Slender: The Eight Devices

Grupo 9:
Anabela Costa e Silva, up201506034
Beatriz Souto de Sá Baldaia, up201505633

Descrição do problema de análise de dados:

O projeto baseia-se no jogo “Slender: The Eight Pages” onde o personagem principal, o jogador, é abandonado numa floresta, apenas com uma lanterna, tendo de recolher oito páginas espalhadas pelo ambiente, enquanto tenta sobreviver ao monstro, Slender Man.

No nosso projeto são oito dispositivos que têm de estar ligados ao mesmo tempo, para terminar o jogo. No entanto, cada dispositivo só fica ativo durante um tempo limite.

Cada jogador possui também um telemóvel que lhe permite comunicar com os restantes jogadores assim como ver melhor na floresta (o raio de visão aumenta). Mas o telemóvel ligado também atrai o Slender e tem bateria limitada, pelo que terá de ser recarregado.

Em contraste ao jogo original, aqui não temos apenas uma pessoa sozinha na floresta, mas sim um grupo de amigos que cooperam para que consigam escapar o mais rápido possível.

Desta forma, o nosso problema de análise será prever a vitória dos jogadores (classificação) e o tempo de execução do jogo (regressão) tendo em conta a variação das variáveis número inicial de jogadores, velocidade de corrida do Slender, velocidade de andamento do Slender, raio de visão do Slender, raio de visão do jogador com o telemóvel ligado, raio de visão do jogador com o telemóvel desligado, velocidade do jogador e temporizador do dispositivo (tempo que um dispositivo aguenta ativo depois de acionado por um jogador).

Experiências realizadas:

Usando a funcionalidade Batch do Repast Simphony corremos a nossa simulação uma vez para cada conjunto de valores dos parâmetros, obtendo um total de 13118 simulações.

Estas simulações devolvem-nos os resultados em forma de dois CSVs: GameEnd (com os valores “State” e “tick” de cada simulação) e um mapa de parâmetros de entrada. Com estes valores conseguimos recolher a informação de como correu a simulação: o seu tempo de execução (número total de ticks) e resultado final da execução (State com os possíveis valores RUNNING, LOST e VICTORY).

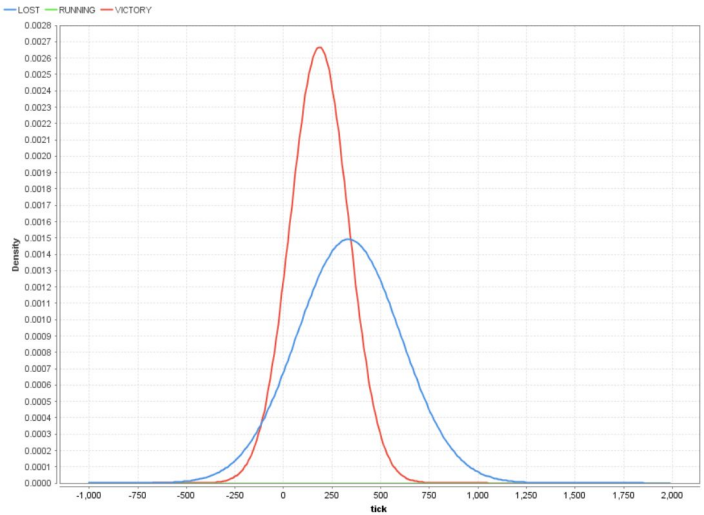
```
<parameter name="player_count" type="number" number_type="int"
    start="3" end="8" step="1">
    <parameter name="device_timeout" type="list" value_type="int"
        values="10 25 50">
        <parameter name="slender_running_speed" type="number"
            number_type="double" start="0.7" end="1.1" step="0.2">
            <parameter name="slender_radius" type="number"
                number_type="int" start="1" end="5" step="2">
                <parameter name="player_big_radius" type="number"
                    number_type="int" start="2" end="6" step="2">
                    <parameter name="player_speed" type="number"
                        number_type="int" start="1" end="3" step="1">
                        <parameter name="player_small_radius" type="number"
                            number_type="int" start="1" end="3" step="1">
                            <parameter name="slender_walking_speed" type="number"
                                number_type="double" start="0.3" end="0.7" step="0.2"></parameter>
```

Excerto do nosso ficheiro batch_params.xml com os períodos de valores para cada parâmetro

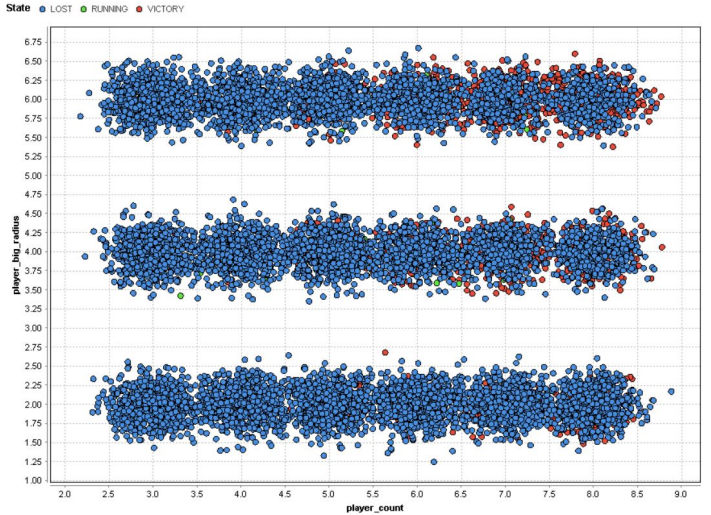
Estatísticas sobre os dados recolhidos:

Sobre os dados recolhidos retiramos algumas estatísticas, por exemplo, **2323** dos casos são **VICTORY**, **10721 LOST** e **74 RUNNING**.
Um jogo demora, em **média**, **318.152 ticks** a ser executado, tendo um **desvio padrão de 286.144**, obtendo-se um coeficiente de variação de aproximadamente 90%, o que significa que a amostra é heterogênea.

O **tempo mínimo** de execução de uma simulação é **14 ticks**, caso de vitória, em que as variáveis relativas aos atributos dos jogadores e número inicial de jogadores assumem os seus valores máximos. Já o tempo **máximo** é **2000 ticks** pois foi este o timeout que definimos.



De uma forma geral, as simulações em que se verifica vitória são mais rápidas do que as que finalizam com derrota dos jogadores.
Após testarmos diferentes gráficos de dispersão, verificamos que as variáveis de maior impacto, isto é, as que separam mais perceptualmente os casos de vitória, são as relativas a atributos dos jogadores, como se pode ver no gráfico à direita.
Assim, quanto maior é o número inicial de jogadores e quanto maior é o raio de visão destes quando ligam o telemóvel, maior é a probabilidade de eles ganharem.



Análise dos dados com RapidMiner:

Usando o RapidMiner avaliamos os dados como um **teste de classificação**, quando se analisa se o jogo acabou em Vitória (**“VICTORY”**) ou Derrota (**“LOST”**) ou quando a simulação foi interrompida (**“RUNNING”**), ao exceder os 2000 ticks.
Assim usamos os algoritmos para tentar descobrir um modelo que nos preveja o resultado antes do jogo acontecer.
Em baixo mostramos os resultados dos erros com os diferentes modelos, quer em teste, quer em treino:

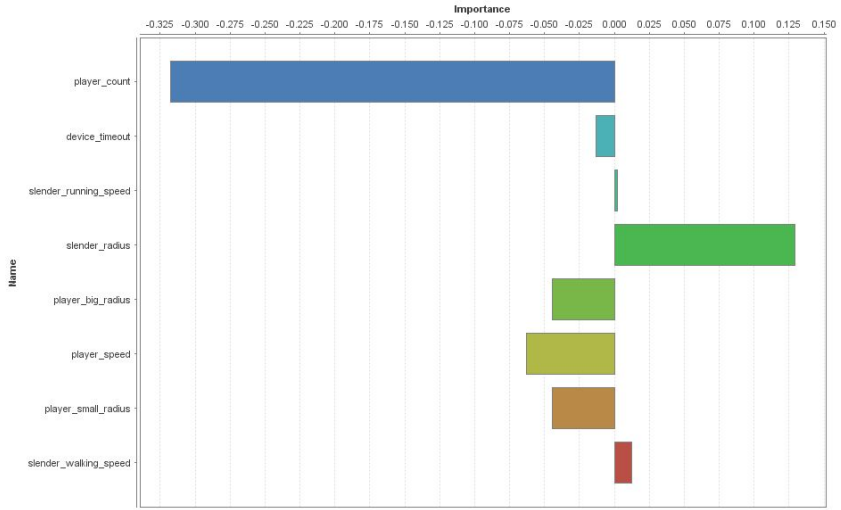
Assim, RandomForest foi o melhor, com precisão de cerca de 87%. no teste e 89% no treino.
O k-NN, nearest neighbour foi o melhor no treino 90% mas no teste não se mostrou tão bom, com apenas 85% de precisão.

Algoritmo	Precisão no treino(%)	Precisão no teste(%)
DecisionTree	89,16	86,53
DefaultModel	81,73	81,73
k-NN	90,29	85,46
LDA	85,92	85,26
NaiveBayes	87,50	86,89
RandomForest	89,70	87,80

Análise dos dados com RapidMiner:

Random Forest:

O modelo está muito exato **(87,80%)** que o mais provável é uma simulação resultar em derrota (“LOST”).
A percentagem de acertar quando afirma LOST é de 90,94%, e consegue descobrir 94,62% dos casos em que foi realmente LOST.
Apesar da alta precisão, a classe RUNNING nunca é correctamente prevista. Tal é explicado pelo facto de serem escassos os os casos em que ela ocorre, tendo em conta a dimensão do conjunto de dados (22 casos no conjunto de teste).



accuracy: 87.80%

	true LOST	true RUNNING	true VICTORY	class precision
pred. LOST	3043	19	284	90.94%
pred. RUNNING	0	0	1	0.00%
pred. VICTORY	173	3	412	70.07%
class recall	94.62%	0.00%	59.11%	

A variável **player_count** (número inicial de jogadores) é muito decisiva para o resultado final, sendo que quanto maior, maior é a probabilidade de ser vitória para os jogadores. De uma forma geral, a possibilidade de os jogadores ganharem aumenta quanto mais rápido for o processo de encontrar todos os dispositivos, pois os fatores mais importantes são relativos aos jogadores e não às características do Slender ou dos dispositivos.

Análise dos dados com RapidMiner:

Usando o RapidMiner avaliamos também os dados como um **teste de regressão** para se prever o tempo de execução do programa (número total de ticks até ao fim do jogo).
Obtivemos os seguintes resultados, usando os dados de teste:

Model	Root Mean Squared Error	Root Relative Squared Error
Decision Tree	221.652	0.807
Deep Learning	217.090	0.791
DefaultModel	274.533	1.000
Gradient Boosted Trees	211.472	0.770
k-NN	242.873	0.885
Random Forest	205.890	0.750

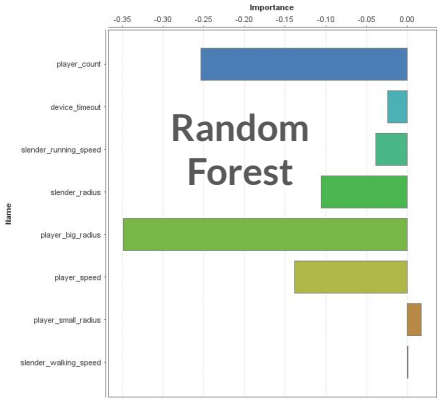
Root Mean Square Error (RMSE) representa o desvio dos resíduos (erros das previsões). Os resíduos são medidos através do quão longe os pontos (previsões) se encontram da linha de regressão. Assim, RMSE é uma medida do quão espalhados estes resíduos estão.
Os melhores resultados foram então obtidos com o modelo **Random Forest** pois é o que apresenta um menor erro (205,890). Todos os modelos (excepto o Default Model) são considerados úteis, mas, por definição, esse é o modelo trivial.

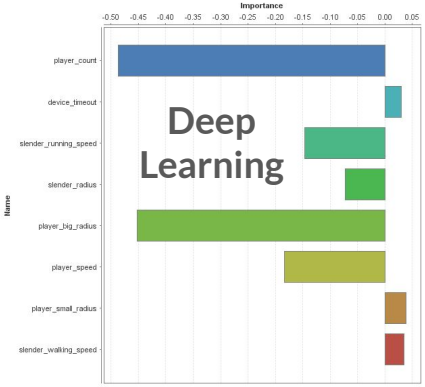
Análise dos dados com RapidMiner:

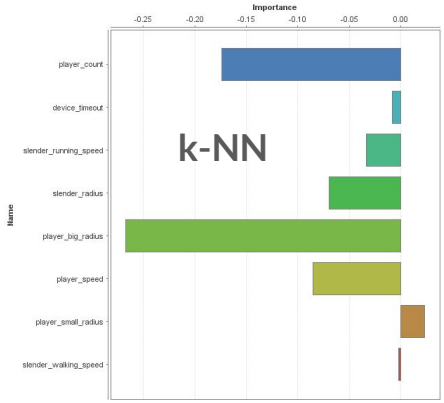
Na análise de um modelo, o diagrama de barras "**Important Factors**" serve para ver com que intensidade contribuem os atributos mais importantes para a previsões atuais. Desta forma, o valor de um atributo pode ser mais ou menos importante para a previsão, contradizendo-a ou suportando-a, isto é, assumindo, correspondentemente, uma importância negativa ou positiva no diagrama de barras. A importância de um atributo é baseada na sua correlação com as previsões feitas na vizinhança de um determinado valor desse atributo.
Esta importância depende do modelo selecionado e para o teste de regressão pudemos encontrar essas diferenças entre os diferentes modelos experimentados, sendo, no entanto, evidente o impacto dos atributos **player_count** e **player_big_radius** nos resultados das previsões.

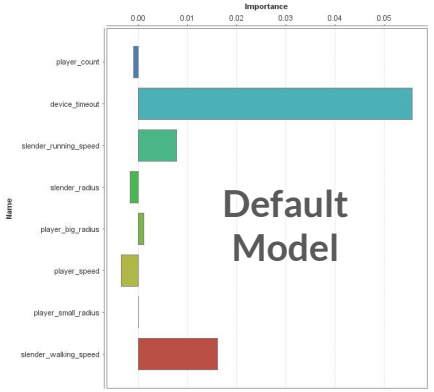
Análise dos dados com RapidMiner:

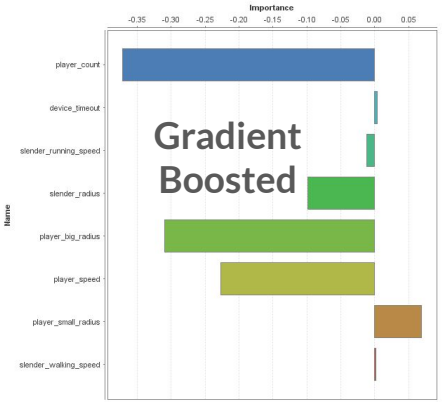
Segue-se uma visão geral da importância de cada atributo para os diferentes modelos no teste de regressão.











Conclusões:

RapidMiner foi bastante útil para analisarmos as previsões que são feitas em relação ao estado final do jogo e tempo de execução do mesmo, usando diferentes modelos, a partir dos dados obtidos de inúmeras simulações executadas com o nosso projeto de interação entre agentes.

Inicialmente corríamos o programa com valores base para cada parâmetro e este valores levavam várias vezes à vitória dos jogadores. No entanto, ao variarmos estes parâmetros e ao analisar os dados nesta plataforma reparamos que são muitos mais os casos de derrota.

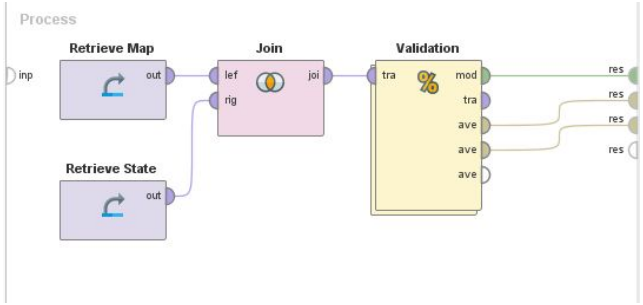
Também, através da análise estatística e da análise das simulações, usando os diferentes modelos, a importância dada aos atributos (nossas variáveis independentes) permitiu averiguar o impacto que o número inicial de jogadores tem no resultado final. De facto, inicialmente pensávamos que a velocidade do Slender seria o mais determinante, o que não se revelou.

De referir ainda, de um modo geral, todos os modelos testados e falados nesta apresentação revelaram uma ótima precisão, entre os 80% e 90%, o que confirma a não aleatoriedade do nosso trabalho.

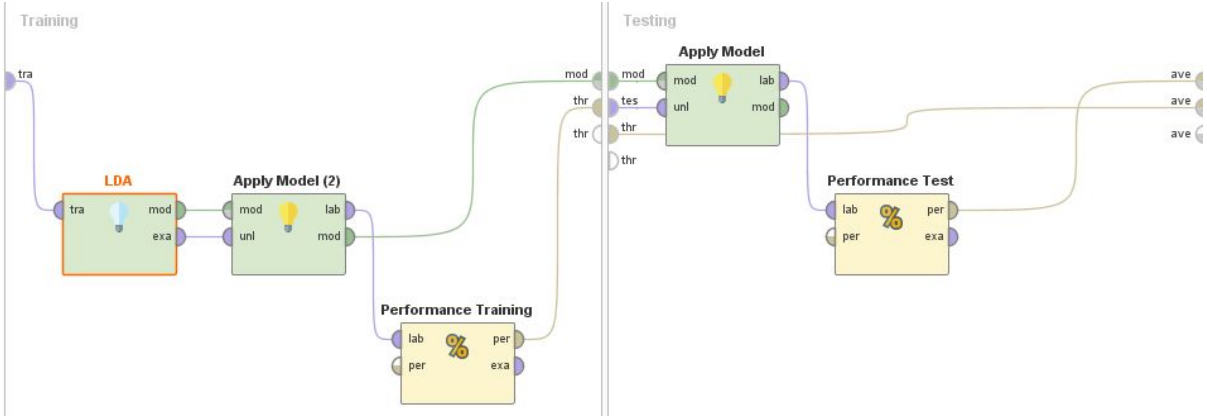
Informação adicional

Processos RapidMiner:

Teste de classificação - **LDA**



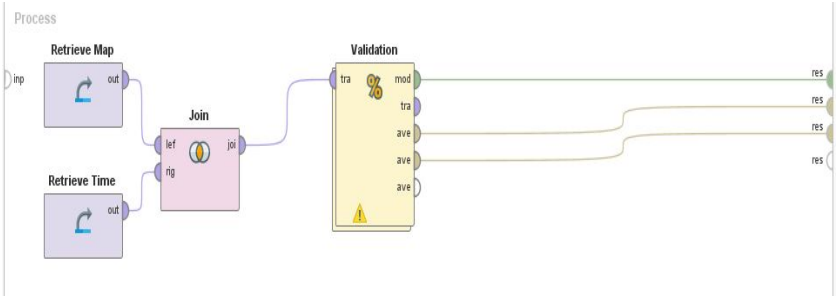
Esta é a estrutura de um processo que usa o modelo LDA. Para os restantes modelos que experimentamos, a estrutura é a mesma, apenas mudamos o operador “Predictive” de “Modeling” para corresponder ao algoritmos desejado.



Os dois Retrieve Map são os dados usados, sendo que um contém os valores das variáveis independentes e o outro das dependentes, daí fazermos join.

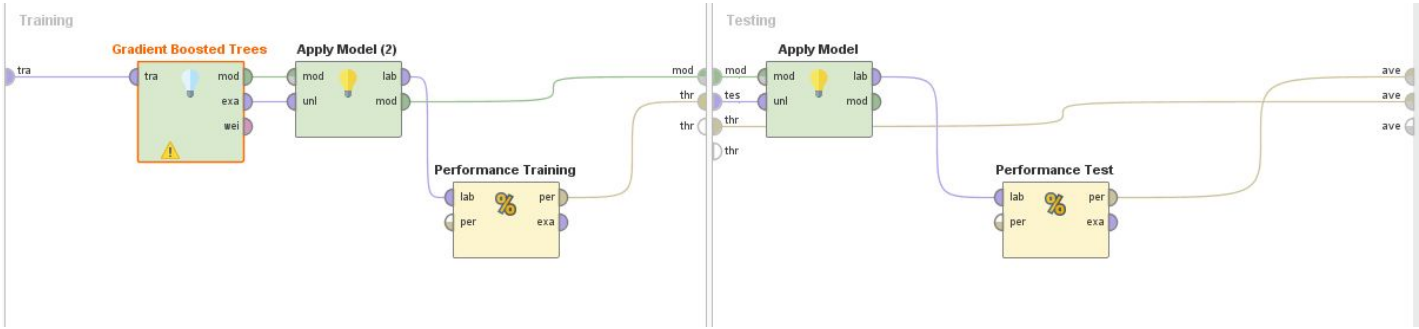
Processos RapidMiner:

Teste de regressão - **Gradient Boosted Trees**



Esta é a estrutura de um processo que usa o modelo Gradient Boosted Trees . Para os restantes modelos que experimentamos, a estrutura é a mesma, apenas mudamos o operador “Predictive” de “Modeling” para corresponder ao algoritmos desejado.

Os dois Retrieve Map são os dados usados, sendo que um contém os valores das variáveis independentes e o outro das dependentes, daí fazermos join.



Processos RapidMiner:

Acrescentamos o operador “Explain Predictions” de “Scoring” para obtermos o diagrama de barras com a informação da importância dos atributos, usando as configurações apresentadas à direita.

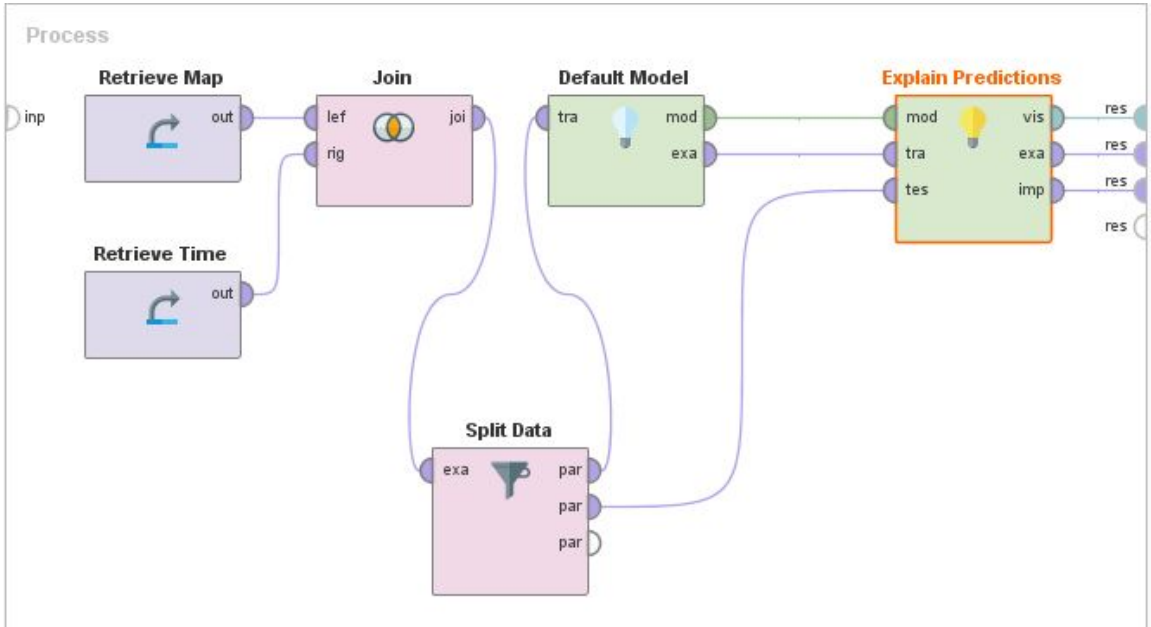


Chart style:

Bars

Group-By Column:

Name

Legend Column:

None

Value Column:

Importance

Absolute values

Aggregation:

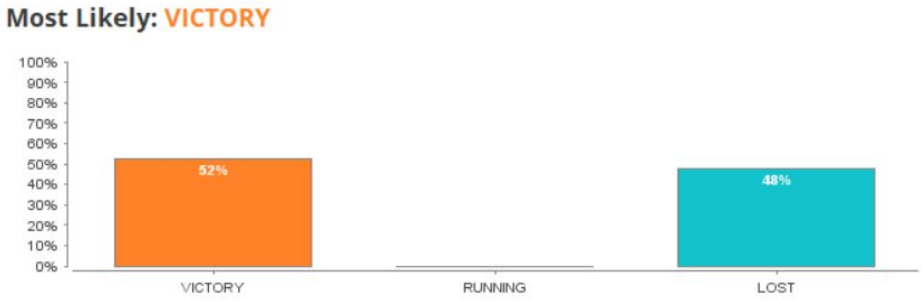
average

Use Only Distinct

Rotate labels

horizontal

Resultados de outras experiências interessantes:



No teste de classificação, ao usarmos o modelo Decision Tree, inesperadamente obtivemos que o mais provável era os jogadores ganharem. No entanto, o modelo não é muito confiável (confiança de 52.45%). O fator que mais suporta esta conclusão é o **player_count** (número inicial de jogadores). 84.11% de todas as previsões feitas são corretas e quando o modelo afirma que é vitória (prevê VICTORY), cobre 72.90% dos casos em que tal se verifica, tendo uma precisão de 54.77% para estas previsões.

classification_error: 15.89%

	true LOST	true RUNNING	true VICTORY	class precision
pred. LOST	1865	11	123	93.30%
pred. RUNNING	0	3	3	50.00%
pred. VICTORY	279	1	339	54.77%
class recall	86.99%	20.00%	72.90%	

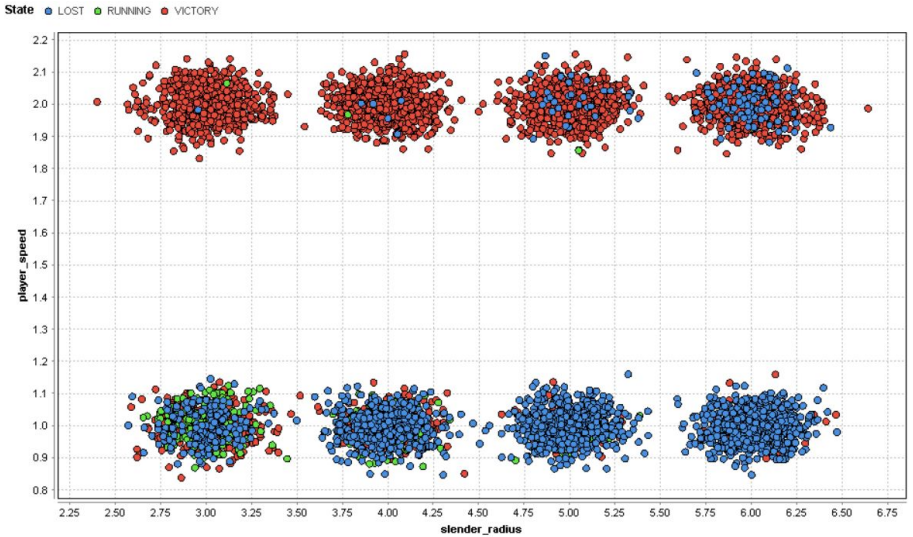
Resultados de outras experiências interessantes:

Experimentamos diferentes valores para os parâmetros, no entanto um pouco irreais no contexto do problema, visto que há casos em que o número inicial de jogadores ultrapassa o de dispositivos, 8, a velocidade do jogador chega a ultrapassar em 0.8 unidades a do Slender e os raios de visão dos jogadores são muito abrangentes. No entanto, não subimos só os valores dos atributos dos jogadores, as características do Slender também foram aumentadas, portanto o jogo não foi só facilitado para os jogadores.
Segue-se um excerto do nosso batch file com os períodos de valores a atribuir a cada parâmetro.

```
<parameter name="device_timeout" type="constant"
    constant_type="int" value="25"></parameter>
<parameter name="player_count" type="number" number_type="int"
    start="8" end="12" step="1">
    <parameter name="slender_running_speed" type="number"
        number_type="double" start="1.1" end="1.3" step="0.1">
        <parameter name="slender_radius" type="number" number_type="int"
            start="3" end="6" step="1">
            <parameter name="player_big_radius" type="number"
                number_type="int" start="5" end="9" step="1">
                <parameter name="player_speed" type="number" number_type="int"
                    start="1" end="2" step="1">
                    <parameter name="player_small_radius" type="number"
                        number_type="int" start="5" end="7" step="1">
                        <parameter name="slender_walking_speed" type="number"
                            number_type="double" start="1" end="2" step="0.5"></parameter>
```

Resultados de outras experiências interessantes:

Para esta nova experiências os **resultados foram significativamente diferentes,** registando-se **3242** casos de **VICTORY**, **1916** de **LOST** e **242** de **RUNNING**. Já a nível de tempo de execução de uma simulação, a **média** foi **334.847 ticks**, com um **desvio padrão de 447.900**.



No gráfico ao lado é possível ver o impacto da velocidade dos jogadores no resultado final, sendo que é quase certa a vitória quanto maior a velocidade, havendo ainda a possibilidade de derrota com o aumento do raio de visão do Slender.

Resultados de outras experiências interessantes:

Testes foram feitos com diferentes modelos, sendo que o melhor foi o **Random Forest**, com uma **precisão de 85.15%.** Este está confiante (em 95%) que será vitória dos jogadores. Desta vez, a importância dos atributos foi bastante diferente, sendo que o de maior impacto foi a **velocidade dos jogadores**, e ao contrário do visto anteriormente, o **player_count** não apresenta uma importância significativa para a previsão do estado final nem é um fator benéfico para a vitória:

Outras observações:

Para melhores resultados e uma análise mais rica, necessitaríamos de uma amostra maior de dados. Contudo, tal não foi possível pois o Repast Simphony não foi capaz de executar o número de simulações desejado.

No início do trabalho foi usado o AutoModel o que ajudou a focarmo-nos nos processos que seriam mais interessantes e relevantes para o nosso trabalho.