Projeto 1

Crawler, Classificador e Extrator

Tiago Moraes (tbm2)
Matheus Borba (mlbas)
Edjan (esvm)

Crawler

Tiago Moraes (tbm2)

1. Análise do *robots.txt*

- Utilizando a lib ***reppy***, conseguimos fazer um parse no *robots.txt*.
  - Baixamos o *robots.txt* e fazemos o parsing dele com a lib
  - Exporta uma função ***allowed -> True | False***

```
robots = Robots.fetch(f'{seed[1]}/robots.txt')
```

2. Processamento e verif. da url

- Processamento de **urls relativas → absolutas**
- Verifica se url é do mesmo domínio e chama o *allowed* do reppy

```python
def process_url(url, root):
    if (url.startswith('/')):
        return f'{root}{url}'

    return url
```

DESAFIO: Loop de query params foi resolvido colocando um limite no tamanho da url

```python
def check_url(url, root, robots):
    # Don't access Disallowed routes by robots.txt
    if (not robots.allowed(url, '*')):
        return False

    # Remove long urls (avoid qp loops)
    if (len(url) > 160):
        return False

    if (url.startswith('/') or
        url.startswith(root) or
        url == root):
        # Means url is usable
        return True

    return False
```

3. Análise do *content-type*

- Fazemos o request do ***header*** antecipadamente, evitando o download desnecessário de páginas que não são ***text/html***

```
# Check headers if response is html text
try:
    hr = requests.head(start)

    if (not 'content-type' in hr.headers):
        return urls

    if (not hr.headers['content-type'].startswith('text/html')):
        return urls
except:
    return urls
```

4. BFS vs Heurística v2

Função para pegar todas as âncoras na página

- Abordagem **BFS:** Todos os *hrefs* da página
- Abordagem **Heurística(v2):** ***(fila prioritária)***

```
WANTED_TERMS = ['vinho', 'produto', 'product',
'tinto', 'branco', 'espumante', 'rose', 'rosado',
'cabernet', 'malbec', 'shiraz', 'primitivo',
'merlot', 'garnacha', 'pinot', 'carmenere', 'brut',
'mosacatel', 'tempranillo', 'reserva','chardonnay',
'riesling', 'seco']

NOT_WANTED_TERMS = ['kits?', 'saca', 'ta(ç|c)a',
'contato', 'sobre', 'cart', 'carrinho', 'termo',
'pol(í|i)tica', 'acess(ó|o)rios', 'blog', 'central']
```

```
def classify_anchor(anchor):
    for term in NOT_WANTED_TERMS:
        if anchor.text and re.match(f'.*{term}.*', anchor.text.lower()):
            return -1

        if anchor['href'] and re.match(f'.*{term}.*', anchor['href'].lower()):
            return -1

    for term in WANTED_TERMS:
        if anchor.text and re.match(f'.*{term}.*', anchor.text.lower()):
            return 1

        if anchor['href'] and re.match(f'.*{term}.*', anchor['href'].lower()):
            return 1

    return 0
```

Abordagem

Heurística v2

- Acessa primeiro a **fila prioritária.**
- Se ela se esvaziar, acessa a **fila regular.**
- As âncoras com *NOT_WANTED_TERMS*, são **descartadas**.

obs.: v1 da heurística se baseava apenas em só acessar âncoras com termos desejados, mas em diversos sites, essa abordagem fazia com que o crawler entrasse em "estagnação", conseguindo *crawlear* um número muito pequeno de páginas. A solução foi criar um sistema de priorização para as âncoras desejadas, mas não excluir as que eram regulares (nem desejadas, nem indesejadas).

PROBLEMA: v2 tinha resultado muito semelhante (senão pior) ao BFS

5. Heurística v3

Abordagem **Heurística(v3):** *(analisa **anchor.text** e **anchor.href**)*

```
WANTED_PATHS = ['/p/', '/produto', '/product', '/prod',
'/item', '/vinho']

WANTED_TERMS = ['tinto', 'branco', 'espumante', 'rose',
'rosado', 'cabernet', 'malbec', 'shiraz', 'primitivo',
'merlot', 'garnacha', 'pinot', 'carmenere', 'brut',
'mosacatel', 'tempranillo', 'reserva','chardonnay',
'riesling', 'seco', 'ver', 'mais']

NOT_WANTED_PATHS = ['/kits?', '/carrinho', '/cart',
'/sacola', '/add/']

NOT_WANTED_TERMS = ['kits?', 'saca', 'ta(ç|c)a',
'contato', 'sobre', 'carrinho', 'termo', 'sacola',
'pol(í|i)tica', 'acess(ó|o)rios', 'blog', 'central']
```

```
def classify_anchor(anchor):
    weight = 0

    for term in WANTED_PATHS:
        if anchor['href'] and re.match(f'.*{term}.*', anchor['href'].lower()):
            weight = weight + 20

    for term in WANTED_TERMS:
        if anchor.text and re.match(f'.*{term}.*', anchor.text.lower()):
            weight = weight + 10

    for term in NOT_WANTED_PATHS:
        if anchor['href'] and re.match(f'.*{term}.*', anchor['href'].lower()):
            weight = weight - 20

    for term in NOT_WANTED_TERMS:
        if anchor.text and re.match(f'.*{term}.*', anchor.text.lower()):
            weight = weight - 10

    return weight
```

Abordagem

Heurística v3

if in ***WANTED_PATHS*** = **+20**

if in ***NOT_WANTED_PATHS*** = **-20**

if in ***WANTED_TERMS*** = **+10**

if in ***NOT_WANTED_PATHS*** = **-10**

obs.: Adição do ***"Crawl Delay"*** para não sobrecarregar os sites

Outros desafios

DESAFIO: Crawler navegava pelas *seeds* de forma linear (1 site por vês), o que deixava o crawler lento. Usamos **multiprocessing**, com um **Pool(10)**, para que os sites fossem *crawleados* em paralelo.

```
def main():
    pool = Pool(10)
    pool.map(bfs_crawl, seeds)
    pool.close()
    pool.join()
```

⬇

Crawling Time ≅ 55 min

(p/ 10 seeds)

DESAFIO: Fluxo infinito de iteração pelos query params

```
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1984', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?harmonizacao=1513', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?produtor=983', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?produtor=960', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1910', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?produtor=630', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?harmonizacao=1512', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?harmonizacao=1529', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?regiao=1750', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?harmonizacao=1530', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?produtor=629', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1811', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1966', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1967', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?produtor=749', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?harmonizacao=1511', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?tipo=1181', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1926', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1762', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1911', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1968', 30)
Entered ('https://www.viavini.com.br/vinhos/tipo/vinho-branco.html?uva=1971', 30)
```

Harvest Ratios

BFS

classificador	total
NAIVE_BAYES	0,518
DECISION_TREE	0,427
SVM	0,298
LOGISTIC_REGRESSION	0,397
MULTILAYER_PERCEPTRON	**0,395**

Heurística v2

classificador	total
NAIVE_BAYES	0,328
DECISION_TREE	0,426
SVM	0,314
LOGISTIC_REGRESSION	0,414
MULTILAYER_PERCEPTRON	**0,416**

Heurística v3

classificador	total
NAIVE_BAYES	0,418
DECISION_TREE	0,508
SVM	0,405
LOGISTIC_REGRESSION	0,482
MULTILAYER_PERCEPTRON	**0,488**

De fato, foram melhores do que BFS e v2

Harvest Ratio

BFS

classificador	viavini	wine	evino	superadega	grandcru	adegamais	divinho	divvino	vivavinho	mistral	total
NAIVE_BAYES	0,152	0,740	0,313	0,636	0,754	0,605	0,274	0,802	0,313	0,600	0,518
DECISION_TREE	0,153	0,602	0,212	0,296	0,632	0,408	0,179	0,748	0,592	0,454	0,427
SVM	0,120	0,565	0,035	0,276	0,685	0,403	0,177	0,032	0,246	0,442	0,298
LOGISTIC_REGRESSION	0,120	0,616	0,129	0,372	0,715	0,416	0,185	0,762	0,194	0,468	0,397
MULTILAYER_PERCEPTRON	0,120	0,606	0,196	0,350	0,698	0,409	0,177	0,745	0,205	0,453	**0,395**

Harvest Ratio

Heurística v2

classificador	viavini	wine	evino	superadega	grandcru	adegamais	divinho	divvino	vivavinho	mistral	total
NAIVE_BAYES	0,121	0,669	0,050	0,117	0,879	0,299	0,137	0,004	0,264	0,676	0,328
DECISION_TREE	0,124	0,749	0,146	0,193	0,928	0,427	0,137	0,008	0,140	0,645	0,426
SVM	0,121	0,633	0,012	0,206	0,851	0,298	0,137	0,007	0,253	0,620	0,314
LOGISTIC_REGRESSION	0,124	0,665	0,090	0,184	0,881	0,299	0,137	0,916	0,204	0,644	0,414
MULTILAYER_PERCEPTRON	0,121	0,653	0,134	0,211	0,853	0,299	0,135	0,915	0,212	0,632	**0,416**

Harvest Ratio

Heurística v3

classificador	viavini	wine	evino	superadega	grandcru	adegamais	divinho	divvino	vivavinho	mistral	total
NAIVE_BAYES	0,000	0,778	0,188	0,487	0,868	0,753	0,131	0,008	0,184	0,777	0,418
DECISION_TREE	0,019	0,816	0,319	0,504	0,849	0,795	0,131	0,794	0,111	0,758	0,508
SVM	0,000	0,755	0,073	0,545	0,847	0,743	0,131	0,002	0,182	0,759	0,405
LOGISTIC_REGRESSION	0,000	0,766	0,233	0,482	0,854	0,753	0,128	0,703	0,139	0,772	0,482
MULTILAYER_PERCEPTRON	0,000	0,779	0,291	0,496	0,851	0,752	0,126	0,686	0,146	0,759	**0,488**

Classificação

Matheus Borba (mlbas)

Abordagem

- Processamento *(nltk)*
  - Bag of words
    - Frequência, Contagem
    - Stemming, stopwords
  - Feature Extractor *(scale vector)*
    - Most Frequent Words
    - **Doc Frequency Difference**
    - Plain Frequency Difference
    - Mixed Frequency Difference
- Treinamento *(Scikit-learn)*
  - Naive Bayes *(Gaussian NB)*
  - Decision Tree
  - SVM
  - Logistic Regression
  - Multilayer Perceptron

Feature Extractor

Pós-processamento do vocabulário com/sem stopwords.

- Doc and Mixed Frequency Difference não sofreram tantas alterações.
- Quantidade de palavras extraídas igual a 50.*

```
wout/MostFrequentWordsExtractor: ['de', 'vinho', 'mai', 'do', 'r$', 'ver', 'comprar', 'tinto', 'com', 'sauvignon', 'cabernet', 'para', 'por', 'espum', 'todo', 'da', 'branco', 'malbec', 'em', 'até', 'wine', 'saiba', 'um', 'pinot', 'paí',
'que', 'uva', '750ml', 'não', 'seu', 'chardonnay', 'tipo', 'ao', 'como', 'os', 'ponto', 'na', 'catena', 'grand', 'política', 'vineyard', 'no', 'produto', 'rosé', 'noir', 'reserva', 'del', 'kit', 'blanc', 'chile']
w/MostFrequentWordsExtractor: ['vinho', 'mai', 'r$', 'ver', 'comprar', 'tinto', 'sauvignon', 'cabernet', 'espum', 'todo', 'branco', 'malbec', 'wine', 'saiba', 'pinot', 'paí', 'uva', '750ml', 'chardonnay', 'tipo', 'ponto', 'catena',
'grand', 'política', 'vineyard', 'produto', 'rosé', 'noir', 'reserva', 'del', 'kit', 'blanc', 'chile', 'zapata', 'syrah', 'visualização', 'rápida', 'itália', 'espanha', 'di', 'loja', 'ano', 'pontuado', 'privacidad', 'carrinho',
'frança', 'preço', 'brasil', 'fruta', 'sul']

wout/DocFrequencyDifferenceExtractor: ['teor', 'alcoólico', 'temperatura', 'harmonização', 'mese', 'relacionado', 'ordenar', 'serviço', 'barrica', 'filtrar', 'guarda', 'fruta', 'carvalho', 'aroma', 'ficha', 'prazo', '750ml', 'final',
'visual', 'exclusivo', 'técnica', 'safra', 'boca', 'est', 'disponibilidad', 'volum', 'nota', 'vermelho', 'classificação', 'melhor', 'tanino', 'produto', 'vinícola', 'nariz', 'francê', 'encontrado', 'carn', 'vermelha', 'olfativo', 'a-z',
'região', 'frete', 'toqu', '18°c', 'baixa', 'desconto', 'seco', 'queijo', 'estoqu', 'juro']
w/DocFrequencyDifferenceExtractor: ['teor', 'alcoólico', 'temperatura', 'harmonização', 'mese', 'relacionado', 'ordenar', 'serviço', 'barrica', 'filtrar', 'guarda', 'fruta', 'carvalho', 'aroma', 'ficha', 'prazo', '750ml', 'final',
'visual', 'exclusivo', 'técnica', 'safra', 'boca', 'est', 'disponibilidad', 'volum', 'nota', 'vermelho', 'classificação', 'melhor', 'tanino', 'produto', 'vinícola', 'nariz', 'francê', 'encontrado', 'carn', 'vermelha', 'olfativo', 'a-z',
'região', 'frete', 'toqu', '18°c', 'baixa', 'desconto', 'seco', 'queijo', 'estoqu', 'juro']

wout/PlainFrequencyDifferenceExtractor: ['comprar', 'de', 'vinho', 'r$', 'visualização', 'rápida', 'quantidad', 'por', 'tinto', 'mai', 'vistamar', 'do', 'artigo', 'ver', 'até', 'wishlist', 'ao', 'sauvignon', 'cabernet', 'adicionar',
'espum', 'os', 'kit', 'assin', 'branco', 'carrinho', 'yith_wcwl_add_to_wishlist', 'que', 'para', 'não', 'carmen', 'ou', 'chile', 'niepoort', 'preço', '750ml', 'rosé', 'esporão', 'chardonnay', 'limpar', 'são', 'produto', 'você',
'reserva', 'vinícola', 'as', 'rose', 'est', 'taça', 'melhor']
w/PlainFrequencyDifferenceExtractor: ['comprar', 'vinho', 'r$', 'visualização', 'rápida', 'quantidad', 'tinto', 'mai', 'vistamar', 'artigo', 'ver', 'wishlist', 'sauvignon', 'cabernet', 'adicionar', 'espum', 'kit', 'assin', 'branco',
'carrinho', 'yith_wcwl_add_to_wishlist', 'carmen', 'chile', 'niepoort', 'preço', '750ml', 'rosé', 'esporão', 'chardonnay', 'limpar', 'produto', 'reserva', 'vinícola', 'rose', 'est', 'taça', 'melhor', 'região', 'obrigatório', 'esgotado',
'la', 'campo', 'ordenar', 'frança', 'harmonização', 'safra', 'barrica', 'brisa', 'spiegelau', '---']

wout/MixedFrequencyDifferenceExtractor: ['comprar', 'visualização', 'teor', 'harmonização', 'artigo', 'por', 'até', '750ml', 'mese', 'ordenar', 'temperatura', 'barrica', 'vinho', 'kit', 'ver', 'alcoólico', 'de', 'quantidad', 'est',
'produto', 'filtrar', 'vinícola', 'guarda', 'melhor', 'ficha', 'carvalho', 'safra', 'limpar', 'região', 'exclusivo', 'relacionado', 'serviço', 'tinto', 'técnica', 'esgotado', 'rápida', 'seco', 'preço', 'cabernet', 'final', 'branco',
'classificação', 'resultado', 'adicionar', 'são', 'espum', 'ou', 'desconto', 'sem', 'comentário']
w/MixedFrequencyDifferenceExtractor: ['comprar', 'visualização', 'teor', 'harmonização', 'artigo', '750ml', 'mese', 'ordenar', 'temperatura', 'barrica', 'vinho', 'kit', 'ver', 'alcoólico', 'quantidad', 'est', 'produto', 'filtrar',
'vinícola', 'guarda', 'melhor', 'ficha', 'carvalho', 'safra', 'limpar', 'região', 'exclusivo', 'relacionado', 'serviço', 'tinto', 'técnica', 'esgotado', 'rápida', 'seco', 'preço', 'cabernet', 'final', 'branco', 'classificação',
'resultado', 'adicionar', 'espum', 'desconto', 'comentário', 'sauvignon', 'la', 'mai', 'vinhedo', 'fresco', 'carmen']
```

Generic Classifier

```python
class DocumentClassifier(Classifier):
    def __init__(self, feature_extractor: FeatureExtractor):
        super().__init__(feature_extractor)
        self.trained = False

    def train(self, docs: [Document], classifier_type: ClassifierType, train_size: float = 1.0, verbose: bool =
        start_time = dt.now()

        positive_docs = list(filter(lambda doc: doc.is_instance == DocumentClass.INSTANCE, docs))
        negative_docs = list(filter(lambda doc: not doc.is_instance == DocumentClass.INSTANCE, docs))

        total_positive = int(len(positive_docs)*train_size)
        total_negative = int(len(negative_docs)*train_size)

        train_docs = positive_docs[:total_positive] + negative_docs[:total_negative]
        train_docs = shuffle(train_docs)

        test_docs = positive_docs[total_positive:] + negative_docs[total_negative:]
        test_docs = shuffle(test_docs)

        self.features = self.feature_extractor.get_feature_words(num_features=50)

        classifiers = {
            classifier_type.NAIVE_BAYES: GaussianNB(),
            classifier_type.DECISION_TREE: DecisionTreeClassifier(),
            classifier_type.SVM: SVC(),
            classifier_type.LOGISTIC_REGRESSION: LogisticRegression(),
            classifier_type.MULTILAYER_PERCEPTRON: MLPClassifier()
        }
        clf = classifiers[classifier_type]
```

```python
    classifiers = {
        classifier_type.NAIVE_BAYES: GaussianNB(),
        classifier_type.DECISION_TREE: DecisionTreeClassifier(),
        classifier_type.SVM: SVC(),
        classifier_type.LOGISTIC_REGRESSION: LogisticRegression(),
        classifier_type.MULTILAYER_PERCEPTRON: MLPClassifier()
    }
    clf = classifiers[classifier_type]
    # parameters = {'solver': ('adam', 'lbfgs', 'sgd'), 'activation': ('relu', 'identity', 'tanh
    # clf = GridSearchCV(clf, parameters, scoring='accuracy', verbose=1)

    self.clf = clf

    x, y, self.scaler = get_vectors_scaler(self.features, train_docs)

    self.clf.fit(x, y)

    if len(test_docs) > 0:
        final_preds = self._internal_predict(test_docs)
        final_preds = doc_class_to_int(final_preds)
        correct_preds = doc_class_to_int([test_doc.is_instance for test_doc in test_docs])

        if verbose:
            print_metrics(final_preds, correct_preds)

    if verbose:
        end_time = dt.now()
        train_duration = (end_time-start_time).total_seconds()
        print("Training took {} seconds".format(train_duration))
    self.trained = True
```

Treinamento - Naive Bayes(Gaussian NB)

	A	B	C	D	E	F	G
1	Stopwords Dropped	Feature Selector	Accuracy	Precision	Recall	F1-Measure	Training took(seconds)
2	True	Most Frequent Words	0.5901639344262295	0.5510204081632653	0.9	0.6835443037974683	0.200358
3	False	Most Frequent Words	0.6065573770491803	0.56	0.9333333333333333	0.7000000000000001	0.175127
4	True	Doc Frequency Difference	0.8688524590163934	0.8055555555555556	0.9666666666666667	0.8787878787878789	0.452479
5	False	Doc Frequency Difference	0.8688524590163934	0.8055555555555556	0.9666666666666667	0.8787878787878789	0.384467
6	True	Plain Frequency Difference	0.8524590163934426	0.7837837837837838	0.9666666666666667	0.8656716417910447	0.533508
7	False	Plain Frequency Difference	0.7377049180327869	0.6590909090909091	0.9666666666666667	0.7837837837837838	0.42628
8	True	Mixed Frequency Difference	0.8524590163934426	0.7837837837837838	0.9666666666666667	0.8656716417910447	0.829031
9	False	Mixed Frequency Difference	0.8524590163934426	0.7837837837837838	0.9666666666666667	0.8656716417910447	0.703819

Treinamento - Decision Tree

	A	B	C	D	E	F	G
1	Stopwords Dropped	Feature Selector	Accuracy	Precision	Recall	F1-Measure	Training took(seconds)
2	True	Most Frequent Words	0.8360655737704918	0.8125	0.8666666666666667	0.8387096774193549	0.223341
3	False	Most Frequent Words	0.7377049180327869	0.7058823529411765	0.8	0.7500000000000001	0.193823
4	True	Doc Frequency Difference	0.8852459016393442	0.8709677419354839	0.9	0.8852459016393444	0.525822
5	False	Doc Frequency Difference	0.9180327868852459	0.9032258064516129	0.9333333333333333	0.9180327868852459	0.327213
6	True	Plain Frequency Difference	0.8852459016393442	0.8484848484848485	0.9333333333333333	0.888888888888889	0.42303
7	False	Plain Frequency Difference	0.7377049180327869	0.71875	0.7666666666666667	0.7419354838709677	0.338326
8	True	Mixed Frequency Difference	0.9180327868852459	0.9032258064516129	0.9333333333333333	0.9180327868852459	0.773576
9	False	Mixed Frequency Difference	0.9016393442622951	0.9285714285714286	0.8666666666666667	0.896551724137931	0.639591

Treinamento - SVM

	A	B	C	D	E	F	G
1	Stopwords Dropped	Feature Selector	Accuracy	Precision	Recall	F1-Measure	Training took(seconds)
2	True	Most Frequent Words	0.5901639344262295	0.5675675675675675	0.7	0.626865671641791	0.210193
3	False	Most Frequent Words	0.7049180327868853	0.6428571428571429	0.9	0.75	0.171704
4	True	Doc Frequency Difference	0.8360655737704918	0.9545454545454546	0.7	0.8076923076923077	0.496865
5	False	Doc Frequency Difference	0.8360655737704918	0.9545454545454546	0.7	0.8076923076923077	0.339314
6	True	Plain Frequency Difference	0.7540983606557377	0.7142857142857143	0.8333333333333334	0.7692307692307692	0.395384
7	False	Plain Frequency Difference	0.6885245901639344	0.6410256410256411	0.8333333333333334	0.7246376811594204	0.33102
8	True	Mixed Frequency Difference	0.819672131147541	0.9523809523809523	0.6666666666666666	0.7843137254901961	0.775635
9	False	Mixed Frequency Difference	0.8360655737704918	0.9545454545454546	0.7	0.8076923076923077	0.6349

Treinamento - Logistic Regression

	A	B	C	D	E	F	G
1	Stopwords Dropped	Feature Selector	Accuracy	Precision	Recall	F1-Measure	Training took(seconds)
2	True	Most Frequent Words	0.7377049180327869	0.7058823529411765	0.8	0.7500000000000001	0.261649
3	False	Most Frequent Words	0.7213114754098361	0.7407407407407407	0.6666666666666666	0.7017543859649122	0.19425
4	True	Doc Frequency Difference	0.9180327868852459	0.8787878787878788	0.9666666666666667	0.9206349206349207	0.445906
5	False	Doc Frequency Difference	0.9180327868852459	0.8787878787878788	0.9666666666666667	0.9206349206349207	0.333979
6	True	Plain Frequency Difference	0.8852459016393442	0.8484848484848485	0.9333333333333333	0.888888888888889	0.401929
7	False	Plain Frequency Difference	0.8688524590163934	0.84375	0.9	0.870967741935484	0.338523
8	True	Mixed Frequency Difference	0.8852459016393442	0.8285714285714286	0.9666666666666667	0.8923076923076922	0.766476
9	False	Mixed Frequency Difference	0.9180327868852459	0.8787878787878788	0.9666666666666667	0.9206349206349207	0.639141

Treinamento - Multilayer Perceptron

	A	B	C	D	E	F	G
1	Stopwords Dropped	Feature Selector	Accuracy	Precision	Recall	F1-Measure	Training took(seconds)
2	True	Most Frequent Words	0.7868852459016393	0.717948717948718	0.9333333333333333	0.8115942028985509	2.189233
3	False	Most Frequent Words	0.819672131147541	0.7567567567567568	0.9333333333333333	0.835820895522388	0.961627
4	True	Doc Frequency Difference	0.9180327868852459	0.8787878787878788	0.9666666666666667	0.9206349206349207	2.412335
5	False	Doc Frequency Difference	0.9180327868852459	0.9032258064516129	0.9333333333333333	0.9180327868852459	0.825137
6	True	Plain Frequency Difference	0.8688524590163934	0.7894736842105263	1.0	0.8823529411764706	1.937562
7	False	Plain Frequency Difference	0.8688524590163934	0.8055555555555556	0.9666666666666667	0.8787878787878789	1.124622
8	True	Mixed Frequency Difference	0.9180327868852459	0.9032258064516129	0.9333333333333333	0.9180327868852459	2.834526
9	False	Mixed Frequency Difference	0.9508196721311475	0.9354838709677419	0.9666666666666667	0.9508196721311476	1.525286

Weighted Ensemble for Accuracy

```python
class AccuracyWeightedEnsemble:
    def __init__(self, classifiers: List[Classifier]):
        clfs = []
        for clf in classifiers:
            clfs.append({
                'acc': 0,
                'clf': clf,
            })
        self.clfs = clfs
        self.trained = False

    def train(self, docs: List['Document'], train_size: float = 1.0, verbose: bool = False):
        start_time = dt.now()
        positive_docs = list(filter(lambda doc: doc.is_instance == 1, docs))
        negative_docs = list(filter(lambda doc: not doc.is_instance == 1, docs))

        total_positive = int(len(positive_docs) * train_size)
        total_negative = int(len(negative_docs) * train_size)

        train_docs = positive_docs[:total_positive] + negative_docs[:total_negative]
        train_docs = shuffle(train_docs)

        test_docs = positive_docs[total_positive:] + negative_docs[total_negative:]
        test_docs = shuffle(test_docs)
```

```python
docs = np.array(docs)
labels = np.array(doc_class_to_int([doc.is_instance for doc in docs]))
kfold = StratifiedKFold(n_splits=10)
for clf in self.clfs:
    accs = []
    for train_index, test_index in kfold.split(docs, labels):
        clf['clf'].train(docs[train_index], train_size=train_size, verbose=verbose)
        preds = clf['clf'].predict(docs[test_index])
        y = labels[test_index]
        _, acc, _, _, _ = compute_metrics(preds, y)
        accs.append(acc)
    clf['acc'] = np.mean(accs)

sum_acc = 0.0
for clf in self.clfs:
    clf['clf'].train(train_docs, train_size=train_size, verbose=verbose)
    sum_acc += clf['acc']

self.total_acc = sum_acc

if len(test_docs) > 0:
    preds = self._internal_predict(test_docs)
    y = doc_class_to_int([doc.is_instance for doc in test_docs])

    if verbose:
        print_metrics(preds, y)

if verbose:
    end_time = dt.now()
    train_duration = (end_time - start_time).total_seconds()
    print("Training took {} seconds".format(train_duration))

self.trained = True
```

Extração

Abordagem

- Um extrator específico para cada site
- Um extrator geral
- A extração foi aplicada nas páginas que foram classificadas como positivas considerando cada um dos classificadores e também nas negativas, para extrair as métricas
- Utilizamos a lib BeautifulSoup para parsear html

Abordagem

```
def extract(domain, type, extract_function):
    classifier_index = 0
    extraction_results_metrics = []
    extraction_results = []
    negative_pages_extraction_results = []
    total_pages = 1001
    pages = [i for i in range(total_pages)]
    for classifier in classifiers:
        print(
            'running extraction from classifier {} for domain {}'.format(get_classifier_name(classifier_index), domain))
        total_pages_for_classifier = len(classifier[domain])
        total_pages_extracted_successfully = 0 # true positive
        total_negative_pages_extracted_succesfully = 0 # false negative
        positive_pages = classifier[domain]
        for index in positive_pages:
            try:
                html_file = open('../crawler/bfs_pages/{}/{}.html'.format(domain, index + 1))
                html_text = html_file.read()
                soup = BeautifulSoup(html_text, 'lxml')
                extract_result = extract_function(soup)
                extraction_results.append(extract_result)
                if extract_result['wine_type'] is not None and extract_result['wine_type'] != '':
                    total_pages_extracted_successfully += 1
            except:
                print('failed to open ../crawler/bfs_pages/{}/{}.html'.format(domain, index + 1))

        negative_pages = list(set(pages) - set(positive_pages))
        for index in negative_pages:
            try:
                html_file = open('../crawler/bfs_pages/{}/{}.html'.format(domain, index + 1))
                html_text = html_file.read()
                soup = BeautifulSoup(html_text, 'lxml')
                negative_pages_extract_result = extract_function(soup)
                negative_pages_extraction_results.append(negative_pages_extract_result)
                if negative_pages_extract_result['wine_type'] is not None and negative_pages_extract_result['wine_type'] != '':
                    total_negative_pages_extracted_succesfully += 1
```

Abordagem

```
            if negative_pages_extract_result['wine_type'] is not None and negative_pages_extract_result['wine_type'] != '':
                total_negative_pages_extracted_succesfully += 1
        except:
            print('failed to open ../crawler/bfs_pages/{}/{}.html'.format(domain, index + 1))

    false_positive = total_pages_for_classifier - total_pages_extracted_successfully
    true_negative = total_pages - total_negative_pages_extracted_succesfully
    precision = (total_pages_extracted_successfully / total_pages_for_classifier) * 100
    recall = 0
    if total_pages_extracted_successfully + total_negative_pages_extracted_succesfully > 0:
        recall = (total_pages_extracted_successfully / (total_pages_extracted_successfully + total_negative_pages_extracted_succesfully)) * 100

    accuracy = 0
    if true_negative + false_positive + total_pages_extracted_successfully + total_negative_pages_extracted_succesfully > 0:
        accuracy = ((true_negative + total_pages_extracted_successfully) / (true_negative + false_positive + total_pages_extracted_successfully + total_negative_

    extraction_results_metrics.append([
        total_pages_for_classifier,
        precision,
        recall,
        accuracy,
    ])
```

Extrator específico

```python
def divinho_extract(soup):
    try:
        name = None
        name_marker = soup.find('span', itemprop="name")
        if name_marker:
            name = name_marker.text

        wine_type = None
        wine_type_marker = soup.find('div', attrs={'data-code': 'tipo'})
        if wine_type_marker:
            wine_type = wine_type_marker.text.replace('\n', ' ').strip(' ')

        grape = None
        grape_marker = soup.find('div', attrs={'data-code': 'uva'})
        if grape_marker:
            grape = grape_marker.text.replace('\n', ' ').strip(' ')

        country = None
        country_marker = soup.find('div', attrs={'data-code': 'country_of_manufacture'})
        if country_marker:
            country = country_marker.text.replace('\n', ' ').strip(' ')

        classification = None
        classification_marker = soup.find('div', attrs={'data-code': 'classificacao'})
        if classification_marker:
            classification = classification_marker.text.replace('\n', ' ').strip(' ')

        alcohol_content = None
        alcohol_content_marker = soup.find('div', attrs={'data-code': 'teor_alcoolico'})
        if alcohol_content_marker:
            alcohol_content = alcohol_content_marker.text.replace('\n', ' ').strip(' ')
```

Extrator genérico

```python
def has_text(tag):
    text = tag.text.replace('\n', '').strip()
    return text != ''

def global_extract(big_soup):
    try:
        # Reduce the scope to the contents div
        soup = big_soup.find(attrs={'class': re.compile('.*(P|p)roduct.*', re.DOTALL)})

        # If the page has a main tag, the product info tends to be inside
        if soup.main:
            soup = soup.main

        name = None
        name_marker = soup.find(itemprop='name')
        if (name_marker):
            name = name_marker.text.replace('\n', '').strip()

        wine_type = None
        wine_type_marker = soup.find(text=re.compile('.*Tipo.*', re.DOTALL))
        if (wine_type_marker):
            wine_type_sibling = wine_type_marker.find_next(has_text)
            if wine_type_sibling:
                wine_type = wine_type_sibling.text.replace('\n', '').strip()

        grape = None
        grape_marker = soup.find(text=re.compile('.*Uvas?.*', re.DOTALL))
        if (grape_marker):
            grape_sibling = grape_marker.find_next(has_text)
            if grape_sibling:
```

```python
        country = None
        country_marker = soup.find(text=re.compile('.*País.*', re.DOTALL))
        if (country_marker):
            country_sibling = country_marker.find_next(has_text)
            if country_sibling:
                country = country_sibling.text.replace('\n', '').strip()

        classification = None
        classification_marker = soup.find(text=re.compile('.*Classificação.*', re.DOTALL))
        if (classification_marker):
            classification_sibling = classification_marker.find_next(has_text)
            if classification_sibling:
                classification = classification_sibling.text.replace('\n', '').strip()

        alcohol_content = None
        alcohol_content_marker = soup.find(text=re.compile('.*(Teor|Graduação).*', re.DOTALL))
        if (alcohol_content_marker):
            alcohol_content_sibling = alcohol_content_marker.find_next(has_text)
            if alcohol_content_sibling:
                alcohol_content = alcohol_content_sibling.text.replace('\n', '').strip()

        year = None
        year_marker = soup.find(text=re.compile('.*Safra.*', re.DOTALL))
        if (year_marker):
            year_sibling = year_marker.find_next(has_text)
            if year_sibling:
                year = year_sibling.text.replace('\n', '').strip()
```

Resultados extrator específico (Grand Cru)

Resultados extrator específico (Grand Cru)

	Total Pages	Precision	Recall	Accuracy
naive_bayer	754.0	67.10875331564988	74.52135493372607	76.01139601139602
mlp	698.0	68.19484240687679	70.10309278350515	74.98528546203649
svm	685.0	67.73722627737226	68.33578792341679	74.1399762752076
decision_tree	632.0	68.67088607594937	63.91752577319587	72.87201469687692
logistic_regression	715.0	68.25174825174825	71.87039764359352	75.64102564102564

Resultados extrator genérico (Grand Cru)

Resultados extrator genérico (Grand Cru)

	Total Pages	Precision	Recall	Accuracy
naive_bayer	754.0	68.16976127320955	74.27745664739885	76.18233618233619
mlp	698.0	69.05444126074498	69.65317919075144	74.92642731018246
svm	685.0	68.61313868613139	67.91907514450867	74.08066429418743
decision_tree	632.0	69.4620253164557	63.4393063583815	72.68830373545622
logistic_regression	715.0	69.0909090909091	71.38728323699422	75.58275058275058

Observações

- Em alguns casos, o extrator genérico foi até melhor pois tinha muitos fallbacks para os campos
- Quando o valor do campo não está numa tag HTML específica, nosso extrator genérico fica bem ruim, vide exemplos na próxima página

Resultados extrator específico (Mistral)

	Total Pages	Precision	Recall	Accuracy
naive_bayer	600.0	71.0	58.758620689655174	70.45596502186133
mlp	453.0	70.86092715231787	44.275862068965516	63.13617606602476
svm	442.0	70.13574660633483	42.758620689655174	62.09286209286209
decision_tree	454.0	70.70484581497797	44.275862068965516	63.092783505154635
logistic_regression	468.0	71.15384615384616	45.93103448275862	64.12525527569775

Resultados extrator genérico (Mistral)

	Total Pages	Precision	Recall	Accuracy
naive_bayer	600.0	0.0	0.0	62.52342286071205
mlp	453.0	0.0	0.0	68.84456671251719
svm	442.0	0.0	0.0	69.36936936936937
decision_tree	454.0	0.0	0.0	68.79725085910653
logistic_regression	468.0	0.0	0.0	68.14159292035397

Post mortem

- Vinho foi o melhor domínio?
- Sites escolhidos ajudaram?
- O que fazer pra melhorar o classificador?