COMS 4771 Lecture 12

1. Introduction to learning theory
2. Cross validation
Introduction to learning theory
Basic setting

- Training data $S$ is an iid sample from some fixed but unknown probability distribution $P$ over space of labeled examples $\mathcal{X} \times \mathcal{Y}$.
Basic setting

- Training data $S$ is an iid sample from some fixed but unknown probability distribution $P$ over space of labeled examples $\mathcal{X} \times \mathcal{Y}$.

- A learning algorithm takes $S$ as input and returns a predictor $\hat{f}: \mathcal{X} \rightarrow \mathcal{Y}$. 

---

Statistical learning

Benchmark: (true) prediction error $\text{err}(\hat{f})$.

(Recall, $\text{err}(f) = \Pr(f(X) \neq Y)$, where $(X, Y)$ has distribution $P$.)

---

In this setting, can any learning algorithm always provide a non-trivial guarantee on the error of the predictor it returns?

No: some assumptions/conditions are required.

("No free lunch" theorem)
Basic setting

- **Training data** $S$ is an iid sample from some fixed but unknown probability distribution $P$ over space of labeled examples $\mathcal{X} \times \mathcal{Y}$.
- A **learning algorithm** takes $S$ as input and returns a predictor $\hat{f}: \mathcal{X} \rightarrow \mathcal{Y}$.
- **Benchmark**: (true) prediction error $\text{err}(\hat{f})$.

(Recall, $\text{err}(f) = \Pr(f(X) \neq Y)$, where $(X, Y)$ has distribution $P$.)
Basic setting

- Training data $S$ is an iid sample from some fixed but unknown probability distribution $P$ over space of labeled examples $\mathcal{X} \times \mathcal{Y}$.
- A learning algorithm takes $S$ as input and returns a predictor $\hat{f}: \mathcal{X} \rightarrow \mathcal{Y}$.
- Benchmark: (true) prediction error $\text{err}(\hat{f})$.

(Recall, $\text{err}(f) = \Pr(f(X) \neq Y)$, where $(X, Y)$ has distribution $P$.)
Basic setting

- Training data $S$ is an iid sample from some fixed but unknown probability distribution $P$ over space of labeled examples $\mathcal{X} \times \mathcal{Y}$.

- A learning algorithm takes $S$ as input and returns a predictor $\hat{f}: \mathcal{X} \rightarrow \mathcal{Y}$.

- Benchmark: (true) prediction error $\text{err}(\hat{f})$.
  (Recall, $\text{err}(f) = \Pr(f(X) \neq Y)$, where $(X, Y)$ has distribution $P$.)

In this setting, can any learning algorithm always provide a non-trivial guarantee on the error of the predictor it returns?
Basic setting

- **Training data** $S$ is an iid sample from some fixed but unknown probability distribution $P$ over space of labeled examples $\mathcal{X} \times \mathcal{Y}$.
- A **learning algorithm** takes $S$ as input and returns a **predictor** $\hat{f}: \mathcal{X} \to \mathcal{Y}$.
- **Benchmark**: (true) prediction error $\text{err}(\hat{f})$.
  (Recall, $\text{err}(f) = \Pr(f(X) \neq Y)$, where $(X, Y)$ has distribution $P$.)

In this setting, can any **learning algorithm** always provide a non-trivial guarantee on the error of the **predictor** it returns?

**No**: some assumptions/conditions are required.
("No free lunch" theorem)
Consider binary classification with $Y = \{0, 1\}$.

**Realizability assumption**: Assume that, for a given class $\mathcal{F}$ of functions from $\mathcal{X} \to \{0, 1\}$, there exists $f^* \in \mathcal{F}$ such that $\text{err}(f^*) = 0$. This implies that $f^*$ is the Bayes classifier.

Examples of function classes $\mathcal{F}$:

- Rectangles in $\mathcal{X} = \mathbb{R}^2$:
  $$f((a, b), (c, d))(x) = 1 \{a \leq x_1 \leq b \text{ and } c \leq x_2 \leq d\}.$$

- Monotone conjunctions in $\mathcal{X} = \{0, 1\}^d$:
  $$f_V(x) = 1 \{x_i = 1 \text{ for all } i \in V\}.$$

- Linear classifiers in $\mathcal{X} = \mathbb{R}^d$. 

**Realizable setting** is essentially the setup of **PAC Learning**, a theoretical model of learning introduced by L. Valiant (1984).
Subsection: Realizability

Consider binary classification with \( \mathcal{Y} = \{0, 1\} \).

**Realizability assumption**: Assume that, for a given class \( \mathcal{F} \) of functions from \( \mathcal{X} \to \{0, 1\} \), there exists \( f^* \in \mathcal{F} \) such that \( \text{err}(f^*) = 0 \).

(This implies that \( f^* \) is the Bayes classifier.)
Assumption: realizability

Consider binary classification with $\mathcal{Y} = \{0, 1\}$.

Realizability assumption: Assume that, for a given class $\mathcal{F}$ of functions from $\mathcal{X} \to \{0, 1\}$, there exists $f^* \in \mathcal{F}$ such that $\text{err}(f^*) = 0$.

(This implies that $f^*$ is the Bayes classifier.)

Examples of function classes $\mathcal{F}$:

- Rectangles in $\mathcal{X} = \mathbb{R}^2$:
  \[
  f((a,b),(c,d))(\bm{x}) = \mathbb{1}\{a \leq x_1 \leq b \text{ and } c \leq x_2 \leq d\}.
  \]

- Monotone conjunctions in $\mathcal{X} = \{0, 1\}^d$:
  \[
  f_V(\bm{x}) = \mathbb{1}\{x_i = 1 \text{ for all } i \in V\}.
  \]

- Linear classifiers in $\mathcal{X} = \mathbb{R}^d$. 

Consider binary classification with $\mathcal{Y} = \{0, 1\}$.

**Realizability assumption**: Assume that, for a given class $\mathcal{F}$ of functions from $\mathcal{X} \rightarrow \{0, 1\}$, there exists $f^* \in \mathcal{F}$ such that $\text{err}(f^*) = 0$.

(This implies that $f^*$ is the Bayes classifier.)

**Examples of function classes $\mathcal{F}$**:

- Rectangles in $\mathcal{X} = \mathbb{R}^2$:
  
  $$f_{((a, b), (c, d))}(\mathbf{x}) = 1\{a \leq x_1 \leq b \text{ and } c \leq x_2 \leq d\}.$$ 

- Monotone conjunctions in $\mathcal{X} = \{0, 1\}^d$:
  
  $$f_V(\mathbf{x}) = 1\{x_i = 1 \text{ for all } i \in V\}.$$ 

- Linear classifiers in $\mathcal{X} = \mathbb{R}^d$.

Realizable setting is essentially the setup of **PAC Learning**, a theoretical model of learning introduced by L. Valiant (1984).
What is a sensible learning algorithm for the realizable setting?

Pick a consistent classifier (a special case of ERM):
Given training data $S$, return any $\hat{f} \in F$ such that $\text{err}(\hat{f}, S) = 0$.

Always possible under realizability assumption!
Example: rectangles in $\mathbb{R}^2$.
Example: monotone conjunctions in $\{0, 1\}^d$.
Return monotone conjunction $f$ with $V := [d] \backslash \{\bigcup (x, x+1) \in S \{i \in [d] : x_i = 0\}\}$.

$\triangleright$ Start with $V := [d]$.
$\triangleright$ For each positive example $(x, x+1) \in S$, remove all $i \in [d]$ s.t. $x_i = 0$.

Example: linear classifiers in $\mathbb{R}^d \rightarrow$ linear programming, Perceptron, or SVM.
Learning in the realizable setting

What is a sensible learning algorithm for the realizable setting?

**Pick a consistent classifier** (a special case of ERM):
Given training data $S$, return any $\hat{f} \in \mathcal{F}$ such that $\text{err}(\hat{f}, S) = 0$.
Always possible under realizability assumption!
What is a sensible learning algorithm for the realizable setting?

**Pick a consistent classifier** (a special case of ERM):
Given training data $S$, return any $\hat{f} \in \mathcal{F}$ such that $\text{err}(\hat{f}, S) = 0$.
Always possible under realizability assumption!

**Example**: rectangles in $\mathbb{R}^2$.

**Example**: monotone conjunctions in $\{0, 1\}^d$.

Return monotone conjunction $f$ with $V := \{d\} \setminus \{\bigcup (x, x+1) \in S \{i \in [d] : x_i = 0\}\}$.

**Example**: linear classifiers in $\mathbb{R}^d \rightarrow \text{linear programming, Perceptron, or SVM.}$
Learning in the realizable setting

What is a sensible learning algorithm for the realizable setting?

**Pick a consistent classifier** (a special case of ERM):
Given training data $S$, return any $\hat{f} \in F$ such that $\text{err}(\hat{f}, S) = 0$.
Always possible under realizability assumption!

**Example**: rectangles in $\mathbb{R}^2$.
(Blue dots = positive examples.)
What is a sensible learning algorithm for the realizable setting?

**Pick a consistent classifier** (a special case of ERM):
Given training data $S$, return any $\hat{f} \in \mathcal{F}$ such that $\text{err}(\hat{f}, S) = 0$.
Always possible under realizability assumption!

**Example**: rectangles in $\mathbb{R}^2$.

(Blue dots = positive examples.)
What is a sensible learning algorithm for the realizable setting?

**Pick a consistent classifier** (a special case of ERM):
Given training data $S$, return any $\hat{f} \in F$ such that $\text{err}(\hat{f}, S) = 0$.
Always possible under realizability assumption!

**Example**: rectangles in $\mathbb{R}^2$.

(Blue dots = positive examples.)

**Example**: monotone conjunctions in $\{0, 1\}^d$.  

(start with $V := [d]$)

For each positive example $(x, +1) \in S$, remove all $i \in [d]$ s.t. $x_i = 0$. 

**Example**: linear classifiers in $\mathbb{R}^d \rightarrow$ linear programming, Perceptron, or SVM.
What is a sensible learning algorithm for the realizable setting?

Pick a consistent classifier (a special case of ERM): Given training data $S$, return any $\hat{f} \in F$ such that $\text{err}(\hat{f}, S) = 0$.

Always possible under realizability assumption!

Example: rectangles in $\mathbb{R}^2$.
(Blue dots = positive examples.)

Example: monotone conjunctions in $\{0, 1\}^d$.

Return monotone conjunction $f_V$ with

$$V := [d] \setminus \left( \bigcup_{(x, +1) \in S} \{i \in [d] : x_i = 0\} \right).$$
What is a sensible learning algorithm for the realizable setting?

**Pick a consistent classifier** (a special case of ERM):
Given training data $S$, return any $\hat{f} \in \mathcal{F}$ such that $\text{err}(\hat{f}, S) = 0$.
Always possible under realizability assumption!

**Example**: rectangles in $\mathbb{R}^2$.
(Blue dots = positive examples.)

**Example**: monotone conjunctions in $\{0, 1\}^d$.
Return monotone conjunction $f_V$ with

$$V := [d] \setminus \left( \bigcup_{(\mathbf{x}, +1) \in S} \{i \in [d] : x_i = 0\} \right).$$

- Start with $V := [d]$.
- For each positive example $(\mathbf{x}, +1) \in S$, remove all $i \in [d]$ from $V$ s.t. $x_i = 0$. 

What is a sensible learning algorithm for the realizable setting?

**Pick a consistent classifier** (a special case of ERM):
Given training data \( S \), return any \( \hat{f} \in \mathcal{F} \) such that \( \text{err}(\hat{f}, S) = 0 \).
Always possible under realizability assumption!

**Example**:
- rectangles in \( \mathbb{R}^2 \).
  - (Blue dots = positive examples.)

**Example**:
- monotone conjunctions in \( \{0, 1\}^d \).
  - Return monotone conjunction \( f_V \) with
    \[
    V := [d] \setminus \left( \bigcup_{(\mathbf{x}, +1) \in S} \{ i \in [d] : x_i = 0 \} \right).
    \]
  - Start with \( V := [d] \).
  - For each positive example \( (\mathbf{x}, +1) \in S \),
    remove all \( i \in [d] \) from \( V \) s.t. \( x_i = 0 \).

**Example**:
- linear classifiers in \( \mathbb{R}^d \) → linear programming, Perceptron, or SVM.
Is the “Consistent Classifier Algorithm” any good in the realizable setting?

▶ Would like to show that typically, the error of the returned classifier \( \hat{f} \in \mathcal{F} \) goes to zero as the number of training data increases.
Consistent Classifier Algorithm

Is the “Consistent Classifier Algorithm” any good in the realizable setting?

▶ Would like to show that typically, the error of the returned classifier
\( \hat{f} \in \mathcal{F} \) goes to zero as the number of training data increases.

▶ Formally, want for any \( \delta \in (0, 1) \),

\[
\Pr \left[ \text{err}(\hat{f}) \leq \varepsilon(|S|) \right] \geq 1 - \delta
\]

for some \( \varepsilon(n) \) satisfying

\[
\lim_{n \to \infty} \varepsilon(n) = 0.
\]

\textbf{Note:} \( \varepsilon(n) \) may also depend on \( \delta, \mathcal{F}, f^*, \ldots \)
Is the "Consistent Classifier Algorithm" any good in the realizable setting?

- Would like to show that typically, the error of the returned classifier \( \hat{f} \in \mathcal{F} \) goes to zero as the number of training data increases.

- Formally, want for any \( \delta \in (0, 1) \),

\[
\Pr \left[ \text{err}(\hat{f}) \leq \varepsilon(|S|) \right] \geq 1 - \delta
\]

for some \( \varepsilon(n) \) satisfying

\[
\lim_{n \to \infty} \varepsilon(n) = 0.
\]

**Note:** \( \varepsilon(n) \) may also depend on \( \delta, \mathcal{F}, f^*, \ldots \)

**Note 2:** Could also ask for

\[
\mathbb{E} \left[ \text{err}(\hat{f}) \right] \leq \varepsilon(|S|) \to 0
\]

(i.e., statistical consistency).
Analysis of “Consistent Classifier Algorithm”

We know that for any \( f : \mathcal{X} \to \{0, 1\} \),

\[
\Pr \left[ \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(1/\delta)}{|S|}} + \frac{2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.
\]

(Upper limit of confidence interval for a coin bias based on Chernoff bounds.)
Consistent Classifier Algorithm

Analysis of “Consistent Classifier Algorithm”

We know that for any \( f : \mathcal{X} \to \{0, 1\} \),

\[
\Pr \left[ \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(1/\delta)}{|S|}} + \frac{2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.
\]

(Upper limit of confidence interval for a coin bias based on Chernoff bounds.)

**Question:**
Does this apply to classifier \( \hat{f} \in \mathcal{F} \) returned by the algorithm?
Analysis of “Consistent Classifier Algorithm”

We know that for any $f: \mathcal{X} \to \{0, 1\}$,

$$\Pr \left[ \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2\text{err}(f, S)\ln(1/\delta)}{|S|}} + \frac{2\ln(1/\delta)}{|S|} \right] \geq 1 - \delta.$$

(Upper limit of confidence interval for a coin bias based on Chernoff bounds.)

**Question:**
Does this apply to classifier $\hat{f} \in \mathcal{F}$ returned by the algorithm?

**Generally no:** $\hat{f}$ is not fixed; it’s chosen based on (random) training data $S$. 
Detour: overfitting

Consider $\mathcal{F} = $ union of up to 9 rectangles in $\mathbb{R}^2$.

Data distribution $P$ over $\mathbb{R}^2 \times \{0, 1\}$.
(blue = positive mass)
Consider $\mathcal{F} =$ union of up to 9 rectangles in $\mathbb{R}^2$.

Particular rectangle function $f_1 \in \mathcal{F}$. 
Consider $\mathcal{F} = \text{union of up to 9 rectangles in } \mathbb{R}^2$.

Random sample $S$ from $P$. 
Consider $\mathcal{F} =$ union of up to 9 rectangles in $\mathbb{R}^2$.

Particular rectangle function $f_1 \in \mathcal{F}$ and $S$. 
$\text{err}(f_1, S) \approx \text{err}(f_1)$
Consider $\mathcal{F} = \text{union of up to 9 rectangles in } \mathbb{R}^2$. 

Union of rectangles function $f_2 \in \mathcal{F}$. 
Consider $\mathcal{F} =$ union of up to 9 rectangles in $\mathbb{R}^2$.

Random sample $S'$ from $P$. 
Consider $\mathcal{F} =$ union of up to 9 rectangles in $\mathbb{R}^2$.

Union of rectangles function $f_2 \in \mathcal{F}$ on $S'$. 
$\text{err}(f_2, S') \approx \text{err}(f_2)$
Consider $\mathcal{F} = \text{union of up to 9 rectangles in } \mathbb{R}^2$. Back to first sample $S$ from $P$. 
Consider $\mathcal{F} = \text{union of up to 9 rectangles in } \mathbb{R}^2$.

Rectangle function $\hat{f}_{3,S} \in \mathcal{F}$ on $S$.

$0 = \text{err}(\hat{f}_{3,S}, S) < \text{err}(\hat{f}_{3,S})$
Consider $\mathcal{F} = \text{union of up to 9 rectangles in } \mathbb{R}^2$.

Rectangle function $\hat{f}_{3,s} \in \mathcal{F}$ on $S'$.

$0 < \text{err}(\hat{f}_{3,s}, S') \approx \text{err}(\hat{f}_{3,s})$
Consider $\mathcal{F} = \text{union of up to 9 rectangles in } \mathbb{R}^2$.

Union of rectangles $\hat{f}_{4,S} \in \mathcal{F}$ on $S$.

$0 = \text{err}(\hat{f}_{4,S}, S) \ll \text{err}(\hat{f}_{4}, S)$
Consider $\mathcal{F} = \text{union of up to 9 rectangles in } \mathbb{R}^2$.

Union of rectangles $\hat{f}_{4,S} \in \mathcal{F}$ on $S'$.

$0 \ll \text{err}(\hat{f}_{4,S}, S') \approx \text{err}(\hat{f}_{4,S})$
Analysis of “Consistent Classifier Algorithm”

Moral: for a classifier chosen using the training data, training error is not an unbiased estimate of true error.
Analysis of “Consistent Classifier Algorithm”

**Moral:** for a classifier *chosen using the training data*, training error is not an unbiased estimate of true error.

For all $f: \mathcal{X} \rightarrow \{0, 1\}$,

$$\Pr\left[ \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2\text{err}(f, S) \ln(1/\delta)}{|S|}} + \frac{2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.$$  

(Upper limit of confidence interval for a coin bias based on Chernoff bounds.)
Analysis of “Consistent Classifier Algorithm”

**Moral:** for a classifier chosen using the training data, training error is not an unbiased estimate of true error.

For all \( f : \mathcal{X} \to \{0, 1\} \),

\[
\Pr \left[ \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(1/\delta)}{|S|}} + \frac{2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.
\]

(Upper limit of confidence interval for a coin bias based on Chernoff bounds.)

**Overkill solution:** ensure upper confidence bounds hold for all \( f \in \mathcal{F} \) simultaneously, with probability \( \geq 1 - \delta \).
Analysis of Consistent Classifier Algorithm for finite $\mathcal{F}$

**Union bound**: for any countable sequence of events $\mathcal{E}_1, \mathcal{E}_2, \ldots$,

$$\Pr\left[\bigcup_{i \geq 1} \mathcal{E}_i\right] \leq \sum_{i \geq 1} \Pr[\mathcal{E}_i].$$
Analysis of Consistent Classifier Algorithm for finite $\mathcal{F}$

**Union bound**: for any countable sequence of events $\mathcal{E}_1, \mathcal{E}_2, \ldots,$

$$\mathbb{P} \left[ \bigcup_{i \geq 1} \mathcal{E}_i \right] \leq \sum_{i \geq 1} \mathbb{P}[\mathcal{E}_i].$$

Apply to events $\mathcal{E}_f$ for $f \in \mathcal{F}$ given by

$$\mathcal{E}_f := \left\{ \text{err}(f) > \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(1/\delta)}{|S|}} + \frac{2 \ln(1/\delta)}{|S|} \right\}.$$

(From last slide: $\mathbb{P}[^{\mathcal{E}_f}] \leq \delta$ for each $f \in \mathcal{F}$.)
Analysis of Consistent Classifier Algorithm for finite $\mathcal{F}$

**Union bound**: for any countable sequence of events $E_1, E_2, \ldots$,

$$\Pr \left[ \bigcup_{i \geq 1} E_i \right] \leq \sum_{i \geq 1} \Pr[E_i].$$

Apply to events $E_f$ for $f \in \mathcal{F}$ given by

$$E_f := \left\{ \text{err}(f) > \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(1/\delta)}{|S|}} + \frac{2 \ln(1/\delta)}{|S|} \right\}.$$

(From last slide: $\Pr[E_f] \leq \delta$ for each $f \in \mathcal{F}$.)

Therefore, $\Pr[\bigcup_{f \in \mathcal{F}} E_f] \leq |\mathcal{F}| \delta \ldots$
Analysis of Consistent Classifier Algorithm for finite $\mathcal{F}$

**Union bound:** for any countable sequence of events $\mathcal{E}_1, \mathcal{E}_2, \ldots$,

$$\Pr \left[ \bigcup_{i \geq 1} \mathcal{E}_i \right] \leq \sum_{i \geq 1} \Pr [\mathcal{E}_i].$$

Apply to events $\mathcal{E}_f$ for $f \in \mathcal{F}$ given by

$$\mathcal{E}_f := \left\{ \text{err}(f) > \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(1/\delta)}{|S|}} + \frac{2 \ln(1/\delta)}{|S|} \right\}.$$  

(From last slide: $\Pr[\mathcal{E}_f] \leq \delta$ for each $f \in \mathcal{F}$.)

Therefore, $\Pr[\bigcup_{f \in \mathcal{F}} \mathcal{E}_f] \leq |\mathcal{F}| \delta$ . . . i.e., (replacing $\delta$ with $\delta/|\mathcal{F}|$)

$$\Pr \left[ \forall f \in \mathcal{F} \cdot \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.$$
Consistent Classifier Algorithm for finite $\mathcal{F}$

From last slide:

$$
Pr \left[ \forall f \in \mathcal{F}. \ err(f) \leq err(f, S) + \sqrt{\frac{2 \ err(f, S) \ ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.
$$
From last slide:

$$\Pr \left[ \forall f \in \mathcal{F} . \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.$$ 

Since the Consistent Classifier Algorithm returns $\hat{f} \in \mathcal{F}$, we know that

$$\Pr \left[ \text{err}(\hat{f}) \leq \text{err}(\hat{f}, S) + \sqrt{\frac{2 \text{err}(\hat{f}, S) \ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.$$
**Consistent Classifier Algorithm for finite \( \mathcal{F} \)**

**From last slide:**

\[
\Pr \left[ \forall f \in \mathcal{F} \cdot \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.
\]

Since the Consistent Classifier Algorithm returns \( \hat{f} \in \mathcal{F} \), we know that

\[
\Pr \left[ \text{err}(\hat{f}) \leq \text{err}(\hat{f}, S) + \sqrt{\frac{2 \text{err}(\hat{f}, S) \ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.
\]

By definition of \( \hat{f} \), \( \text{err}(\hat{f}, S) = 0 \), and therefore

\[
\Pr \left[ \text{err}(\hat{f}) \leq \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.
\]
Consistent Classifier Algorithm for finite $\mathcal{F}$

From last slide:

$$\Pr \left[ \forall f \in \mathcal{F} \cdot \text{err}(f) \leq \text{err}(f, S) + \sqrt{\frac{2 \text{err}(f, S) \ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.$$ 

Since the Consistent Classifier Algorithm returns $\hat{f} \in \mathcal{F}$, we know that

$$\Pr \left[ \text{err}(\hat{f}) \leq \text{err}(\hat{f}, S) + \sqrt{\frac{2 \text{err}(\hat{f}, S) \ln(|\mathcal{F}|/\delta)}{|S|}} + \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.$$ 

By definition of $\hat{f}$, $\text{err}(\hat{f}, S) = 0$, and therefore

$$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2 \ln(|\mathcal{F}|/\delta)}{|S|} \right] \geq 1 - \delta.$$ 

True error of $\hat{f}$ goes to zero as $|S| \to \infty$ at $O\left(\frac{\log(|\mathcal{F}|/\delta)}{|S|}\right)$ rate. $\square$
Another interpretation:

- Suppose learning algorithm finds classifier \( \hat{f} \) with \( \text{err}(\hat{f}, S) = 0 \)
i.e., a perfect classifier on the training data!
  (This is possible under realizability assumption.)
Another interpretation:

1. Suppose learning algorithm finds classifier \( \hat{f} \) with \( \text{err}(\hat{f}, S) = 0 \)
i.e., a perfect classifier on the training data!
   (This is possible under realizability assumption.)

2. How does this perfection generalize to future examples?
Another interpretation:

- Suppose learning algorithm finds classifier $\hat{f}$ with $\text{err}(\hat{f}, S) = 0$
  i.e., a perfect classifier on the training data!
  (This is possible under realizability assumption.)

- How does this perfection generalize to future examples?

- **Theory says:** with high probability (over random training data),
  true error is not much larger than training error:

  $$\text{err}(\hat{f}) \leq O\left(\frac{\log |\mathcal{F}|}{|S|}\right).$$

  Sometimes true error is also called generalization error.
Another interpretation:

- Suppose learning algorithm finds classifier $\hat{f}$ with $\text{err}(\hat{f}, S) = 0$ i.e., a **perfect classifier on the training data**!(This is possible under realizability assumption.)

- How does this **perfection** generalize to future examples?

- **Theory says**: with high probability (over random training data), **true error** is not much larger than **training error**:

  $\text{err}(\hat{f}) \leq O\left(\frac{\log |\mathcal{F}|}{|S|}\right)$.

  Sometimes **true error** is also called **generalization error**.

- Clearly only reasonable if $\log |\mathcal{F}|$ is finite and not too large!
What about infinite function classes? (e.g., rectangles, linear classifiers)
Infinite function classes

What about infinite function classes? (e.g., rectangles, linear classifiers)

Okay as long as \# effective behaviors of \( \mathcal{F} \) w.r.t. \( S \) is relatively small.
What about infinite function classes? (e.g., rectangles, linear classifiers)

Okay as long as \# effective behaviors of $\mathcal{F}$ w.r.t. $S$ is relatively small.

Let $S := ((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)}))$. Then

$$\mathcal{F}_S := \left\{ (f(x^{(1)}), f(x^{(2)}), \ldots, f(x^{(n)})) : f \in \mathcal{F} \right\} \subseteq \{0, 1\}^{|S|}$$

(i.e., all the different ways $S$ can be labeled by functions in $\mathcal{F}$).
What about infinite function classes? (e.g., rectangles, linear classifiers)

Okay as long as the number of effective behaviors of $\mathcal{F}$ w.r.t. $S$ is relatively small.

Let $S := ((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)}))$. Then

$$\mathcal{F}|_S := \left\{(f(x^{(1)}), f(x^{(2)}), \ldots, f(x^{(n)})) : f \in \mathcal{F}\right\} \subseteq \{0, 1\}^{|S|}$$

(i.e., all the different ways $S$ can be labeled by functions in $\mathcal{F}$).

(Some of the ways to label the five points by linear classifiers—there are several more.)
Effective behaviors

What is the size of $F_{|S}$?
Effective Behaviors

What is the size of $\mathcal{F}|_{S}$?

Some possibilities:

- **Bad situation:** $|\mathcal{F}|_{S} = 2^{|S|}$ (all labelings possible).

  Function class is too rich for this data set $S$. Enough functions in $\mathcal{F}$ to perfectly explain all possible labelings (even a random labeling).
Effective behaviors

What is the size of $\mathcal{F}|_S$?

Some possibilities:

- **Bad situation**: $|\mathcal{F}|_S| = 2^{|S|}$ (all labelings possible).
  
  Function class is too rich for this data set $S$. Enough functions in $\mathcal{F}$ to perfectly explain all possible labelings (even a random labeling).

- **Good situation**: $|\mathcal{F}|_S| \leq (c|S|)^v$ for some constant $c > 0$ and non-negative integer $v$.
  
  Function class $\mathcal{F}$ is limited in capacity to assign labels to points in $S$. 

---

† VC = Vapnik-Chervonenkis (1971), same duo who proposed SVMs.
Effective behaviors

What is the size of $\mathcal{F}|_S$?

Some possibilities:

- **Bad situation**: $|\mathcal{F}|_S| = 2^{|S|}$ (all labelings possible).

  Function class is too rich for this data set $S$. Enough functions in $\mathcal{F}$ to perfectly explain all possible labelings (even a random labeling).

- **Good situation**: $|\mathcal{F}|_S| \leq (c|S|)^v$ for some constant $c > 0$ and non-negative integer $v$.

  Function class $\mathcal{F}$ is limited in capacity to assign labels to points in $S$.

We say $\mathcal{F}$ is a **VC class** if, as the number of training data $|S|$ increases, we are eventually in the Good situation (regardless of the actual points in $S$).

†VC = Vapnik-Chervonenkis (1971), same duo who proposed SVMs.
**Effective behaviors**

**Example:** arbitrary convex shapes in $\mathbb{R}^2$

For any $n$, there is a set of $n$ points in $\mathbb{R}^2$ for which convex shapes realize all possible labelings. $\rightarrow$ **Could always be in the Bad situation.**
**Effective behaviors**

**Example:** arbitrary convex shapes in $\mathbb{R}^2$

For any $n$, there is a set of $n$ points in $\mathbb{R}^2$ for which convex shapes realize all possible labelings. → **Could always be in the Bad situation.**

**Example:** linear classifiers in $\mathbb{R}^2$

There are no sets of 4 points in $\mathbb{R}^2$ where linear classifiers realize all possible labelings. → It turns out that $|\mathcal{F}|S| \leq (c|S|)^3$. 
Would be great if we could “plug-in” $|\mathcal{F}|_S$ in place of $|\mathcal{F}|$ in guarantee for Consistent Classifier Algorithm:

$$\Pr\left[\text{err}(\hat{f}) \leq \frac{2\ln(|\mathcal{F}|/\delta)}{|S|}\right] \geq 1 - \delta.$$  

(*)
Effective behaviors

Would be great if we could “plug-in” $|\mathcal{F}_S|$ in place of $|\mathcal{F}|$ in guarantee for Consistent Classifier Algorithm:

$$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2 \ln(|\mathcal{F}_S|/\delta)}{|S|} \right] \geq 1 - \delta.$$  

$\star$
Effective behaviors

Would be great if we could “plug-in” $|\mathcal{F}|_S$ in place of $|\mathcal{F}|$ in guarantee for Consistent Classifier Algorithm:

$$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2 \ln(|\mathcal{F}|_S/\delta)}{|S|} \right] \geq 1 - \delta. \quad (*)$$

- **Bad situation**: $|\mathcal{F}|_S = 2^{|S|}$ (all labelings possible). Useless in ($\star$):

  $$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2|S| \ln(2) + 2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.$$
Effective behaviors

Would be great if we could “plug-in” $|\mathcal{F}_S|$ in place of $|\mathcal{F}|$ in guarantee for Consistent Classifier Algorithm:

$$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2 \ln(|\mathcal{F}_S|/\delta)}{|S|} \right] \geq 1 - \delta. \quad (\star)$$

- **Bad situation**: $|\mathcal{F}_S| = 2^{|S|}$ (all labelings possible). Useless in (\star):

  $$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2 |S| \ln(2) + 2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.$$ 

- **Good situation**: $|\mathcal{F}_S| \leq (c|S|)^v$. Bound in (\star) becomes

  $$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2v \ln(c|S|) + 2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.$$
Effective behaviors

Would be great if we could “plug-in” $|\mathcal{F}_S|$ in place of $|\mathcal{F}|$ in guarantee for Consistent Classifier Algorithm:

$$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2 \ln(|\mathcal{F}_S|/\delta)}{|S|} \right] \geq 1 - \delta. \quad (\star)$$

- **Bad situation:** $|\mathcal{F}_S| = 2^{|S|}$ (all labelings possible). Useless in $(\star)$:

  $$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2^{|S|} \ln(2) + 2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.$$ 

- **Good situation:** $|\mathcal{F}_S| \leq (c|S|)^v$. Bound in $(\star)$ becomes

  $$\Pr \left[ \text{err}(\hat{f}) \leq \frac{2^v \ln(c|S|) + 2 \ln(1/\delta)}{|S|} \right] \geq 1 - \delta.$$ 

This straight-up “plugging-in” isn’t technically legal, but different argument implies something like $(\star)$ is true!
Recap and final remarks

- The Consistent Classifier Algorithm returns \( \hat{f} \in \mathcal{F} \) with \( \text{err}(\hat{f}) \to 0 \) as \( |S| \to \infty \) with high probability, provided that:
  - labels are realized by some \( f^* \in \mathcal{F} \);
  - \( |\mathcal{F}| \) is finite, or \( |\mathcal{F}|_{|S|} \) only grows polynomially with \( |S| \).

- Guarantees depend on complexity of function class \( \mathcal{F} \) (either cardinality or effective number of behaviors).

- Without realizability assumption, essentially same argument applies to give a different guarantee.

Using ERM:

\[
\hat{f} := \arg \min_{f \in \mathcal{F}} \text{err}(f, S),
\]

with high probability, excess error \( \text{err}(\hat{f}) - \min_{f \in \mathcal{F}} \text{err}(f) \) goes to zero as \( |S| \) increases under same complexity conditions.
The Consistent Classifier Algorithm returns $\hat{f} \in \mathcal{F}$ with $\text{err}(\hat{f}) \to 0$ as $|S| \to \infty$ with high probability, provided that:

- labels are realized by some $f^* \in \mathcal{F}$;
- $|\mathcal{F}|$ is finite, or $|\mathcal{F}|_S$ only grows polynomially with $|S|$.

Guarantees depend on complexity of function class $\mathcal{F}$ (either cardinality or effective number of behaviors).
 Recap and final remarks

- The Consistent Classifier Algorithm returns \( \hat{f} \in \mathcal{F} \) with \( \text{err}(\hat{f}) \to 0 \) as \( |S| \to \infty \) with high probability, provided that:
  - labels are realized by some \( f^* \in \mathcal{F} \);
  - \( |\mathcal{F}| \) is finite, or \( |\mathcal{F}_{|S}| \) only grows polynomially with \( |S| \).
- Guarantees depend on complexity of function class \( \mathcal{F} \)
  (either cardinality or effective number of behaviors).
- Without realizability assumption, essentially same argument applies to give a different guarantee.

Using ERM: \( \hat{f} := \arg\min_{f \in \mathcal{F}} \text{err}(f, S) \), with high probability, excess error

\[
\text{err}(\hat{f}) - \min_{f \in \mathcal{F}} \text{err}(f)
\]

goes to zero as \( |S| \) increases under same complexity conditions.
Cross validation
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, $\lambda$ 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 $\lambda$ and Gaussian kernel bandwidth $h > 0$:

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

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

Naïve approach

- Also minimize over $(\lambda, h)$ in SVM optimization problem.
- Leads to overfitting: resulting SVM classifier adapts too closely to specific properties of the training data, rather than underlying distribution.
Classifier in this example has bandwidth parameter $\sigma$ (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 $h$ 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.

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

2. Train classifier $\hat{f}_h$ on *Training* data for different values of $h$.

3. Compute *Validation* ("hold-out") error for each $\hat{f}_h$: $\text{err}(\hat{f}_h, \text{Validation})$.

4. Selection: $\hat{h} =$ value of $h$ with lowest *Validation* error.

5. Train classifier $\hat{f}$ using $\hat{h}$ with *Training* + *Validation* data.

**Model assessment:**

6. Finally: estimate the error of $\hat{f}$ using *test* data.
### Main idea behind hold-out validation

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on **Training** data $\rightarrow$ $err(\hat{f}_h, \text{Validation})$.

<table>
<thead>
<tr>
<th>Training + Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on **Training + Validation** data $\rightarrow$ $err(\hat{f}_h, \text{Test})$.

The hope is that these quantities are similar! (Making this rigorous is actually rather tricky.)
Main idea behind hold-out validation

Training | Validation | Test
--- | --- | ---
Classifier \( \hat{f}_h \) trained on Training data \( \rightarrow \) err(\( \hat{f}_h \), Validation).

Training + Validation | Test
--- | ---
Classifier \( \hat{f}_h \) trained on Training + Validation data \( \rightarrow \) err(\( \hat{f}_h \), Test).

The hope is that these quantities are similar!
Main idea behind hold-out validation

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on Training data $\rightarrow \text{err}(\hat{f}_h, \text{Validation})$.

<table>
<thead>
<tr>
<th>Training + Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on Training + Validation data $\rightarrow \text{err}(\hat{f}_h, \text{Test})$.

The hope is that these quantities are similar!

(Making this rigorous is actually rather tricky.)
Beyond simple hold-out validation

Standard hold-out:

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier \( \hat{f}_h \) trained on *Training* data \( \rightarrow \) err(\( \hat{f}_h \), *Validation*).

Could also:

- Train \( \hat{f}_h \) using *Validation* data,
- Evaluate \( \hat{f}_h \) using *Training* data.
### Beyond Simple Hold-out Validation

Standard hold-out:

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on *Training* data $\rightarrow$ err($\hat{f}_h$, *Validation*).

Could also:

- train $\hat{f}_h$ using *Validation* data, and
- evaluate $\hat{f}_h$ using *Training* data.

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on *Validation* data $\rightarrow$ err($\hat{f}_h$, *Training*).
Beyond simple hold-out validation

Standard hold-out:

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on Training data $\rightarrow$ $\text{err}(\hat{f}_h, \text{Validation})$.

Could also:

1. train $\hat{f}_h$ using Validation data, and
2. evaluate $\hat{f}_h$ using Training data.

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
</table>

Classifier $\hat{f}_h$ trained on Validation data $\rightarrow$ $\text{err}(\hat{f}_h, \text{Training})$.

Idea: Do both, and average results as overall validation error for $h$. 
**Model selection by $K$-fold cross validation**

**Model selection:**

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

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

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Training</th>
<th>Training</th>
<th>Validation</th>
<th>Training</th>
</tr>
</thead>
</table>

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

**Model assessment:**

6. Finally: estimate the error of $\hat{f}$ using test data.
How to choose $K$?

Argument for small $K$
Better simulates “variation” between different training samples drawn from underlying distribution.

$K = 2$

<table>
<thead>
<tr>
<th>Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>Training</td>
</tr>
</tbody>
</table>

$K = 4$

| Validation | Training | Training | Training |
| Training | Validation | Training | Training |
| Training | Training | Validation | Training |
| Training | Training | Training | Validation |
How to choose $K$?

Argument for small $K$
Better simulates “variation” between different training samples drawn from underlying distribution.

<table>
<thead>
<tr>
<th>$K = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Validation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$K = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Training</td>
</tr>
</tbody>
</table>

Argument for large $K$
Some learning algorithms exhibit *phase transition* behavior (e.g., output is complete rubbish until sample size sufficiently large). Using large $K$ best simulates training on all data (except test, of course).
How to choose $K$?

### Argument for small $K$

Better simulates “variation” between different training samples drawn from underlying distribution.

<table>
<thead>
<tr>
<th>$K = 2$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td></td>
<td>Validation</td>
</tr>
<tr>
<td>Validation</td>
<td></td>
<td>Training</td>
</tr>
</tbody>
</table>

| $K = 4$ |  |  |  |  |
| --- | --- | --- | --- |
| Validation | Training | Training | Training |
| Training | Validation | Training | Training |
| Training | Training | Validation | Training |
| Training | Training | Training | Validation |

### Argument for large $K$

Some learning algorithms exhibit *phase transition* behavior (e.g., output is complete rubbish until sample size sufficiently large). Using large $K$ best simulates training on all data (except test, of course).

**In practice: usually** $K = 5$ or $K = 10$. 
Recap

- **Model selection**: goal is to pick best model (e.g., features, kernels, hyper-parameter settings) to achieve low true error.
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. (Sometimes "averaging" the models in some way can help.)
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.
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.

(Sometimes “averaging” the models in some way can help.)