifier where $\{y \rightarrow +1, \mathcal{Y} \setminus \{y\} \rightarrow -1\}$. Then define the classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$ as:

$$f(x) = \arg\max_{y \in \mathcal{Y}} a^{(y)}(x)$$

Theorem: error rate is at most $(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$, where $\bar{\epsilon}$ is the average error rate among the binary classifiers.

3. All-versus-all (“tournament”): build $\binom{|\mathcal{Y}|}{2}$ classifiers, pairing every $y, y' \in \mathcal{Y}$.

Theorem: error rate is at most $2(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$. 


Multiclass Classification

Suppose you have a set of classes, \( \mathcal{Y} \), such that \(|\mathcal{Y}| > 2\).

1. See A5 for generalizations of familiar loss functions.

2. One-versus-all training: train \(|\mathcal{Y}|\) binary classifiers, letting each \( y \in \mathcal{Y} \) take a turn as the positive class. Let \( a^{(y)} \) be the activation function for the classifier where \( \{y \rightarrow +1, \mathcal{Y} \setminus \{y\} \rightarrow -1\} \). Then define the classifier \( f: \mathcal{X} \rightarrow \mathcal{Y} \) as:

\[
f(x) = \arg\max_{y \in \mathcal{Y}} a^{(y)}(x)
\]

Theorem: error rate is at most \((|\mathcal{Y}| - 1) \cdot \bar{\epsilon}\), where \( \bar{\epsilon} \) is the average error rate among the binary classifiers.

3. All-versus-all ("tournament"): build \( \binom{|\mathcal{Y}|}{2} \) classifiers, pairing every \( y, y' \in \mathcal{Y} \).

Theorem: error rate is at most \(2(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}\).

4. Tree-structured tournament. Theorem: error rate is at most \( \lceil \log_2 |\mathcal{Y}| \rceil \cdot \bar{\epsilon} \).
Multiclass Classification

Suppose you have a set of classes, $\mathcal{Y}$, such that $|\mathcal{Y}| > 2$.

1. See A5 for generalizations of familiar loss functions.

2. One-versus-all training: train $|\mathcal{Y}|$ binary classifiers, letting each $y \in \mathcal{Y}$ take a turn as the positive class. Let $a^{(y)}$ be the activation function for the classifier where $\{y \rightarrow +1, \mathcal{Y} \setminus \{y\} \rightarrow -1\}$. Then define the classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$ as:

$$f(x) = \arg\max_{y \in \mathcal{Y}} a^{(y)}(x)$$

Theorem: error rate is at most $(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$, where $\bar{\epsilon}$ is the average error rate among the binary classifiers.

3. All-versus-all (“tournament”): build $\binom{|\mathcal{Y}|}{2}$ classifiers, pairing every $y, y' \in \mathcal{Y}$.

Theorem: error rate is at most $2(|\mathcal{Y}| - 1) \cdot \bar{\epsilon}$.

4. Tree-structured tournament. Theorem: error rate is at most $\lceil \log_2 |\mathcal{Y}| \rceil \cdot \bar{\epsilon}$.

Challenge: you must choose the tree.
Tree-Structured Tournament for Multiclass Classification

\[ f_1: \{y_1, y_2, y_3, y_4\} \text{ vs. } \{y_5, y_6, y_7, y_8\} \]

\[ f_2: \{y_1, y_2\} \text{ vs. } \{y_3, y_4\} \]

\[ f_3: \{y_5, y_6\} \text{ vs. } \{y_7, y_8\} \]

\[ f_4: y_1 \text{ vs. } y_2 \]

\[ f_5: y_3 \text{ vs. } y_4 \]

\[ f_6: y_5 \text{ vs. } y_6 \]

\[ f_7: y_7 \text{ vs. } y_8 \]
Most common setup: $x_n = (q_n, d)$, where $q_n$ is a query and $d$ is a (fixed, universal) set of documents $\{d_1, \ldots, d_M\}$. Output $y_n$ is a ranking of $d$. 
Most common setup: \( x_n = \langle q_n, d \rangle \), where \( q_n \) is a query and \( d \) is a (fixed, universal) set of documents \( \{d_1, \ldots, d_M\} \). Output \( y_n \) is a ranking of \( d \).

Pairwise encoding: let \( x_{n,i,j} \) be the features encoding the comparison of \( d_i \) with \( d_j \), under query \( q_n \).
Most common setup: $x_n = \langle q_n, d \rangle$, where $q_n$ is a query and $d$ is a (fixed, universal) set of documents $\{d_1, \ldots, d_M\}$. Output $y_n$ is a ranking of $d$.

Pairwise encoding: let $x_{n,i,j}$ be the features encoding the comparison of $d_i$ with $d_j$, under query $q_n$.

Output: $y_{n,i,j}$ is $+1$ if $d_i$ is more relevant to $q_n$ than $d_j$; $-1$ otherwise.
Ranking

Most common setup: \( x_n = \langle q_n, d \rangle \), where \( q_n \) is a query and \( d \) is a (fixed, universal) set of documents \( \{d_1, \ldots, d_M\} \). Output \( y_n \) is a ranking of \( d \).

Pairwise encoding: let \( x_{n,i,j} \) be the features encoding the comparison of \( d_i \) with \( d_j \), under query \( q_n \).

Output: \( y_{n,i,j} \) is +1 if \( d_i \) is more relevant to \( q_n \) than \( d_j \); −1 otherwise.

Training on the binary problem \( \langle (x_{n,i,j}, y_{n,i,j}) \rangle_{n \in \{1, \ldots, N\}; i,j \in \{1, \ldots, M\}} \) makes sense when the ranking is meant to separate relevant \( d_i \) from irrelevant \( d_i \), known as “bipartite” ranking.
Nuanced Ranking Problems

Intuitively, we want a scoring function, specific to query $q$, that is highest for the *most* relevant documents.
Intuitively, we want a scoring function, specific to query $q$, that is highest for the most relevant documents.

Let $\sigma : \{1, \ldots, M\} \to \mathbb{R}$ denote a document-scoring function; the true scoring function for training example $n$ is $\sigma_n$, and $\hat{\sigma}_n$ is what we’ve estimated.
Intuitively, we want a scoring function, specific to query $q$, that is highest for the most relevant documents.

Let $\sigma : \{1, \ldots, M\} \rightarrow \mathbb{R}$ denote a document-scoring function; the true scoring function for training example $n$ is $\sigma_n$, and $\hat{\sigma}_n$ is what we’ve estimated.

Let $\omega(i, j)$ be the nonnegative cost of putting something in position $j$ when it belongs in position $i$. 
Intuitively, we want a scoring function, specific to query $q$, that is highest for the most relevant documents.

Let $\sigma : \{1, \ldots, M\} \to \mathbb{R}$ denote a document-scoring function; the true scoring function for training example $n$ is $\sigma_n$, and $\hat{\sigma}_n$ is what we’ve estimated.

Let $\omega(i, j)$ be the nonnegative cost of putting something in position $j$ when it belongs in position $i$.

One example:

$$\omega(i, j) = \begin{cases} 1 & \text{if } \min(i, j) \leq 10 \text{ and } i \neq j \\ 0 & \text{otherwise} \end{cases}$$

(More in the book.)
Intuitively, we want a scoring function, specific to query \( q \), that is highest for the most relevant documents.

Let \( \sigma : \{1, \ldots, M\} \to \mathbb{R} \) denote a document-scoring function; the true scoring function for training example \( n \) is \( \sigma_n \), and \( \hat{\sigma}_n \) is what we’ve estimated.

Let \( \omega(i, j) \) be the nonnegative cost of putting something in position \( j \) when it belongs in position \( i \).

Loss:

\[
E_{(q, \sigma) \sim D} \left[ \sum_{i, j: i \neq j} \left[ \sigma(i) < \sigma(j) \right] \cdot \left[ \hat{\sigma}(i) < \hat{\sigma}(j) \right] \cdot \omega(i, j) \right]
\]
Nuanced Ranking Problems

Intuitively, we want a scoring function, specific to query $q$, that is highest for the most relevant documents.

Let $\sigma : \{1, \ldots, M\} \rightarrow \mathbb{R}$ denote a document-scoring function; the true scoring function for training example $n$ is $\sigma_n$, and $\hat{\sigma}_n$ is what we’ve estimated.

Let $\omega(i, j)$ be the nonnegative cost of putting something in position $j$ when it belongs in position $i$.

Loss:

$$\mathbb{E}_{(q, \sigma) \sim \mathcal{D}} \left[ \sum_{i, j : i \neq j} \left[ \sigma(i) < \sigma(j) \right] \cdot \left[ \hat{\sigma}(i) < \hat{\sigma}(j) \right] \cdot \omega(i, j) \right]$$

Deriving a learning algorithm is left as an exercise. (See the book for an example.)