Version Spaces + Candidate Elimination

Lecture Outline:

- Quick Review of Concept Learning and General-to-Specific Ordering
- Version Spaces
- The Candidate Elimination Algorithm
- Inductive Bias

Reading:
Chapter 2 of Mitchell
Version Spaces

- One limitation of the **FIND-S** algorithm is that it outputs just one hypothesis consistent with the training data – there might be many.

  To overcome this, introduce notion of **version space** and algorithms to compute it.
Version Spaces

- One limitation of the **FIND-S** algorithm is that it outputs just one hypothesis consistent with the training data – there might be many.

To overcome this, introduce notion of **version space** and algorithms to compute it.

- A hypothesis \( h \) is **consistent** with a set of training examples \( D \) of target concept \( c \) if and only if \( h(x) = c(x) \) for each training example \( \langle x, c(x) \rangle \) in \( D \).

\[
\text{Consistent}(h, D) \equiv (\forall \langle x, c(x) \rangle \in D \ h(x) = c(x))
\]
Version Spaces

- One limitation of the **FIND-S** algorithm is that it outputs just one hypothesis consistent with the training data – there might be many.

  To overcome this, introduce notion of **version space** and algorithms to compute it.

- A hypothesis \( h \) is **consistent** with a set of training examples \( D \) of target concept \( c \) if and only if \( h(x) = c(x) \) for each training example \( \langle x, c(x) \rangle \) in \( D \).

  \[
  Consistent(h, D) \equiv (\forall \langle x, c(x) \rangle \in D) \; h(x) = c(x)
  \]

- The **version space**, \( V_{H,D} \), with respect to hypothesis space \( H \) and training examples \( D \), is the subset of hypotheses from \( H \) consistent with all training examples in \( D \).

  \[
  V_{H,D} \equiv \{ h \in H | Consistent(h, D) \}
  \]
Version Spaces

• One limitation of the FIND-S algorithm is that it outputs just one hypothesis consistent with the training data – there might be many.

To overcome this, introduce notion of version space and algorithms to compute it.

• A hypothesis \( h \) is consistent with a set of training examples \( D \) of target concept \( c \) if and only if \( h(x) = c(x) \) for each training example \( \langle x, c(x) \rangle \) in \( D \).

\[
Consistent(h, D) \equiv (\forall \langle x, c(x) \rangle \in D) \ h(x) = c(x)
\]

• The version space, \( V_{S_{H,D}} \), with respect to hypothesis space \( H \) and training examples \( D \), is the subset of hypotheses from \( H \) consistent with all training examples in \( D \).

\[
V_{S_{H,D}} \equiv \{ h \in H | Consistent(h, D) \}
\]

• Note difference between definitions of consistent and satisfies:
  – an example \( x \) satisfies hypothesis \( h \) when \( h(x) = 1 \), regardless of whether \( x \) is +ve or −ve example of target concept
  – an example \( x \) is consistent with hypothesis \( h \) iff \( h(x) = c(x) \)
The **List-Then-Eliminate** Algorithm

- Can represent version space by listing all members.

- Leads to **List-Then-Eliminate** concept learning algorithm:

  1. $\text{VersionSpace} \leftarrow$ a list containing every hypothesis in $H$
  2. For each training example, $\langle x, c(x) \rangle$
     remove from $\text{VersionSpace}$ any hypothesis $h$ for which $h(x) \neq c(x)$
  3. Output the list of hypotheses in $\text{VersionSpace}$

- **List-Then-Eliminate** works in principle, so long as version space is finite.

- However, since it requires exhaustive enumeration of all hypotheses in practice it is not feasible.

- Is there a more compact way to represent version spaces?
The **Candidate-Elimination Algorithm**

- The **Candidate-Elimination** algorithm is similar to **List-Then-Eliminate** algorithm but uses a more compact representation of version space.
  - represents version space by its **most general** and **most specific** members
The Candidate-Elimination Algorithm

- The Candidate-Elimination algorithm is similar to List-Then-Eliminate algorithm but uses a more compact representation of version space.
  - represents version space by its most general and most specific members
- For EnjoySport example Find-S outputs the hypothesis: \( h = \langle Sunny, Warm, ?, Strong, ?, ? \rangle \) which was one of 6 hypotheses consistent with the data.

\[ S: \{ \langle Sunny, Warm, ?, Strong, ?, ? \rangle \} \]

\[ G: \{ \langle Sunny, ?, ?, Strong, ?, ? \rangle, \langle Sunny, Warm, ?, ?, ?, ? \rangle, \langle ?, Warm, ?, Strong, ?, ? \rangle \} \]
The Candidate-Elimination Algorithm

- The Candidate-Elimination algorithm is similar to List-Then-Eliminate algorithm but uses a more compact representation of version space.
  - represents version space by its most general and most specific members

- For EnjoySport example Find-S outputs the hypothesis: \( h = \langle \text{Sunny, Warm, ?, Strong, ?, ?} \rangle \)
  which was one of 6 hypotheses consistent with the data.

\[
S: \{ \langle \text{Sunny, Warm, ?, Strong, ?, ?} \rangle \}
\]

\[
G: \{ \langle \text{Sunny, ?, ?, ?, ?, ?} \rangle, \langle \text{?, Warm, ?, ?, ?, ?} \rangle \}
\]

- The Candidate-Elimination algorithm represents the version space by recording only the most general members (\( G \)) and its most specific members (\( S \))
  - other intermediate members in general-to-specific ordering can be generated as needed
The Candidate-Elimination Algorithm (cont)

- The **General boundary**, G, of version space $V_{SH,D}$ is the set of its maximally general members.

- The **Specific boundary**, S, of version space $V_{SH,D}$ is the set of its maximally specific members.
The Candidate-Elimination Algorithm (cont)

- The **General boundary**, \(G\), of version space \(V_{SH,D}\) is the set of its maximally general members.

- The **Specific boundary**, \(S\), of version space \(V_{SH,D}\) is the set of its maximally specific members.

- **Version Space Representation Theorem**
  Every member of the version space lies between these boundaries

  \[
  V_{SH,D} = \{ h \in H | (\exists s \in S)(\exists g \in G)(g \geq g h \geq g s) \}
  \]

  where \(x \geq g y\) means \(x\) is more general or equal to \(y\)
  (see Mitchell, p. 32, for proof)
The **Candidate-Elimination Algorithm** (cont)

- The **General boundary**, \( G \), of version space \( V_{S_{H,D}} \) is the set of its maximally general members.

- The **Specific boundary**, \( S \), of version space \( V_{S_{H,D}} \) is the set of its maximally specific members.

- **Version Space Representation Theorem**
  Every member of the version space lies between these boundaries

\[
V_{S_{H,D}} = \{ h \in H | (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s) \}
\]

where \( x \geq_g y \) means \( x \) is more general or equal to \( y \)
(see Mitchell, p. 32, for proof)

- Intuitively, **Candidate-Elimination** algorithm proceeds by
  - initialising \( G \) and \( S \) to the maximally general and maximally specific hypotheses in \( H \)
  - considering each training example in turn and
    - using positive examples to drive the maximally specific boundary up
    - using negative examples to drive the maximally general boundary down
The **CANDIDATE-ELIMINATION Algorithm** (cont)

\[
G \leftarrow \text{maximally general hypotheses in } H \\
S \leftarrow \text{maximally specific hypotheses in } H
\]
The **Candidate-Elimination Algorithm** (cont)

\[ G \leftarrow \text{maximally general hypotheses in } H \]
\[ S \leftarrow \text{maximally specific hypotheses in } H \]

For each training example \( d \), do
The **Candidate-Elimination Algorithm** (cont)

$G \leftarrow$ maximally general hypotheses in $H$

$S \leftarrow$ maximally specific hypotheses in $H$

For each training example $d$, do

- If $d$ is a positive example
The Candidate-Elimination Algorithm (cont)

\[ G \leftarrow \text{maximally general hypotheses in } H \]
\[ S \leftarrow \text{maximally specific hypotheses in } H \]

For each training example \( d \), do

- If \( d \) is a positive example
  - Remove from \( G \) any hypothesis inconsistent with \( d \)
The Candidate-Elimination Algorithm (cont)

\[ G \leftarrow \text{maximally general hypotheses in } H \]
\[ S \leftarrow \text{maximally specific hypotheses in } H \]

For each training example \( d \), do

- If \( d \) is a positive example
  - Remove from \( G \) any hypothesis inconsistent with \( d \)
  - For each hypothesis \( s \) in \( S \) that is not consistent with \( d \)
The Candidate-Elimination Algorithm (cont)

\[ G \leftarrow \text{maximally general hypotheses in } H \]
\[ S \leftarrow \text{maximally specific hypotheses in } H \]

For each training example \( d \), do

- If \( d \) is a positive example
  - Remove from \( G \) any hypothesis inconsistent with \( d \)
  - For each hypothesis \( s \) in \( S \) that is not consistent with \( d \)
    * Remove \( s \) from \( S \)
The **Candidate-Elimination Algorithm** (cont)

$G \leftarrow$ maximally general hypotheses in $H$

$S \leftarrow$ maximally specific hypotheses in $H$

For each training example $d$, do

- If $d$ is a positive example
  - Remove from $G$ any hypothesis inconsistent with $d$
  - For each hypothesis $s$ in $S$ that is not consistent with $d$
    * Remove $s$ from $S$
    * Add to $S$ all minimal generalizations $h$ of $s$ such that
The **Candidate-Elimination Algorithm** (cont)

\[ G \leftarrow \text{maximally general hypotheses in } H \]
\[ S \leftarrow \text{maximally specific hypotheses in } H \]

For each training example \( d \), do

- If \( d \) is a positive example
  - Remove from \( G \) any hypothesis inconsistent with \( d \)
  - For each hypothesis \( s \) in \( S \) that is not consistent with \( d \)
    * Remove \( s \) from \( S \)
    * Add to \( S \) all minimal generalizations \( h \) of \( s \) such that
      1. \( h \) is consistent with \( d \), and
      2. some member of \( G \) is more general than \( h \)
The Candidate-Elimination Algorithm (cont)

\[ G \leftarrow \text{maximally general hypotheses in } H \]
\[ S \leftarrow \text{maximally specific hypotheses in } H \]

For each training example \( d \), do

- If \( d \) is a positive example
  - Remove from \( G \) any hypothesis inconsistent with \( d \)
  - For each hypothesis \( s \) in \( S \) that is not consistent with \( d \)
    * Remove \( s \) from \( S \)
    * Add to \( S \) all minimal generalizations \( h \) of \( s \) such that
      1. \( h \) is consistent with \( d \), and
      2. some member of \( G \) is more general than \( h \)
  * Remove from \( S \) any hypothesis that is more general than another hypothesis in \( S \)
The CANDIDATE-ELIMINATION Algorithm (cont)

\[ G \leftarrow \text{maximally general hypotheses in } H \]
\[ S \leftarrow \text{maximally specific hypotheses in } H \]

For each training example \( d \), do

- If \( d \) is a positive example
  - Remove from \( G \) any hypothesis inconsistent with \( d \)
  - For each hypothesis \( s \) in \( S \) that is not consistent with \( d \)
    * Remove \( s \) from \( S \)
    * Add to \( S \) all minimal generalizations \( h \) of \( s \) such that
      1. \( h \) is consistent with \( d \), and
      2. some member of \( G \) is more general than \( h \)
    * Remove from \( S \) any hypothesis that is more general than another hypothesis in \( S \)

- If \( d \) is a negative example
  - Remove from \( S \) any hypothesis inconsistent with \( d \)
  - For each hypothesis \( g \) in \( G \) that is not consistent with \( d \)
    * Remove \( g \) from \( G \)
    * Add to \( G \) all minimal specializations \( h \) of \( g \) such that
      1. \( h \) is consistent with \( d \), and
      2. some member of \( S \) is more specific than \( h \)
    * Remove from \( G \) any hypothesis that is less general than another hypothesis in \( G \)
The **Candidate-Elimination Algorithm**: Example

Training Examples:

T1: \(\langle\text{Sunny, Warm, Normal, Strong, Warm, Same}\rangle, \text{Yes}\)

T2: \(\langle\text{Sunny, Warm, High, Strong, Warm, Same}\rangle, \text{Yes}\)

T3: \(\langle\text{Rainy, Cold, High, Strong, Warm, Change}\rangle, \text{No}\)

T4: \(\langle\text{Sunny, Warm, High, Strong, Cool, Change}\rangle, \text{Yes}\)
The Candidate-Elimination Algorithm: Remarks

- Version space learned by Candidate-Elimination algorithm will converge towards correct hypothesis provided:
  - no errors in training examples
  - there is a hypothesis in $H$ that describes target concept

In such cases algorithm may converge to empty version space
The Candidate-Elimination Algorithm: Remarks

- Version space learned by Candidate-Elimination algorithm will converge towards correct hypothesis provided:
  - no errors in training examples
  - there is a hypothesis in $H$ that describes target concept
In such cases algorithm may converge to empty version space

- If algorithm can request next training example (e.g. from teacher) can increase speed of convergence by requesting examples that split the version space
  - E.g. T5: $\langle\text{Sunny, Warm, Normal, Light, Warm, Same}\rangle$ satisfies 3 hypotheses in previous example
    * If T5 positive, $S$ generalised, 3 hypotheses eliminated
    * If T5 negative, $G$ specialised, 3 hypotheses eliminated
  - Optimal query strategy is to request examples that exactly split version space – converge in $\lceil\log_2|VS|\rceil$ steps. However, this is not always possible.
The Candidate-Elimination Algorithm: Remarks (cont)

- When using (i.e. not training) a classifier that has not completely converged, new examples may be
  1. classed as positive by all $h \in VS$
  2. classed as negative by all $h \in VS$
  3. classed as positive by some, and negative by other, $h \in VS$

Cases 1 and 2 are unproblematic. In case 3. may want to consider proportion of positive vs. negative classifications (but then a priori probabilities of hypotheses are relevant)
Inductive Bias

As noted, version space learned by **Candidate-Elimination** algorithm will converge towards correct hypothesis provided:

- no errors in training examples
- there is a hypothesis in $H$ that describes target concept

What if no concept in $H$ that describes the target concept?
**Inductive Bias**

- As noted, version space learned by **Candidate-Elimination** algorithm will converge towards correct hypothesis provided:
  - no errors in training examples
  - there is a hypothesis in $H$ that describes target concept

What if no concept in $H$ that describes the target concept?

- Consider the training data

<table>
<thead>
<tr>
<th>Example</th>
<th>Sky</th>
<th>Temp</th>
<th>Humid</th>
<th>Wind</th>
<th>Water</th>
<th>Forecast</th>
<th>EnjoySport</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunny</td>
<td>Warm</td>
<td>Normal</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Cloudy</td>
<td>Warm</td>
<td>Normal</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Rainy</td>
<td>Warm</td>
<td>Normal</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>No</td>
</tr>
</tbody>
</table>
Inductive Bias

- As noted, version space learned by **Candidate-Elimination** algorithm will converge towards correct hypothesis provided:
  - no errors in training examples
  - there is a hypothesis in $H$ that describes target concept

What if no concept in $H$ that describes the target concept?

- Consider the training data

<table>
<thead>
<tr>
<th>Example</th>
<th>Sky</th>
<th>Temp</th>
<th>Humid</th>
<th>Wind</th>
<th>Water</th>
<th>Forecast</th>
<th>EnjoySport</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sunny</td>
<td>Warm</td>
<td>Normal</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Cloudy</td>
<td>Warm</td>
<td>Normal</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Rainy</td>
<td>Warm</td>
<td>Normal</td>
<td>Strong</td>
<td>Warm</td>
<td>Same</td>
<td>No</td>
</tr>
</tbody>
</table>

- No hypotheses consistent with 3 examples.
  Most specific hypothesis consistent with Ex 1 and 2 *and representable in* $H$:

$$\langle ?, \text{Warm, Normal, Strong, Warm, Same} \rangle$$

But this is inconsistent with Ex 3.
Inductive Bias (cont)

- Need more expressive hypothesis representation language.
  E.g. allow disjunctive or negative attribute values:

\[
\begin{align*}
\text{Sky} &= \text{Sunny} \lor \text{Cloudy} \\
\text{Sky} &\not= \text{Rainy}
\end{align*}
\]
An Unbiased Learner

• What about ensuring every concept can be represented in $H$?
  – Since concepts are subsets of instance space $X$, want $H$ to be able to represent any set in power set of $X$
    * for *EnjoySport* there were 96 possible instances
    so, power set contains $2^{96} \approx 10^{28}$ possible target concepts
    * recall biased conjunctive hypothesis space can represent only 973 of these

• Can do this by allowing hypotheses that are arbitrary conjunctions, disjunctions and negations of our earlier hypotheses
  – New problem: concept learning algorithm cannot generalise beyond observed examples!
    * $S$ boundary = disjunction of positive examples – exactly covers observed positive examples
    * $G$ boundary = negation of disjunction of negative examples – exactly rules out observed negative examples
An Unbiased Learner

- Capacity of Candidate-Elimination to generalise lies in its implicit assumption of bias – that target concept can be represented as a conjunction of attribute values

- Fundamental property of inductive inference:

  a learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances

  I.e. bias-free learning is futile
Inductive Bias, More Formally

- Since all inductive learning involves bias, useful to characterise learning approaches by the type of bias they employ

- Consider
  - concept learning algorithm $L$
  - instances $X$, target concept $c$
  - training examples $D_c = \{\langle x, c(x) \rangle \}$
  - let $L(x_i, D_c)$ denote the classification, positive or negative, assigned to the instance $x_i$ by $L$ after training on data $D_c$.

Definition:
The **inductive bias** of $L$ is any minimal set of assertions $B$ such that for any target concept $c$ and corresponding training examples $D_c$

\[
(\forall x_i \in X)[(B \land D_c \land x_i) \vdash L(x_i, D_c)]
\]

where $A \vdash B$ means $A$ logically entails $B$
Modelling Inductive Systems by Deductive Systems

Inductive system

Training examples → Candidate Elimination Algorithm → Using Hypothesis Space $H$ → Classification of new instance, or "don't know"

New instance →

Classification of new instance, or "don't know"

Equivalent deductive system

Training examples →

Theorem Prover → Classification of new instance, or "don't know"

New instance →

Assertion "$H$ contains the target concept"

Inductive bias made explicit
Summary

- The **version space** with respect to a hypothesis space $H$ and a set of training examples $D$ is the subset of all hypotheses in $H$ **consistent** with all the examples in $D$. 
Summary

- The **version space** with respect to a hypothesis space $H$ and a set of training examples $D$ is the subset of all hypotheses in $H$ **consistent** with all the examples in $D$.

- The version space may be compactly represented by recording its **general boundary** $G$ and **specific boundary** $S$.
  Every hypothesis in the version space is guaranteed to lie between $G$ and $S$ by the **version space representation theorem**.
Summary

- The **version space** with respect to a hypothesis space $H$ and a set of training examples $D$ is the subset of all hypotheses in $H$ **consistent** with all the examples in $D$.

- The version space may be compactly represented by recording its **general boundary** $G$ and **specific boundary** $S$.
  Every hypothesis in the version space is guaranteed to lie between $G$ and $S$ by the **version space representation theorem**.

- The **Candidate-Elimination algorithm** exploits this theorem by searching for $H$ for the version space by using the examples in training data $D$ to progressively generalise the specific boundary and specialise the general boundary.
Summary

- The **version space** with respect to a hypothesis space $H$ and a set of training examples $D$ is the subset of all hypotheses in $H$ consistent with all the examples in $D$.

- The version space may be compactly represented by recording its **general boundary** $G$ and **specific boundary** $S$.
  Every hypothesis in the version space is guaranteed to lie between $G$ and $S$ by the **version space representation theorem**.

- The **Candidate-Elimination algorithm** exploits this theorem by searching for $H$ for the version space by using the examples in training data $D$ to progressively generalise the specific boundary and specialise the general boundary.

- There are certain concepts the **Candidate-Elimination** algorithm cannot learn because of the **bias** of the hypothesis space – every concept must be representable as a conjunction of attribute values.
Summary

• The version space with respect to a hypothesis space $H$ and a set of training examples $D$ is the subset of all hypotheses in $H$ consistent with all the examples in $D$.

• The version space may be compactly represented by recording its general boundary $G$ and specific boundary $S$.
  Every hypothesis in the version space is guaranteed to lie between $G$ and $S$ by the version space representation theorem.

• The Candidate-Elimination algorithm exploits this theorem by searching for $H$ for the version space by using the examples in training data $D$ to progressively generalise the specific boundary and specialise the general boundary.

• There are certain concepts the Candidate-Elimination algorithm cannot learn because of the bias of the hypothesis space – every concept must be representable as a conjunction of attribute values.

• In fact, all inductive learning supposes some a priori assumptions about the nature of the target concept, or else there is no basis for generalisation beyond observed examples: bias-free learning is futile.