15-466
Computer Game Programming

Learning I:
Learning to Adjust, Learning to Predict

Maxim Likhachev
Robotics Institute
Carnegie Mellon University
Purpose/benefits of Learning

• More believable (less predictable) characters

• Less work to program decisions for all the possible situations
Purpose/benefits of Learning

• More believable (less predictable) characters

• Less work to program decisions for all possible situations

Challenges using learning in games?

Lack of control over learnt behaviors

Overfitting
Types of Learning

• Offline
  - between levels of games
  - at the game development studio before the game is released

• Online
  - during the game itself

*Most common approach*

*Allows to test the learnt behaviors*
Types of Learning

• Adaptation of the behavior parameters (intra-behavior learning)
  - learning to target precisely
  - learning the best patrol routes on the current level
  - learning good set of cover points for the given room
  - …
Types of Learning

- Predicting the behavior of the enemy
  - predicting the next move in a fight based on the last few moves
  - predicting the attack based on the assembly of enemy troops
  - …
Types of Learning

- Reacting to the behavior of the enemy (inter-behavior learning)
  - making the most effective counter-move in a fight
  - making the most effective reallocation of the troops
  - making the best move in a board game based on all the pieces
  - …
Adaptation of the Parameters

• Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function: \( \Psi^* = \arg \max_{\Psi} f(\Psi) \)

Start with the best guess for the parameter vector \( \Psi \)

Until no further improvement

try changing \( \Psi \) by small \( \Delta \) in all directions and pick the best:

\[
\Psi = \Psi + \arg \max_{\Delta} f(\Psi + \Delta)
\]
Adaptation of the Parameters

- **Hill Climbing**

  The goal is to find the parameter vector that results in the most optimal value of the function:
  \[ \Psi^* = \arg\max_\Psi f(\Psi) \]

  Start with the best guess for the parameter vector \( \Psi \)
  Until no further improvement
  try changing \( \Psi \) by small \( \Delta \) in all directions and pick the best:
  \[ \Psi = \Psi + \arg\max_\Delta f(\Psi + \Delta) \]

  \[ \text{Energy (fitness or score)} \]

  \[ \text{Parameter value} \]

  from “Artificial Intelligence for Games” by I. Millington & J. Funge
Adaptation of the Parameters

• Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function:

$$\Psi^* = \text{argmax}_\Psi f(\Psi)$$

Start with the best guess for the parameter vector $\Psi$

Until no further improvement

try changing $\Psi$ by small $\Delta$ in all directions and pick the best:

$$\Psi = \Psi + \text{argmax}_\Delta f(\Psi + \Delta)$$

$\Delta$ is often adapted depending on the rate of improvement in $f(\Psi + \Delta)$

from “Artificial Intelligence for Games” by I. Millington & J. Funge
Adaptation of the Parameters

• Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function:

\[ \Psi^* = \arg\max_{\Psi} f(\Psi) \]

Start with the best guess for the parameter vector \( \Psi \)

Until no further improvement

try changing \( \Psi \) by small \( \Delta \) in all directions and pick the best:

\[ \Psi = \Psi + \arg\max_{\Delta} f(\Psi + \Delta) \]

Example:

finding the best shooting distance

\( \Psi = \text{distance (scalar variable)} \)

\( \Delta = +/-\text{distance increment} \)

\( f(\Psi) = \text{distance between enemy ship and where cannonball lands} \)
Adaptation of the Parameters

• Hill Climbing

*The goal is to find the parameter vector that results in the most optimal value of the function:* \( \Psi^* = \text{argmax}_\Psi f(\Psi) \)

*Start with the best guess for the parameter vector \( \Psi \)
*Until no further improvement*

*try changing \( \Psi \) by small \( \Delta \) in all directions and pick the best:*

\[ \Psi = \Psi + \text{argmax}_\Delta f(\Psi + \Delta) \]

Equivalent formulation for differentiable functions (Steepest ascent):

*Start with the best guess for the parameter vector \( X \)
*Until no further improvement*

\[ X = X + \varepsilon f'(X) \]
Adaptation of the Parameters

• Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function: 

$$\Psi^* = \arg\max_\Psi f(\Psi)$$

Start with the best guess for the parameter vector $\Psi$

Until no further improvement

try changing $\Psi$ by small $\Delta$ in all directions and pick the best:

$$\Psi = \Psi + \arg\max_\Delta f(\Psi + \Delta)$$

Equivalent formulation for differentiable functions (Steepest descent):

Start with the best guess for the parameter vector $X$

Until no further improvement

$$X = X + \varepsilon f'(X)$$

for small $\varepsilon > 0$
Adaptation of the Parameters

• Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function: \( \Psi^* = \arg \max_\Psi f(\Psi) \)

Start with the best guess for the parameter vector \( \Psi \)

Until no further improvement

try changing \( \Psi \) by small \( \Delta \) in all directions and pick the best:

\[
\Psi = \Psi + \arg \max_\Delta f(\Psi + \Delta)
\]

Equivalent formulation for differentiable functions (Steepest ascent):

Start with the best guess for the parameter vector \( X \)

Until no further improvement

\[
X = X + \varepsilon \nabla f(X)
\]

For multi-dimensional vector \( X \),

\[
f'(X) \equiv \nabla f(X)
\]
Adaptation of the Parameters

• Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function: \[ \Psi^* = \arg\max_{\Psi} f(\Psi) \]

Start with the best guess for the parameter vector \( \Psi \)

Until no further improvement

try changing \( \Psi \) by small \( \Delta \) in all directions and pick the best:

\[ \Psi = \Psi + \arg\max_{\Delta} f(\Psi + \Delta) \]

Equivalent formulation for differentiable functions (Steepest ascent):

Start with the best guess for the parameter vector \( X \)

Until no further improvement

\[ X = X + \varepsilon \nabla f(X) \]
Adaptation of the Parameters

• Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function:

$$\Psi^* = \arg\max_{\Psi} f(\Psi)$$

Start with the best guess for the parameter vector $\Psi$

Until no further improvement

try changing $\Psi$ by small $\Delta$ in all directions and pick the best:

$$\Psi = \Psi + \arg\max_{\Delta} f(\Psi + \Delta)$$

Equivalent formulation for differentiable functions (Steepest ascent):

Start with the best guess for the parameter vector $X$

Until no further improvement

$$X = X + \epsilon \nabla f(X)$$
Adaptation of the Parameters

- Hill Climbing

The goal is to find the parameter vector that results in the most optimal value of the function:

$$\Psi^* = \arg\max_{\Psi} f(\Psi)$$

Start with the best guess for the parameter vector \( \Psi \)

Until no further improvement

try changing \( \Psi \) by small \( \Delta \) in all directions and pick the best:

$$\Psi = \Psi + \arg\max_{\Delta} f(\Psi + \Delta)$$

from “Artificial Intelligence for Games” by I. Millington & J. Funge
Adaptation of the Parameters

• Extensions to Hill Climbing to overcome local minima problem

- Momentum: persist in going in the same direction (when picking the next $\Delta$, the previous direction $\Delta$ gets an additional term proportional to the previous improvement)

Disadvantages?

from “Artificial Intelligence for Games” by I. Millington & J. Funge
Adaptation of the Parameters

• Extensions to Hill Climbing to overcome local minima problem

- Multiple trials: try random initial locations

Disadvantages, in particular in the context of games?

from “Artificial Intelligence for Games” by I. Millington & J. Funge
Adaptation of the Parameters

- Extensions to Hill Climbing to overcome local minima problem
  - Simulated Annealing: the selection of $\Delta$ is done at random

Start with the best guess for the parameter vector $\Psi$

Until no further improvement
  pick a random direction $\Delta$
  With probability $P$, set $\Psi = \Psi + \Delta$, where $P$ is a function of the improvement $(f(\Psi+\Delta)-f(\Psi))$ and temperature $T$
Adaptation of the Parameters

• Extensions to Hill Climbing to overcome local minima problem

- Simulated Annealing: the selection of $\Delta$ is done at random

Start with the best guess for the parameter vector $\Psi$

Until no further improvement

pick a random direction $\Delta$

With probability $P$, set $\Psi = \Psi + \Delta$, where $P$ is a function of the improvement ($f(\Psi + \Delta) - f(\Psi)$) and temperature $T$

$P$ is increasing as $f(\Psi + \Delta) - f(\Psi)$ increases

$P$ is decreasing as $T$ decreases

Temperature $T$ is being decreased over time

As $T$ gets closer to 0 ($T$ “cools off”), the algorithm becomes greedy descent

Disadvantages?
Action Prediction

• Predict future action of the player based on past actions and anything else relevant

Any ideas how to do it?
Action Prediction

• N-gram predictor
  - Very popular in all the combat (martial arts, boxing, swords, …) games (e.g., predicting next move)
  - Often requires reduction in the power of AI to make it beatable
  - Extends to other predictions such as what weapons will be used or how the attacks will occur
Action Prediction

• N-gram predictor
  - Maintain the probabilities of future actions based on N-1 preceding observations
  - For prediction, always return the most likely action based on last N-1 observations
Action Prediction

• N-gram predictor
  - Maintain the probabilities of future actions based on N-1 preceding observations
  - For prediction, always return the most likely action based on last N-1 observations

Example of applying 3-gram predictor:

training data (observed sequence of moves): LRRLRLLLRRRRLRR

learnt prediction table:

<table>
<thead>
<tr>
<th></th>
<th>..R</th>
<th>..L</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>LR</td>
<td>(\frac{3}{5})</td>
<td>(\frac{2}{5})</td>
</tr>
<tr>
<td>RL</td>
<td>(\frac{3}{4})</td>
<td>(\frac{1}{4})</td>
</tr>
<tr>
<td>RR</td>
<td>(\frac{0}{2})</td>
<td>(\frac{2}{2})</td>
</tr>
</tbody>
</table>

There were 5 LRIs
3 resulted in R
2 resulted in L
Action Prediction

- N-gram predictor
  - Maintain the probabilities of future actions based on N-1 preceding observations
  - For prediction, always return the most likely action based on last N-1 observations

Example of applying 3-gram predictor:

training data (observed sequence of moves): LRRLRLLRRLRLRR

learnt prediction table:

<table>
<thead>
<tr>
<th></th>
<th>..R</th>
<th>..L</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>LR</td>
<td>3/5</td>
<td>2/5</td>
</tr>
<tr>
<td>RL</td>
<td>3/4</td>
<td>1/4</td>
</tr>
<tr>
<td>RR</td>
<td>0/2</td>
<td>2/2</td>
</tr>
</tbody>
</table>

Mathematically, what are you learning?
Action Prediction

- N-gram predictor
  - Maintain the probabilities of future actions based on N-1 preceding observations
  - For prediction, always return the most likely action based on last N-1 observations

Example of applying 3-gram predictor:

training data (observed sequence of moves): \[ LRRLRLLLRLRRLRR \]

learnt prediction table:

<table>
<thead>
<tr>
<th></th>
<th>..R</th>
<th>..L</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>( \frac{1}{2} )</td>
<td>( \frac{1}{2} )</td>
</tr>
<tr>
<td>LR</td>
<td>( \frac{3}{5} )</td>
<td>( \frac{2}{5} )</td>
</tr>
<tr>
<td>RL</td>
<td>( \frac{3}{4} )</td>
<td>( \frac{1}{4} )</td>
</tr>
<tr>
<td>RR</td>
<td>( \frac{0}{2} )</td>
<td>( \frac{2}{2} )</td>
</tr>
</tbody>
</table>

Mathematically, what are you learning?

\[ P(A=L|LL), \]
\[ P(A=L|LR), \]

...
Action Prediction

• N-gram predictor
  - Maintain the probabilities of future actions based on N-1 preceding observations
  - For prediction, always return the most likely action based on last N-1 observations

Accuracy of prediction may reduce as N increases past some point

Why?

from “Artificial Intelligence for Games” by I. Millington & J. Funge
Action Prediction

• N-gram predictor
  - *Maintain the probabilities of future actions based on N-1 preceding observations*
  - *For prediction, always return the most likely action based on last N-1 observations*

Game Performance in initial stages: what to do before the N-gram predictor is learnt?
Action Prediction

• Hierarchical N-gram predictor
  - Learn 2-gram, 3-gram, ... predictors simultaneously
  - For prediction, pick N-gram predictor with largest $N$ and sufficient number of training samples for the given input

Game Performance in initial stages: what to do before the N-gram predictor is learnt?
Action Prediction

- Hierarchical N-gram predictor
  - Learn 1-gram, 2-gram, 3-gram, ... predictors simultaneously
  - For prediction, pick N-gram predictor with largest N and sufficient number of training samples for the given input
  - If none have sufficient samples, then output random prediction
**Action Prediction**

- Hierarchical N-gram predictor
  - *Learn* 1-gram, 2-gram, 3-gram, ... *predictors simultaneously*
  - *For prediction, pick N-gram predictor with largest N and sufficient number of training samples for the given input*
  - *If none have sufficient samples, then output random prediction*

**Example:**

*training data (observed sequence of moves):* LRRLRLLLRRRLRLRR

<table>
<thead>
<tr>
<th>1-gram:</th>
<th>2-gram:</th>
<th>3-gram:</th>
</tr>
</thead>
<tbody>
<tr>
<td>L R # of samples</td>
<td>Obs. L R # of samples</td>
<td>Obs. L R # of samples</td>
</tr>
<tr>
<td>7/15 8/15 15</td>
<td>L 2/7 5/7 7</td>
<td>LL 1/2 1/2 2</td>
</tr>
<tr>
<td>R</td>
<td>4/7 3/7 7</td>
<td>LR 2/5 3/5 5</td>
</tr>
<tr>
<td>R</td>
<td>4/7 3/7 7</td>
<td>RL 1/4 3/4 4</td>
</tr>
<tr>
<td>R</td>
<td>4/7 3/7 7</td>
<td>RR 1 0 2</td>
</tr>
</tbody>
</table>
Action Prediction

• Hierarchical N-gram predictor
  - Learn 1-gram, 2-gram, 3-gram, ... predictors simultaneously
  - For prediction, pick N-gram predictor with largest N and sufficient number of training samples
  - If none have sufficient samples, output random prediction

Example:

training data (observed sequence of moves): LRRLRLLLRRLRLRR

<table>
<thead>
<tr>
<th>1-gram:</th>
<th>2-gram:</th>
<th>3-gram:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L</strong></td>
<td><strong>R</strong></td>
<td># of samples</td>
</tr>
<tr>
<td>7/15</td>
<td>8/15</td>
<td>15</td>
</tr>
<tr>
<td>Obs. L</td>
<td>Obs. L</td>
<td># of samples</td>
</tr>
<tr>
<td>L</td>
<td>2/7</td>
<td>5/7</td>
</tr>
<tr>
<td>R</td>
<td>4/7</td>
<td>3/7</td>
</tr>
<tr>
<td><strong>L</strong></td>
<td><strong>R</strong></td>
<td># of samples</td>
</tr>
<tr>
<td>Obs. L</td>
<td>Obs. L</td>
<td># of samples</td>
</tr>
<tr>
<td>LL</td>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>LR</td>
<td>2/5</td>
<td>3/5</td>
</tr>
<tr>
<td>RL</td>
<td>1/4</td>
<td>3/4</td>
</tr>
<tr>
<td>RR</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Suppose we want to have at least 4 samples for prediction, what is the predicted next action for input=RRR?
Action Prediction

• Hierarchical N-gram predictor
  - Learn 1-gram, 2-gram, 3-gram, ... predictors simultaneously
  - For prediction, pick N-gram predictor with largest N and sufficient number of training samples
  - If none have sufficient samples, output random prediction

Example:

training data (observed sequence of moves): \textbf{LRRLRLLLRRRRLRLRR}

<table>
<thead>
<tr>
<th>1-gram:</th>
<th>2-gram:</th>
<th>3-gram:</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>R</td>
<td># of samples</td>
</tr>
<tr>
<td>7/15</td>
<td>8/15</td>
<td>15</td>
</tr>
<tr>
<td>L</td>
<td>2/7</td>
<td>5/7</td>
</tr>
<tr>
<td>R</td>
<td>4/7</td>
<td>3/7</td>
</tr>
</tbody>
</table>

Suppose we want to have at least 9 samples for prediction, what is the predicted next action for input=RRR?
Action Prediction

- Hierarchical N-gram predictor
  - Learn 1-gram, 2-gram, 3-gram, ... predictors simultaneously
  - For prediction, pick N-gram predictor with largest N and sufficient number of training samples for the given input
  - If none have sufficient samples, then output random prediction

Example:

|training data (observed sequence of moves)|LRRLRLLLRRRLRLRR|

1-gram:

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
<th># of samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/15</td>
<td>8/15</td>
<td>15</td>
</tr>
</tbody>
</table>

2-gram:

<table>
<thead>
<tr>
<th>Obs.</th>
<th>L</th>
<th>R</th>
<th># of samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>2/7</td>
<td>5/7</td>
<td>7</td>
</tr>
<tr>
<td>R</td>
<td>4/7</td>
<td>3/7</td>
<td>7</td>
</tr>
</tbody>
</table>

3-gram:

<table>
<thead>
<tr>
<th>Obs.</th>
<th>L</th>
<th>R</th>
<th># of samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>LL</td>
<td>1/2</td>
<td>1/2</td>
<td>2</td>
</tr>
<tr>
<td>LR</td>
<td>2/5</td>
<td>3/5</td>
<td>5</td>
</tr>
<tr>
<td>RL</td>
<td>1/4</td>
<td>3/4</td>
<td>4</td>
</tr>
<tr>
<td>RR</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

N-gram based prediction is still hard to scale with large # of possibly relevant observations.
Action Prediction

• Naïve Bayes Classifiers
  - Scale much better to large # of input variables
  - Very popular (and powerful) for machine learning problems
Action Prediction

• Naïve Bayes Classifiers
  - predicts action $A^{predict} = \arg\max_a P(A=a) \prod_i P(X_i = u_i | A=a)$
Action Prediction

• Naïve Bayes Classifiers
  - predicts action $A^{\text{predict}} = \arg\max_a P(A=a) \prod_i P(X_i = u_i \mid A=a)$

Derivations of the above formula:
$A^{\text{predict}} = \arg\max_a P(A=a \mid X_1 = u_1 \ldots X_k = u_k)$

$A^{\text{predict}} = \arg\max_a P(A=a \mid X_1 = u_1 \ldots X_k = u_k) / P(X_1 = u_1 \ldots X_k = u_k)$

$A^{\text{predict}} = \arg\max_a P(X_1 = u_1 \ldots X_k = u_k \mid A=a)P(A=a) / P(X_1 = u_1 \ldots X_k = u_k)$

$A^{\text{predict}} = \arg\max_a P(X_1 = u_1 \ldots X_k = u_k \mid A=a)P(A=a) \quad \text{Why?}$

Assuming conditional independence of input variables given predicted output:
$A^{\text{predict}} = \arg\max_a P(A=a) \prod_i P(X_i = u_i \mid A=a)$
Action Prediction

• Naïve Bayes Classifiers
  - predicts action \( A^{\text{predict}} = \arg\max_a P(A=a) \prod_i P(X_i = u_i \mid A=a) \)

Example: Predicts whether the player will slow down (break) based on the distance to obstacle corner and the speed of the car

<table>
<thead>
<tr>
<th>brake?</th>
<th>distance</th>
<th>speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>near</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>near</td>
<td>fast</td>
</tr>
<tr>
<td>N</td>
<td>far</td>
<td>fast</td>
</tr>
<tr>
<td>Y</td>
<td>far</td>
<td>fast</td>
</tr>
<tr>
<td>N</td>
<td>near</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>far</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>near</td>
<td>fast</td>
</tr>
</tbody>
</table>

training data (previously collected): Suppose our current input is:

\( \text{Distance} = \text{far}, \text{Speed} = \text{slow} \)

Will character break?
Action Prediction

• Naïve Bayes Classifiers
  - predicts action $A^{predict} = \text{argmax}_a P(A=a) \prod_i P(X_i = u_i | A=a)$

Example: Predicts whether the player will slow down (break) based on the distance to obstacle corner and the speed of the car

Training data (previously collected): Suppose our current input is:

<table>
<thead>
<tr>
<th>brake?</th>
<th>distance</th>
<th>speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>near</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>near</td>
<td>fast</td>
</tr>
<tr>
<td>N</td>
<td>far</td>
<td>fast</td>
</tr>
<tr>
<td>Y</td>
<td>far</td>
<td>fast</td>
</tr>
<tr>
<td>N</td>
<td>near</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>far</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>near</td>
<td>fast</td>
</tr>
</tbody>
</table>

Distance = far, Speed = slow

Will character break?

$P(\text{break}) = 5/7$;

$P(\text{Dist=}\text{far}|\text{break})=2/5$; $P(\text{Speed=}\text{slow}|\text{break}) = 2/5$;

$P(A=a) \prod_i P(X_i = u_i | A=a) = 5/7*2/5*2/5=4/35$;

$P(\text{not break}) = 2/7$;

$P(\text{Dist=}\text{far}|\text{not break})=1/2$; $P(\text{Speed=}\text{slow}|\text{not break}) = 1/2$;

$P(A=a) \prod_i P(X_i = u_i | A=a) = 2/7*1/2*1/2=1/14$;
Action Prediction

• Naïve Bayes Classifiers
  - predicts action $A^{predict} = \text{argmax}_a P(A=a) \prod_i P(X_i = u_i \mid A=a)$

Example: Predicts whether the player will slow down (break) based on the distance to obstacle corner.

training data (previously collected): Suppose our current input is:

<table>
<thead>
<tr>
<th>brake?</th>
<th>distance</th>
<th>speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>near</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>near</td>
<td>fast</td>
</tr>
<tr>
<td>N</td>
<td>far</td>
<td>fast</td>
</tr>
<tr>
<td>Y</td>
<td>far</td>
<td>fast</td>
</tr>
<tr>
<td>N</td>
<td>near</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>far</td>
<td>slow</td>
</tr>
<tr>
<td>Y</td>
<td>near</td>
<td>fast</td>
</tr>
</tbody>
</table>

Distance = far, Speed = slow

Will character break?

$P(\text{break}) = 5/7$;
$P(\text{Dist=far} \mid \text{break}) = 2/5$; $P(\text{Speed=slow} \mid \text{break}) = 2/5$;
$P(A=a) \prod_i P(X_i = u_i \mid A=a) = 5/7 \times 2/5 \times 2/5 = 4/35$;

$P(\text{not break}) = 2/7$;
$P(\text{Dist=far} \mid \text{not break}) = 1/2$; $P(\text{Speed=slow} \mid \text{not break}) = 1/2$;
$P(A=a) \prod_i P(X_i = u_i \mid A=a) = 2/7 \times 1/2 \times 1/2 = 1/14$;

The character is more likely to break ($4/35 > 1/14$).