

# **RI : Jogos**

**João Vasconcelos  
Lucas Santana  
Rodrigo Carneiro**

# Domínio

Páginas a serem crawleadas (até o momento):

- [Steam](#)
- [Green Gaming](#)
- [Gamers Gate](#)
- [One Play](#)
- [GoG](#)
- [Humble Bundle](#)
- [Origin](#)
- [Nuvvem](#)
- [Itch](#)
- [Uplay](#)

# Crawler

- Heurística:
  - Para alguns sites, a url muda quando falamos das páginas de jogos (ex: [www.somedomain.com/games/id](http://www.somedomain.com/games/id))
  - Para esses sites, eu priorizei quando aparece a regex 'games', atribuindo peso 1 para esses links e 0 para os demais
  - Dessa forma, primeiro irei crawlear as páginas específicas de jogos, além de aumentar a frequência destas nas 1000 páginas.
  - Utilizei uma PriorityQueue para priorizar as páginas de maior peso

# Crawler

- Possíveis melhorias:
  - Enquanto parsear a procura de links, procurar também tags importantes como `<div class = 'classe_de_jogos'>`
  - Dessa forma é possível prever para todos os sites, não se restringindo apenas a URL
  - Problema: custo de parsear novamente em algumas situações

# Crawler (fases)

- Visito a página inicial de cada site (basic\_url) e, a partir dela, extraio o robots.txt
- Para utilizar o robots, importei um módulo
  - `from urllib.robotparser import RobotFileParser`
  - Sem esse módulo não haveria tanta precisão na hora de buscar as páginas permitidas pois o robots não está formatado em regex
  - Mas é possível fazer sem!!
- Com a página inicial e o robots, buscamos em todas as âncoras por hrefs que sejam relevantes
  - Evitar buscar páginas repetidas (set) e links de fora do domínio (facebook, twitter, etc)

# Crawler (fases)

- Após parsear a procura dos hrefs, adicionamos todos aqueles relevantes em uma PriorityQueue da forma (-peso, url)
  - (-peso) pois a PQ implementada é uma minHeap

# Crawler (basic\_crawler)

- Classe “pai” de todos os crawlers
- Define métodos básicos de busca e procura de páginas
- Métodos abstratos a serem implementados por cada crawler:
  - `downloadPage`: se difere por crawler pelo nome a ser baixado
  - `getRank`: método para estimar a relevância de cada página
  - `cleanUrl`: alguns sites vem com url “sujas” (mesmas páginas com url diferentes). Esse método serve para “unificar” essas páginas
  - `fixUrl`: alguns sites trabalham com urls inteiras no href, outros apenas com o path. Esse método conserta essas url para por na heap
  - `checkRegex`: método para decidir se uma página deve ser posta na heap ou não. Caso seja de outro domínio, ou seja uma página inútil (um dos sites colocavam imagens nos hrefs. Esse método filtraria isso)

# Problemas (Green Gaming)

robots.txt aparentemente está barrando todos os agentes por causa do Disallow selecionado:

```
Sitemap: https://www.greenmangaming.com/Sitemap.xml
User-agent: *
```

```
Disallow: /search/
Disallow: /my-account/
Disallow: /profile/
Disallow: /title-no-longer-available/
Disallow: /*?
Disallow: /game-unavailable/
Disallow: /your-cart---order/
Disallow: /es/blog/*
Disallow: /de/blog/*
Disallow: /pt/blog/*
Disallow: *product.Url*
Disallow: *p.pageUrl*
Disallow: /gsdrhtfdguf/*
Disallow: /new-games/*
```



# Problemas (OnePlay)

Mesmo caso anterior, mas esse deixou explícito que ninguém pode crawlear, exceto os que ele especificou

```
# Block all other spiders  
User-agent: *  
Disallow: /
```

# Problema (Humble Bundle)

Não consigo pegar nenhuma página relevante pois os links das páginas de jogos não foram “expostos” em tempo de request. Após acessarmos o site, o JavaScript povoa o HTML com os links. Dessa forma eu não consigo pegar nada interessante:

```
https://jobs.humblebundle.com  
/terms  
/privacy  
/developer  
/publishing  
https://www.facebook.com/humblebundle/  
https://twitter.com/humble  
https://plus.google.com/+HumbleBundle  
https://www.instagram.com/humblebundle  
http://blog.humblebundle.com/  
https://www.humblebundle.com/monthly?hmb_campaign=humble_monthly_aler
```

# Problema (Uplay)

- Excesso de imagens nas href
  - Criei uma regra de exceção para elas
- O crawler não conseguiu pegar muitas páginas

# Problemas (Origin)

- Problema do JavaScript não povoar as páginas antes de parseá-las

```
xlarge:2600},STORE:{xsmall:600,medium:1150,large:1600},PDP:{medium:1300,large:1500},HOME:{xsmall:600,medium:1150,large:1600},GAMELIBRARY:{small:900,medium:1300,large:1800}});\\n// # sourceMappingURL=true'},1641:function(e,t,a){a(41)(a(1642))},1642:function(e,t){e.exports='!function(){\"undefined\"==typeof OriginKernel&&(OriginKernel={}),\"undefined\"==typeof OriginKernel.configs&&(OriginKernel.configs={}),OriginKernel.configs[\"dist/configs/components-config.json\"]={settings:{partnerId:\"origin\",storeId:\"origin-store\",cartNameWeb:\"store-cart-purchase-{timestamp}\",cartNameClient:\"store-cart-purchase-client\",invoiceSourceBase:\"ORIGIN-STORE\"},hostname:{base:\"origin.com\",basedata:\"https://{env}data1.{base}/{cmsstage}\"},checkout:{paymentProviderRedirect:\"/views/checkout.html\",loading:\"/views/checkoutLoading.html\",geoipCountryOverrides:{UA:\"UA\"}},urls:{basedataTEST:\"{basedata}/somecomponentsdataurl\",homeStory:\"views/{sectionType}/{feedType}.html\",youtubeEmbedUrl:\"//www.youtube.com/embed/{videoId}?wmode=opaque&rel=0&enablejsapi=1&iv_load_policy=3&showinfo=0&modestbranding=1&hl={hl}\",achievementTileImageUrl:\"http://static.cdn.ea.com/ebisu/u/f/achievements/{achievementSetId}/images/en_US/overview_{achievementSetIdMD5}.jpg\",epilepsyWarningUrl:\"http://akamai.cdn.ea.com/edownloads/u/f/manuals/EPILEPSY/{locale}/GENERIC_EPI_{lang}.png\",greekEpilepsyWarningUrl:\"http://akamai.cdn.ea.com/edownloads/u/f/manuals/EPILEPSY/gr_GK/GENERIC_EPI_EL.png\",
```

# Crawler (must do)

- Utilizar um classificador para dizer se uma página é relevante
- Acima do ponto anterior, estimar a melhoria do uso de uma heurística nos crawlers
- Buscar mais domínios:
  - <https://gamejolt.com>
- Problemas com Javascript:
  - <https://stackoverflow.com/questions/8049520/web-scraping-javascript-page-with-python>
  - <https://medium.com/@hoppy/how-to-test-or-scrape-javascript-rendered-websites-with-python-selenium-a-beginner-step-by-c137892216aa>

# Crawler (to do)

- Melhorar a forma de estimar pesos para as páginas baixadas (a ideia de parsear em procura de elementos será o próximo passo)
- Pensar no extrator
- Trabalhar na modularização do código (já está, mas posso melhorar)
  - Formalizar as classes e métodos abstratos
- Pensar em como armazenar as páginas a fim de não ocupar espaço

Classificador

# Pré-processamento

- Remoção do html das páginas
- Remoção de pontos, vírgulas, números e caracteres especiais
- Utilização de bag of words
  - Contagem e frequência (stemming e stop words)



# Pré-processamento: Dificuldades

- Um dos sites continha sempre o mesmo html →Direct2drive
- Outro site baixava sempre a mesma página →Origin
- Páginas nem sempre vinham codificadas em UTF-8

# Classificação

- Scikit-learn
- Funções utilizadas:
  - Naives Bayes -> Gaussian NB
  - SVM -> SVC
  - Logistic Regression -> LogisticRegression
  - Decision Tree -> DecisionTreeClassifier
  - Multilayer Perceptron -> MLPClassifier
- Utilização de validação cruzada estratificada

A	Accuracy	Recall	Precision	Train time
tokenTfidf	0.35	0	0	0.57
tokenTfidf	0.90	0.90	0.90	0.57
tokenTfidf	0.55	0.60	0.55	0.58
tokenTfidf	0.90	1	0.83	0.57
tokenTfidf	0.55	0.80	0.53	0.57
tokenTfidf	0.55	1	0.53	0.57
tokenTfidf	0.55	1	0.53	0.56
tokenTfidf	0.80	0.90	0.75	0.57
tokenTfidf	0.85	0.70	1	0.58
tokenTfidf	0.65	0.80	0.62	0.56

# Classificação

- Cálculo de desempenho
  - Acurácia
  - Recall
  - Precisão
  - Tempo de treinamento
- Utilização de 9 bag of words diferentes
  - Contagem e frequência
  - Stemming, stop words, information gain

# Information gain

- Utilização da função `mutual_info_classif` do scikit-learn
- Melhores palavras são as que falam dos requisitos do sistema

Information_gain
gb
processor
geforce
requirements
intel
recommended
radeon
amd
directx
graphics
ram
gtx
nvidia
minimum
publisher
electricity
windows
itch
memory

# Desempenho por classificador

# Naive bayes

A	Accuracy	Recall	Precision	Train time
tokenTfidf	0.60	1	0.56	0.05
token	0.70	0.60	0.75	0.06
stopwordsTfidf	0.60	1	0.56	0.09
stopwords	0.70	0.60	0.75	0.06
stopNstemTfi...	0.85	0.80	0.89	0.05
stopNstem	0.85	0.80	0.89	0.06
stemmingTfidf	0.85	0.80	0.89	0.06
stemming	0.85	0.80	0.89	0.05
info_gain	0.75	0.70	0.78	0.01

# SVM

A	Accuracy	Recall	Precision	Train time
tokenTfidf	0.90	1	0.83	0.57
stopwords	0.90	1	0.83	0.54
stopwordsT...	0.95	0.90	1	0.56
stemming	0.90	0.90	0.90	0.42
stemmingT...	0.90	0.90	0.90	0.54
stopNstem	0.90	0.90	0.90	0.48
info_gain	0.85	0.70	1	0.09
stopNstemT...	0.90	0.90	0.90	0.49
token	0.90	0.90	0.90	0.61

# Logistic Regression

A	Accuracy	Recall	Precision	Train time
tokenTfidf	0.90	0.90	0.90	0.03
stopwords	0.85	0.90	0.82	0.09
stopwordsT...	0.95	0.90	1	0.03
stemming	0.90	1	0.83	0.09
stemmingT...	0.95	0.90	1	0.03
stopNstem	0.90	1	0.83	0.08
info_gain	0.90	1	0.83	0.01
stopNstemT...	0.95	0.90	1	0.02
token	0.90	1	0.83	0.11



# Decision Tree

A	Accuracy	Recall	Precision	Train time
tokenTfidf	1	1	1	0.10
stopwords	0.90	1	0.83	0.09
stopwordsT...	1	1	1	0.10
stemming	1	1	1	0.07
stemmingT...	0.90	0.80	1	0.09
stopNstem	1	1	1	0.07
info_gain	0.85	0.80	0.89	0.02
stopNstemT...	0.90	0.90	0.90	0.10
token	0.85	0.80	0.89	0.10

# Multilayer Perceptron

A	Accuracy	Recall	Precision	Train time
tokenTfidf	0.90	1	0.83	14.81
stopwords	0.95	0.90	1	5.67
stopwordsT...	0.90	0.90	0.90	26.55
stemming	0.95	0.90	1	6.55
stemmingT...	0.85	0.80	0.89	17.52
stopNstem	0.95	0.90	1	8.82
info_gain	1	1	1	2.70
stopNstemT...	0.85	0.80	0.89	16.26
token	0.95	0.90	1	7.53

# Para o futuro

- Terminar de modularizar o código para o crawler poder classificar
- Definir uma estratégia para definir se uma página é positiva ou não utilizando os 5 classificadores
- É possível salvar os vectorizers e os classificadores para não ter que rodar eles sempre?

FIM