Aprendizagem Automática

Aula Prática

Trabalhar com dados de texto

Modelo “Bag of Words”

Exemplo:

Análise de Críticas de Cinema no IMDb (Internet Movie Database)

𝔾. Marques

Dados de Texto

Dados de Texto

1. Modelo *Bag of Words* (BoW)
   - *Tokenization*
   - Construção do Vocabulário
   - Representação numérica
   - Método tf-idf
2. Base de dados IMDb
   - Estrutura dos dados
   - Leitura em Python
3. Trabalhar com os dados IMDb
   - Limpeza do vocabulário
   - *Stop Words*
   - *Stemming*
   - *n-gramas*

Dados de Texto

- Texto escrito de forma livre aparece em variados contextos, como por exemplo em tweets, comentários, o conteúdo da Wikipédia, o projecto Gutenberg, etc. Todos estes exemplos contêm informação sobre forma de frases compostas por palavras.
- Nas áreas de Processamento Natural de Linguagem (NLP) e de Recolha de Informação (IR), a base de dados (uma dada colecção de textos) é chamada de *corpus*, e um único texto de *documento*.
- Para poder utilizar técnicas de aprendizagem automática (classificação, regressão, clustering) neste tipo de dados **é necessário representar cada documento por um vetor numérico**.

Modelo Bag of Words (BoW)

Uma das formas de representar texto é o modelo “Bag of Word” (i.e. saco de palavras). Nesta representação a forma, estrutura do texto é descartada bem como a ordem das palavras e é só tido em conta o número de ocorrências de cada palavra em cada documento do corpus. O resultado final é uma matriz denominada “documento-termo” (do inglês *document-term matrix*) com dimensão $N \times d$, onde N o número de documentos no corpus e d é o número de palavras do vocabulário. BoW é uma técnica não supervisionada de representar um texto por um vetor numérico.

A representação BoW consiste no seguintes passos:

1. **Tokenization**
   Este processo consiste em dividir cada documento em palavras (ou *tokens*), por exemplo separando as palavras nos textos através dos caracteres de espaço ou pontuação.
2. **Construção do Vocabulário:**
   Construir um vocabulário constituído por todas ou por um sub-conjunto das palavras presentes no corpus.
3. **Codificação:**
   - Contar o número de vezes que cada palavra do vocabulário aparece em cada documento.
   - Representar cada documento por um vetor de d dimensões, uma por cada palavra no vocabulário, com valores proporcionais ao número de ocorrências dessa palavra no documento. (Nota: estes vetores terão a maior parte dos seus coeficientes a zero)
   - Construir a matriz documento-termo.

Modelo Bag of Words (BoW)

Uma das formas de representar texto é o modelo “Bag of Word” (i.e. saco de palavras). Nesta representação a forma, estrutura do texto é descartada bem como a ordem das palavras e é só tido em conta o número de ocorrências de cada palavra em cada documento do corpus. O resultado final é uma matriz denominada “documento-termo” (do inglês *document-term matrix*) com dimensão $N \times d$, onde N o número de documentos no corpus e d é o número de palavras do vocabulário. BoW é uma técnica não supervisionada de representar um texto por um vetor numérico.

Questões práticas:

1. Cada texto é representado por um vetor de dimensão igual ao número de palavras no vocabulário.
2. Devido à diversidade de termos, palavras, interjeições, etc, presentes na maioria dos idiomas, se não houver nenhum pré-processamento dos textos, o vocabulário pode ter vários milhares de palavras.
3. É por isso aconselhável antes de fazer a representação BoW, processar os documentos base de dados de forma a reduzir a dimensão do vocabulário. Existem várias técnicas, tais como considerar só palavras que tenham um número de ocorrências superior a um dado limiar, converter palavras semelhantes numa única palavra, etc.
4. O processo de limpeza tem que ser o mesmo para todos os documentos (documentos de treino e de teste, e outros documentos nunca classificados), e deve ser aplicado antes de obter a representação BoW.

Modelo Bag of Words (BoW)

Em Pyton:

A implementação do modelo BoW em `scikit-learn` é feita através da classe `CountVectorizer`. Esta classe aplica a representação BoW a strings ou ficheiros de texto. Por omissão, os caracteres são convertidos para letras minúsculas e as palavras (tokens) são obtidas através de uma expressão regular que divide strings nos espaços brancos e pontuação.

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus=[
"you better start swimming or you'll sink like a stone for the times they are a-changing",
"the loser now will be later to win cause the times they are a-changing",
"it'll soon shake your windows and rattle your walls for the times they are a-changing"]
# (três versos de uma música de Bob Dylan)
>>> CVect=CountVectorizer()
>>> CVect.fit(corpus)
>>> Vocab=CVect.vocabulary_
>>> print(Vocab)
u'and': 0, u'be': 2, u'rattle': 14, u'win': 27, u'it': 7, u'soon': 17, u'walls': 25,
u'are': 1, u'they': 22, u'shake': 15, u'changing': 5, u'now': 12, u'the': 21, u'your':
30, u'will': 26, u'stone': 19, u'like': 9, u'for': 6, u'start': 18, u'windows': 28,
u'll': 10, u'later': 8, u'loser': 11, u'times': 23, u'better': 3, u'to': 24, u'you':
29, u'swimming': 20, u'cause': 4, u'or': 13, u'sink': 16
```

- `Vocab` – variável do tipo dicionário, contendo as palavras como chaves, e índices das mesmas como valores (31 entradas no total).

Modelo Bag of Words (BoW)

Em Pyton:

- Para obter a representação matricial BoW, usa-se o método `.transform()`

```
>>> BoW=CVect.transform(corpus)
>>> BoW
<3x31 sparse matrix of type '<type 'numpy.int64'>' with 43 stored elements in
Compressed Sparse Row format>
```

- Esta representação usa uma matriz esparsa de `SciPy` em que só os valores não nulos são guardados. Pode-se usar o método associado a esta classe, `.toarray()` para a converter num `NumPy` array. Neste caso resulta numa matriz de 3×31.
Atenção: não é aconselhável fazer esta conversão quando se lida com um corpus com milhares de documentos, como é o caso do IMDb.

```
>>> X=BoW.toarray()
>>> print(X)
[[0 1 0 1 0 1 1 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 2 0]
 [0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 2 1 1 1 0 1 1 0 0 0]
 [1 1 0 0 0 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 2]]
```

Matriz contém o número de ocorrências de cada termo do vocabulário. O primeiro `and` só aparece no terceiro verso, enquanto o segundo `are` já aparece nos três (o mesmo para `changing` 5º, `the` 21º, `they` 22º, e `times` 23º). Os termos que aparecem duas vezes são `the` 21º, `you` 29º e `your` 30º.

Modelo Bag of Words (BoW)

Em Pyton:

- Pode ser útil ter as palavras do vocabulário ordenadas pela seu índice. De notar que da forma que se encontra guardado (num dicionário), não é imediato aceder à palavra pelo índice. Os dois primeiros comandos criam uma lista com as palavras do vocabulário ordenadas segundo os seu índices.

```
>>> word,idx=(Vocab.keys(),np.argsort(Vocab.values()))
>>> wordSort = [ word[i] for i in idx]
>>> for i in range(len(wordSort)):
        print('%d.  %s'%(i,wordSort[i]))
```

```
0.  and
1.  are
2.  be
3.  better
4.  cause
5.  changing
6.  for
7.  it
8.  later
9.  like
10.  ll
11.  loser
12.  now
13.  or
14.  rattle
15.  shake
16.  sink
17.  soon
18.  start
19.  stone
20.  swimming
21.  the
22.  they
23.  times
24.  to
25.  walls
26.  will
27.  win
28.  windows
29.  you
30.  your
```

- Neste caso, a classe `CountVectorizer` já tem um método associado que retorna as palavras do vocabulário pela ordem correcta (a mesma que o índice das colunas da matriz documento-termo).

```
>>> wordSort2 = CVect.get_feature_names()
```

Método tf-idf

tf-idf: term frequency-inverse document frequency

A técnica tf-idf representa de outra forma a matriz de contagens documento-termo, dando mais relevância a termos mais informativos. A intuição por detrás do método é dar um peso maior a termos que aparecem frequentemente num dado documento, mas não em muitos documentos do corpus. Se uma palavra aparece em poucos documentos, mas muitas vezes num em particular, então o pressuposto é que essa palavra descreve bem o conteúdo desse documento.

- Há muitas variantes do método e da formula usada para pesar os termos do vocabulário. No `scikit-learn` é usado:

$$\text{tf-idf}(p,t) = \text{tf} \times \left(\log \left(\frac{N+1}{N_p+1} \right) + 1 \right)$$

  tf-idf(p,t) é o valor dado à palavra p do texto (documento) t. tf é o número de ocorrências normalizado de p em t (ver próximo slide). N é o número total de documentos no corpus e N_p é o número de documentos onde aparece a palavra p.

Método tf-idf

tf: term frequency

O modelo BoW converte o corpus na matriz "document-term" em estão o número de ocorrências das palavras do vocabulário por cada documento. Isto faz com que documentos longos tenha valores mais elevados que documentos curtos. O comprimento de um documento não é relevante para atribuir a classe, mas a diferença de valores entre linhas da matriz pode afetar o desempenho da classificação. A solução passa por uma normalização de cada vetor/documento:

- Dado que os valores são positivos pode-se normalizar cada vetor segundo $\mathbf{x}_{\text{novo}} = \frac{\mathbf{x}}{\sum_i x_i}$, onde $\mathbf{x} = [x_1, x_2, \ldots, x_d]^\top$ é a representação BoW de um documento e d o número de palavras no vocabulário.
- Outra normalização comum é escalar cada vetor/documento de modo a ter norma unitária: $\mathbf{x}_{\text{novo}} = \frac{\mathbf{x}}{\sqrt{\sum_i x_i^2}}$
  (esta normalização é feita pelas classes do `scikit-learn`, `TfidfTransformer` e `TfidfVectorizer`).

Método tf-idf

Em Pyton:

A implementação do método tf-idf em `scikit-learn` é feita através de uma de duas classes: `TfidfTransformer`, e `TfidfVectorizer`.

- `TfidfTransformer`:
  Esta classe recebe como argumento a matriz documento-termo obtida por `CountVectorizer` e devolve a matriz tf-idf.

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> CVect=CountVectorizer().fit(corpus)
>>> X=CVect.transform(corpus)
>>> tfidf=TfidfTransformer()
>>> tfidf.fit(X)
>>> Y=tfidf.transform(X)
>>> print(Y.toarray()) # (só as 1ªs 21 colunas)
[[0.0  0.16 0.0  0.27 0.0  0.16 0.2  0.0  0.0  0.27 0.2  0.0  0.0  0.27 0.0  0.0  0.27 0.0  0.27 0.27 0.16 ···
 [0.0  0.18 0.3  0.0  0.3  0.18 0.0  0.0  0.3  0.0  0.0  0.3  0.3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.36 ···
 [0.27 0.16 0.0  0.0  0.0  0.16 0.2  0.27 0.0  0.0  0.2  0.0  0.0  0.0  0.27 0.27 0.0  0.27 0.0  0    0.16 ···
>>> print(X.toarray())
[[0 1 0 1 0 1 1 0 0 1 1 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 2 0]
 [0 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 2 1 1 1 0 1 1 0 0 0]
 [1 1 0 0 0 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 1 0 2]]
```

Método tf-idf

Em Pyton:

A implementação do método tf-idf em `scikit-learn` é feita através de uma de duas classes: `TfidfTransformer`, e `TfidfVectorizer`.

- `TfidfVectorizer`:
  Esta classe recebe directamente os dados de texto e internamente calcula representação BoW (feita `CountVectorizer`), e calcula igualmente a representação tf-idf.

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> tfidf=tfidfVectorizer().fit(corpus)
>>> Y=tfidf.transform(corpus)
>>> print(Y.toarray()) # (só as 1ªs 21 colunas)
```

```
[[0.0  0.16 0.0  0.27 0.0  0.16 0.2  0.0  0.0  0.27 0.2  0.0  0.0  0.27 0.0  0.0  0.27 0.0  0.27 0.27 0.16 ···
 [0.0  0.18 0.3  0.0  0.3  0.18 0.0  0.0  0.3  0.0  0.0  0.3  0.3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.36 ···
 [0.27 0.16 0.0  0.0  0.0  0.16 0.2  0.27 0.0  0.0  0.2  0.0  0.0  0.0  0.27 0.27 0.0  0.27 0.0  0    0.16 ···
```

Base de dados IMDb

A IMDb (de Internet Movie Database) é um conjunto de críticas de cinema recolhidas por Andrew Maas, um investigador da Universidade de Stanford, e que se encontram disponiveis em http://ai.stanford.edu/ amaas/data/sentiment/.

- A base de dados contém 50,000 críticas (em forma de texto - 84.1MB total) dividida em duas partes de 25k, uma para treino e outra para teste .
- A base de dados está balanceada: tem um número igual de críticas positivas e negativas (25,000 cada).
- Não é permitido mais do que 30 críticas por filme.
- Os dados de treino e testes contêm conjuntos de filmes disjuntos.
- As críticas negativas tem uma pontuação (rating) $4 \leq$ em 10.
- As críticas positivas tem uma pontuação ≥ 7 em 10.
- Críticas mais neutrais não estão incluídas (ratings 5 e 6).

Base de dados IMDb

A IMDb (de Internet Movie Database) é um conjunto de críticas de cinema recolhidas por Andrew Maas, um investigador da Universidade de Stanford, e que se encontram disponiveis em http://ai.stanford.edu/ amaas/data/sentiment/.

Esta base de dados também contém 50,000 documentos não classificados (sem ratings) que se encontram na sub-directoria `unsup` da directoria `train/`.

- Críticas com todos com todos os ratings (de 1-10) estão incluídas.
- Existem um número igual de documentos com ratings > 5 e com $5 \leq$.

Tarefas associadas à IMDb

Esta base de dados está em primeiro lugar estruturada para um problema de classificação binária: determinar se uma determinada crítica é positiva ou negativa. Pode também ser usada noutros dois contextos:

- Regressão: Os ficheiros têm no seu nome, o rating dado. Pode-se por isso tentar prever, através de técnicas de regressão, o valor do rating baseado no texto da crítica.
- Clustering: Pode-se com todos os dados (os com e sem rating) descobrir através das palavras partilhadas entre textos, se há certos grupos de documentos que se focam sobre temas ou áreas específicas. Para tal, pode-se usar métodos de agrupamento (ou clustering) como é, por exemplo, o algoritmo k-médias.

Base de dados IMDb

Em Pyton:

- Estrutura de ficheiros:
  - O ficheiro `clImdb_v1.tar.gz` contém duas directorias de topo `train/` e `test/` que correspondem as dados de treino e teste.
  - Cada uma destas directorias contém duas sub-directorias `pos/` e `neg/` com os exemplos positivos e negativos respectivamente.
  - Para os ficheiros de texto respectivos às críticas, a atribuição do nome do ficheiro seguiu a seguinte convenção: `id_rating.txt`. `id` é um identificador único e `rating` é a pontuação (escala de 1-10).
  - No conjunto não-supervisionado, tem `0` para todos os ratings.
- O `scikit-learn` tem uma função para carregar dados com esta estrutura de directorias denominada `load_files`.

Base de dados IMDb

Em Pyton:

Atenção: Mova a sub-directoria `unsup` para a directoria de topo (conjuntamente com as directorias `train` e `test`). Caso contrário serão carregado mais 50,000 ficheiros de texto!

```
>>> from sklearn.datasets import load_files
>>> trainDic=load_files('aclImdb/train/')
```

Ficheiros de treino carregados num dicionário (com campos semelhantes às outras bases de dados do `scikit-learn`. Para carregar os dados de treino e respectivas classes, basta fazer:

```
>>> text_train=trainDic.data
>>> class_train=trainDic.target
>>> print('Tipo de dados:  %s'%type(text_train))
>>> print('Tamanho:  %d'%len(text_train))
>>> print('Número de classes:  %d'%len(np.unique(class_train)))
>>> print('Número de +:  %d'%np.sum(class_train==1))
>>> print('Número de -:  %d'%np.sum(class_train==0))
Tipo de dados:  <type 'list'>
Tamanho:  25000
Número de classes:  2
Número de +:  12500
Número de -:  12500
```

Base de dados IMDb

Em Pyton:

Atenção: Mova a sub-directoria `unsup` para a directoria de topo (conjuntamente com as directorias `train` e `test`). Caso contrário serão carregado mais 50,000 ficheiros de texto!

```
>>> from sklearn.datasets import load_files
>>> trainDic=load_files('aclImdb/train/')
>>> text_train,class_train=trainDic.data,trainDic.target
>>> print('2ª crítica negativa:%s'% text_train[3]))
```

```
There are a lot of highly talented filmmakers/actors in Germany now.  None of them are
associated with this "movie".<br /><br />Why in the world do producers actually invest
money in something like this this?  You could have made 10 good films with the budget of
this garbage!  It's not entertaining to have seven grown men running around as dwarfs,
pretending to be funny.  What IS funny though is that the film's producer (who happens to
be the oldest guy of the bunch) is playing the YOUNGEST dwarf.<br /><br />The film is filled
with moments that scream for captions saying "You're supposed to laugh now!".  It's hard to
believe that this crap's supposed to be a comedy.<br /><br />Many people actually stood up and
left the cinema 30 minutes into the movie.  I should have done the same instead of wasting my
time...<br /><br />Pain!
```

Para saber qual o ficheiro correspondente a esta crítica:

```
>>> print(trainDic.filenames[3])
./aclImdb/train/neg/9698_1.txt
```

Trabalhar como os dados IMDb

Pode-se aplicar directamente aos textos de treino os métodos do `scikit-learn` para obter a representação documento-term (com contagens normalizadas ou com valores tf-idf). Como podemos ver, convém primeiro limpar o vocabulário antes de aplicar qualquer representação matricial BoW.
(é assumido que a variável `text_train` é uma lista que contém as críticas de treino).

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> tfidf=tfidfVectorizer().fit(text_train)
>>> Y=tfidf.transform(text_train)
<25000x74849 sparse matrix of type '<type 'numpy.float64'>'
with 3445861 stored elements in Compressed Sparse Row format>
>>> words=tfidf.get_feature_names()
```

A representação tf-idf contém um vocabulário com 74849, o que é aproximadamente 3 vezes mais que o número de criticas (25000). Analisando as palavras do vocabulário vemos que há algumas que não são relevantes (ex: números, símbolos), e muitas que são relacionadas e por isso não é ideal atribuir diferentes termos a palavras com um sentido semântico tão próximo.

```
>>> words[:30]# 1ªs 30 palavras
[u'00', u'000', u'0000000000001', u'00001', u'00015', u'000s', u'001', u'003830', u'006',
u'007', u'0079', u'0080', u'0083', u'0093638', u'00am', u'00pm', u'00s', u'01', u'01pm',
u'02', u'020410', u'029', u'03', u'04', u'041', u'05', u'050', u'06', u'06th', u'07']
```

Trabalhar como os dados IMDb

Pode-se aplicar directamente aos textos de treino os métodos do `scikit-learn` para obter a representação documento-term (com contagens normalizadas ou com valores tf-idf). Como podemos ver, convém primeiro limpar o vocabulário antes de aplicar qualquer representação matricial BoW.
(é assumido que a variável `text_train` é uma lista que contém as críticas de treino).

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> tfidf=tfidfVectorizer().fit(text_train)
>>> Y=tfidf.transform(text_train)
<25000x74849 sparse matrix of type '<type 'numpy.float64'>'
with 3445861 stored elements in Compressed Sparse Row format>
>>> words=tfidf.get_feature_names()
```

A representação tf-idf contém um vocabulário com 74849, o que é aproximadamente 3 vezes mais que o número de criticas (25000). Analisando as palavras do vocabulário vemos que há algumas que não são relevantes (ex: números, símbolos), e muitas que são relacionadas e por isso não é ideal atribuir diferentes termos a palavras com um sentido semântico tão próximo.

```
>>> words[1000:1030]# de 1000 a 1030
[u'9of10', u'9pm', u'9s', u'9th', u'____', u'_____', u'______', u'________', u'_____________________________',
u'_____________________________', u'____________________________________________________________', u'_a', u'_absolute',
u'_am_', u'_and_', u'_angel_', u'_annie_', u'_any_', u'_anything_', u'_apocalyptically', u'_as',
u'_atlantis', u'_atlantis_', u'_attack', u'_before_', u'_blair', u'_both_', u'_bounce_', u'_by',
u'_can']
```

Trabalhar como os dados IMDb

Pode-se aplicar directamente aos textos de treino os métodos do `scikit-learn` para obter a representação documento-term (com contagens normalizadas ou com valores tf-idf). Como podemos ver, convém primeiro limpar o vocabulário antes de aplicar qualquer representação matricial BoW.
(é assumido que a variável `text_train` é uma lista que contém as críticas de treino).

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> tfidf=tfidfVectorizer().fit(text_train)
>>> Y=tfidf.transform(text_train)
<25000x74849 sparse matrix of type '<type 'numpy.float64'>'
with 3445861 stored elements in Compressed Sparse Row format>
>>> words=tfidf.get_feature_names()
```

A representação tf-idf contém um vocabulário com 74849, o que é aproximadamente 3 vezes mais que o número de criticas (25000). Analisando as palavras do vocabulário vemos que há algumas que não são relevantes (ex: números, símbolos), e muitas que são relacionadas e por isso não é ideal atribuir diferentes termos a palavras com um sentido semântico tão próximo.

```
>>> words[25000:25030]# de 25000 a 25030
[u'fleece', u'fleecing', u'fleed', u'fleeing', u'fleer', u'flees', u'fleet', u'fleeting',
u'fleetingly', u'fleetwood', u'fleggenheimer', u'fleischer', u'fleischers', u'fleisher',
u'fleming', u'flemish', u'flemming', u'flemmish', u'flemyng', u'flender', u'flesh',
u'flesheaters', u'flesheating', u'fleshed', u'fleshes', u'fleshiness', u'fleshing',
u'fleshpots', u'fleshy', u'fletch']
```

Trabalhar como os dados IMDb

Limpeza do texto

Existem muitos caracteres ou sequências de caracteres, como mudanças de linha em HTML `<br />`, que não contribuem para discriminar entre boas e más críticas mas podem prejudicar bastante o desempenho dos classificadores. Existem alguns comandos simples que ajudam a limpar o vocabulário.

- Para limpar caracteres ou sequências de caracteres pode-se recorrer a funções da classe `string` como a função `.replace()`. Por exemplo, para tirar a mudança de linha HTML, basta fazer:

  ```
  >>> text_train=[doc.replace('<br />',' ') for doc in text_train]
  ```

  Nota: este comando cria uma lista em que cada entrada é um documento da lista `text_train` com a sequência de caracteres `<br />` substituída por um espaço.
- Pode-se igualmente usar o módulo de expressões regulares `re`, para fazer a limpeza dos textos. Por exemplo, para guardar só os caracteres alfabéticos, pode-se fazer o seguinte comando:

  ```
  >>> import re
  >>> text_train=[re.sub(r'[^a-zA-Z]+',' ',doc) for doc in
  text_train]
  ```

  Atenção: A ordem das operações de limpeza é importante! Pode alterar os resultados.

Trabalhar como os dados IMDb

Limpeza do texto

- Ao executar estes comandos, são eliminadas do vocabulário 1603 palavras:

```
>>> import re
>>> text_train=[doc.replace('<br />',' ') for doc in text_train]
>>> text_train=[re.sub(r'[^a-zA-Z]+',' ',doc) for doc in
text_train]
>>> tfidf=tfidfVectorizer().fit(text_train)
>>> words=tfidf.get_feature_names()
>>> print(len(words))
73246
```

Apesar desta redução não ser muito substancial, descartar palavras desnecessárias reduz o tempo de processamento e torna os algoritmos mais robustos e interpretáveis.

Trabalhar como os dados IMDb

Limpeza do texto

- As classes `CountVectorizer` e `TfidfVectorizer` também executam uma limpeza dos dados. Estas classes só extraem palavras com comprimento maior que dois caracteres e converte os caracteres alfabéticos para minúsculas.
  Há igualmente a possibilidade de extrair só as palavras que aparecem em mais que um número pré-definido de documentos com o parâmetro `min_df` (minimum document frequency). Por exemplo, o seguinte comando só extrai palavras que aparecem em 5 ou mais documentos.

```
>>> tfidf=tfidfVectorizer(min_df=5).fit(text_train)
>>> words=tfidf.get_feature_names()
>>> print(len(words))
26970
```

  Ao escolher só as palavras que aparecem em 5 ou mais documentos, reduzimos substancialmente a dimensão do vocabulário. No entanto, podemos ainda reduzir mais o vocabulário visto haver muitas palavras com sentidos semânticos muito próximos que podem ser codificadas num único token.

Trabalhar como os dados IMDb

Limpeza do texto

- As classes `CountVectorizer` e `TfidfVectorizer` também executam uma limpeza dos dados. Estas classes só extraem palavras com comprimento maior que dois caracteres e converte os caracteres alfabéticos para minúsculas.
  Há igualmente a possibilidade de extrair só as palavras que aparecem em mais que um número pré-definido de documentos com o parâmetro `min_df` (minimum document frequency). Por exemplo, o seguinte comando só extrai palavras que aparecem em 5 ou mais documentos.

```
>>> tfidf=tfidfVectorizer(min_df=5).fit(text_train)
>>> words=tfidf.get_feature_names()
>>> print(len(words))
26970
```

```
>>> words[:30]# 1as 30 palavras
[u'aa', u'aaa', u'aag', u'aames', u'aamir', u'aankhen', u'aardman', u'aaron', u'ab', u'aback',
u'abandon', u'abandoned', u'abandoning', u'abandonment', u'abandons', u'abbas', u'abbey',
u'abbot', u'abbott', u'abbreviated', u'abby', u'abc', u'abduct', u'abducted', u'abduction',
u'abe', u'abel', u'abetted', u'abhay', u'abhishek']
```

Trabalhar como os dados IMDb

Limpeza do texto

- As classes `CountVectorizer` e `TfidfVectorizer` também executam uma limpeza dos dados. Estas classes só extraem palavras com comprimento maior que dois caracteres e converte os caracteres alfabéticos para minúsculas.
  Há igualmente a possibilidade de extrair só as palavras que aparecem em mais que um número pré-definido de documentos com o parâmetro `min_df` (minimum document frequency). Por exemplo, o seguinte comando só extrai palavras que aparecem em 5 ou mais documentos.

```
>>> tfidf=tfidfVectorizer(min_df=5).fit(text_train)
>>> words=tfidf.get_feature_names()
>>> print(len(words))
26970
```

```
>>> words[1000:1030]# de 1000 a 1030
[u'anywhere', u'ao', u'aoki', u'ap', u'apache', u'apart', u'apartheid', u'apartment',
u'apartments', u'apathetic', u'apathy', u'apatow', u'ape', u'apes', u'apex', u'aphrodite',
u'aplenty', u'aplomb', u'apocalypse', u'apocalyptic', u'apollo', u'apollonia', u'apologies',
u'apologise', u'apologist', u'apologize', u'apologized', u'apologizes', u'apologizing',
u'apology']
```

Trabalhar como os dados IMDb

Limpeza do texto

- As classes `CountVectorizer` e `TfidfVectorizer` também executam uma limpeza dos dados. Estas classes só extraem palavras com comprimento maior que dois caracteres e converte os caracteres alfabéticos para minúsculas.
  Há igualmente a possibilidade de extrair só as palavras que aparecem em mais que um número pré-definido de documentos com o parâmetro `min_df` (minimum document frequency). Por exemplo, o seguinte comando só extrai palavras que aparecem em 5 ou mais documentos.

```
>>> tfidf=tfidfVectorizer(min_df=5).fit(text_train)
>>> words=tfidf.get_feature_names()
>>> print(len(words))
26970
```

```
>>> words[25000:25030]# de 25000 a 25030
[u'umpteenth', u'un', u'una', u'unabashed', u'unable', u'unacceptable', u'unadulterated',
u'unaffected', u'unafraid', u'unanimous', u'unanswered', u'unapologetic', u'unapologetically',
u'unappealing', u'unappetizing', u'unappreciated', u'unarmed', u'unashamed', u'unashamedly',
u'unassuming', u'unattractive', u'unavailable', u'unavoidable', u'unavoidably', u'unaware',
u'unbalanced', u'unbearable', u'unbearably', u'unbeatable', u'unbecoming']
```

Trabalhar como os dados IMDb

Limpeza do texto

- As classes `CountVectorizer` e `TfidfVectorizer` também executam uma limpeza dos dados. Estas classes só extraem palavras com comprimento maior que dois caracteres e converte os caracteres alfabéticos para minúsculas.
  Há igualmente a possibilidade especificar qual a expressão regular usada no processo de tokenização. Por omissão a expressão é `r"\b\w\w+\b"`. Isto significa que serão extraídas sequências de caracteres compostas por 2 ou mais letras ou números (`\w`) e que estão separadas por caracteres de pontuação ou espaços (`\b`). Por exemplo, o seguinte comando, além de extrair só palavras que aparecem em 5 ou mais documentos, também só extrai palavras com 4 ou mais caracteres

```
>>> tfidf=tfidfVectorizer(min_df=5,token_pattern=r'\b\w\w\w\w+\b')
>>> tfidf.fit(text_train)
>>> print(len(tfidf.get_feature_names()))
25855
```

```
>>> words[:30]# 1as 30 palavras
[u'aames', u'aamir', u'aankhen', u'aardman', u'aaron', u'aback', u'abandon', u'abandoned',
u'abandoning', u'abandonment', u'abandons', u'abbas', u'abbey', u'abbot', u'abbott',
u'abbreviated', u'abby', u'abduct', u'abducted', u'abduction', u'abel', u'abetted', u'abhay',
u'abhishek', u'abhorrent', u'abiding', u'abigail', u'abilities', u'ability', u'abject']
```

Trabalhar como os dados IMDb

Limpeza do texto

- As classes `CountVectorizer` e `TfidfVectorizer` também executam uma limpeza dos dados. Estas classes só extraem palavras com comprimento maior que dois caracteres e converte os caracteres alfabéticos para minúsculas.
  Há igualmente a possibilidade especificar qual a expressão regular usada no processo de tokenização. Por omissão a expressão é `r"\b\w\w+\b"`. Isto significa que serão extraídas sequências de caracteres compostas por 2 ou mais letras ou números (`\w`) e que estão separadas por caracteres de pontuação ou espaços (`\b`). Por exemplo, o seguinte comando, além de extrair só palavras que aparecem em 5 ou mais documentos, também só extrai palavras com 4 ou mais caracteres

```
>>> tfidf=tfidfVectorizer(min_df=5,token_pattern=r'\b\w\w\w\w+\b')
>>> tfidf.fit(text_train)
>>> print(len(tfidf.get_feature_names()))
25855
```

```
>>> words[1000:1030]# de 1000 a 1030
[u'applauds', u'applause', u'apple', u'appleby', u'applegate', u'apples', u'appliance',
u'appliances', u'applicable', u'application', u'applied', u'applies', u'apply', u'applying',
u'appointed', u'appointment', u'apposite', u'appreciable', u'appreciate', u'appreciated',
u'appreciates', u'appreciating', u'appreciation', u'appreciative', u'apprehended',
u'apprehension', u'apprehensive', u'apprentice', u'approach', u'approached']
```

Trabalhar como os dados IMDb

Limpeza do texto

- As classes `CountVectorizer` e `TfidfVectorizer` também executam uma limpeza dos dados. Estas classes só extraem palavras com comprimento maior que dois caracteres e converte os caracteres alfabéticos para minúsculas.
  Há igualmente a possibilidade especificar qual a expressão regular usada no processo de tokenização. Por omissão a expressão é `r"\b\w\w+\b"`. Isto significa que serão extraídas sequências de caracteres compostas por 2 ou mais letras ou números (`\w`) e que estão separadas por caracteres de pontuação ou espaços (`\b`). Por exemplo, o seguinte comando, além de extrair só palavras que aparecem em 5 ou mais documentos, também só extrai palavras com 4 ou mais caracteres

```
>>> tfidf=tfidfVectorizer(min_df=5,token_pattern=r'\b\w\w\w\w+\b')
>>> tfidf.fit(text_train)
>>> print(len(tfidf.get_feature_names()))
25855
```

```
>>> words[25000:25030]# de 25000 a 25030
[u'walters', u'walthall', u'walton', u'waltons', u'waltz', u'wanda', u'wander', u'wandered',
u'wandering', u'wanderings', u'wanders', u'wane', u'waned', u'wang', u'waning', u'wanna',
u'wannabe', u'wannabee', u'wannabes', u'want', u'wanted', u'wanting', u'wanton', u'wants',
u'waqt', u'warbling', u'warburton', u'ward', u'warden', u'wardh']
```

Trabalhar como os dados IMDb

Stop Words

Outra maneira de reduzir o tamanho do vocabulário é eliminar palavras é através de uma listas de “stop words”. Stop words são as palavras que ocorrem frequentemente numa dada língua (cada idioma tem o seu conjunto específico de stop words). O `scikit-learn` contém uma lista de stop words em inglês no módulo `feature_extraction.text`.

```
>>> from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
>>> print(len(ENGLISH_STOP_WORDS))
318
>>> print(ENGLISH_STOP_WORDS)
'all', 'six', 'less', 'being', 'indeed', 'over', 'move', 'anyway', 'fifty', 'four', 'not',
'own', 'through', 'yourselves', 'go', 'where', 'mill', 'only', 'find', 'before', 'one',
'whose', 'system', 'how', 'somewhere', 'with', 'thick', 'show', 'had', 'enough', 'should',
'to', 'must', 'whom', 'seeming', 'under', 'ours', 'has', 'might', 'thereafter', 'latterly',
'do', 'them','his', 'around', 'than', 'get', 'very', 'de', 'none', 'cannot', 'every',
'whether', 'they','front', 'during', 'thus', 'now', 'him', 'nor', 'name', 'several',
'hereafter', 'always', 'who','cry', 'whither', 'this', 'someone', 'either', 'each',
'become', 'thereupon', 'sometime', 'side','two', 'therein', 'twelve', 'because', 'often',
'ten', 'our', 'eg', 'some', 'back', 'up', 'namely','towards', 'are', 'further', 'beyond',
'ourselves', 'yet', 'out', 'even', 'will', 'what', 'still','for', 'bottom', 'mine', 'since',
'please', 'forty', 'per', 'its', 'everything', 'behind', 'un','above', 'between', 'it', ...
```

Trabalhar como os dados IMDb

Stop Words

Outra maneira de reduzir o tamanho do vocabulário é eliminar palavras é através de uma listas de “stop words”. Stop words são as palavras que ocorrem frequentemente numa dada língua (cada idioma tem o seu conjunto específico de stop words). O `scikit-learn` contém uma lista de stop words em inglês no módulo `feature_extraction.text`.

```
>>> tp=r'\b\w\w\w\w+\b'
>>>
tfidf=tfidfVectorizer(min_df=5,stop_words="english",token_pattern=tp)
>>> tfidf.fit(text_train)
>>> print(len(tfidf.get_feature_names()))
25626   # 229 palavras removidas
```

- A remoção das stop words não tem uma contribuição significativa para o melhoramento da caracterização dos documentos(pode-se treinar modelos com vocabulários sem e com stop words e verificar se há melhoria no desempenho). Em certas situações, como é o caso de n-gramas, a remoção pode ser prejudicial. Existem outros métodos mais eficazes de reduzir a dimensão do vocabulário.

Trabalhar como os dados IMDb

Stemming

Esta técnica é o processo de transformar uma palavra na sua raiz, o que permite mapear palavras semelhantes numa única palavra (ex:`studies`, `studying`, `studied`,$\Longrightarrow$ `studi`). O primeiro algoritmo foi desenvolvido por Martin F. Porter, e ficou conhecido como Porter Stemmer. Este algoritmo está disponível no módulo de Python Natural Language Toolkit (`nltk`), conjuntamente com outros algoritmos de stemming como são os casos do Snowball e do Lancaster stermmers.

Porter Stemmer:

```
>>> from nltk.stem import PorterStemmer
>>> porter=PorterStemmer()
>>> DocText='descend descendant descendants descended descending descends descent described
describes describing descript description descriptions descriptive'
>>> print([porter.stem(word) for word in DocText.split()])
['descend', 'descend', 'descend', 'descend', 'descend', 'descend', 'descent', 'describ',
'describ', 'describ', 'descript', 'descript', 'descript', 'descript']
```

Trabalhar como os dados IMDb

Stemming

Esta técnica é o processo de transformar uma palavra na sua raiz, o que permite mapear palavras semelhantes numa única palavra (ex:`studies`, `studying`, `studied`,$\Longrightarrow$ `studi`). O primeiro algoritmo foi desenvolvido por Martin F. Porter, e ficou conhecido como Porter Stemmer. Este algoritmo está disponível no módulo de Python Natural Language Toolkit (`nltk`), conjuntamente com outros algoritmos de stemming como são os casos do Snowball e do Lancaster stermmers.

Snowball Stemmer:

```
>>> from nltk.stem import SnowballStemmer
>>> snowB=SnowballStemmer('english')# é necessario definir a língua
>>> DocText='descend descendant descendants descended descending descends descent described
describes describing descript description descriptions descriptive'
>>> print([snowB.stem(word) for word in DocText.split()])
['descend', 'descend', 'descend', 'descend', 'descend', 'descend', 'descent', 'describ',
'describ', 'describ', 'descript', 'descript', 'descript', 'descript']
```

Trabalhar como os dados IMDb

Stemming

Esta técnica é o processo de transformar uma palavra na sua raiz, o que permite mapear palavras semelhantes numa única palavra (ex:`studies`, `studying`, `studied`,$\Longrightarrow$ `studi`). O primeiro algoritmo foi desenvolvido por Martin F. Porter, e ficou conhecido como Porter Stemmer. Este algoritmo está disponível no módulo de Python Natural Language Toolkit (`nltk`), conjuntamente com outros algoritmos de stemming como são os casos do Snowball e do Lancaster stermmers.

Lancaster Stemmer:

```
>>> from nltk.stem import LancasterStemmer
>>> lanc=LancasterStemmer()
>>> DocText='descend descendant descendants descended descending descends descent described
describes describing descript description descriptions descriptive'
>>> print([lanc.stem(word) for word in DocText.split()])
['descend', 'descend', 'descend', 'descend', 'descend', 'descend', 'desc', 'describ',
'describ', 'describ', 'describ', 'describ', 'describ']
```

Trabalhar como os dados IMDb

Stemming

As funções de stemming do `nltk` funcionam palavra a palavra, por isso, para limpar a base de dados IMDb é necessário extrair cada documento (crítica) e dividir-lo em palavras e por fim aplicar o algoritmo de stemming às palavras individuais. Pode-se no entanto aferir os benefícios desta técnica sem ter que fazer stemming a todos os documentos do corpus, basta fazer stemming às palavras do vocabulário. Desta forma temos noção da redução do número de palavras do vocabulário.

```
>>> tp=r'\b\w\w\w\w+\b'
>>> tfidf=tfidfVectorizer(min_df=5,stop_words="english",token_pattern=tp)
>>> tfidf.fit(text_train)
>>> vocab=tfidf.get_feature_names()# 25626 palavras
```

Porter Stemmer:

```
>>> stemFunc=PorterStemmer()
>>> vocabNew=[stemFunc.stem(w) for w in vocab]
>>> print(len(np.unique(vocabNew)))
16489
```

Trabalhar como os dados IMDb

Stemming

As funções de stemming do `nltk` funcionam palavra a palavra, por isso, para limpar a base de dados IMDb é necessário extrair cada documento (crítica) e dividir-lo em palavras e por fim aplicar o algoritmo de stemming às palavras individuais. Pode-se no entanto aferir os benefícios desta técnica sem ter que fazer stemming a todos os documentos do corpus, basta fazer stemming às palavras do vocabulário. Desta forma temos noção da redução do número de palavras do vocabulário.

```
>>> tp=r'\b\w\w\w\w+\b'
>>> tfidf=tfidfVectorizer(min_df=5,stop_words="english",token_pattern=tp)
>>> tfidf.fit(text_train)
>>> vocab=tfidf.get_feature_names()# 25626 palavras
```

Snowball Stemmer:

```
>>> stemFunc=SnowballStemmer('english')
>>> vocabNew=[stemFunc.stem(w) for w in vocab]
>>> print(len(np.unique(vocabNew)))
16153
```

Trabalhar como os dados IMDb

Stemming

As funções de stemming do `nltk` funcionam palavra a palavra, por isso, para limpar a base de dados IMDb é necessário extrair cada documento (crítica) e dividir-lo em palavras e por fim aplicar o algoritmo de stemming às palavras individuais. Pode-se no entanto aferir os benefícios desta técnica sem ter que fazer stemming a todos os documentos do corpus, basta fazer stemming às palavras do vocabulário. Desta forma temos noção da redução do número de palavras do vocabulário.

```
>>> tp=r'\b\w\w\w\w+\b'
>>> tfidf=tfidfVectorizer(min_df=5,stop_words="english",token_pattern=tp)
>>> tfidf.fit(text_train)
>>> vocab=tfidf.get_feature_names()# 25626 palavras
```

Lancaster Stemmer:

```
>>> stemFunc=LancasterStemmer()
>>> vocabNew=[stemFunc.stem(w) for w in vocab]
>>> print(len(np.unique(vocabNew)))
13549
```

Análise dos Resultados

Representação tf-idf

O método tf-idf associa considera importantes palavras que aparecem muitas vezes em poucos documentos, e associa a estas um valor elevado. Podemos inspecionar quais palavras no caso da IMDb são as mais importantes. Convém realçar que a técnica tf-idf é não supervisionada, e o que considera "importante" não está necessariamente relacionado com críticas positivas ou negativas. De notar igualmente que a conversão dos textos para a representação tf-idf deve ser feita depois da limpeza dos documentos.

```
# Dados previamente limpos (sem stemming - total de 25626 palavras)
>>> tp=r'\b\w\w\w\w+\b'
>>> tfidf=TfidfVectorizer(min_df=5,stop_words='english',token_pattern=tp)
>>> vocab=tfidf.get_feature_names(text_train)
>>> Xtrain=tfidf.transform(text_train)
>>> max_x=Xtrain.max(axis=0).toarray().ravel()
>>> idx=np.argsort(-max_x)
>>> print([vocab[i] for i in idx[:40]])
[u'pokemon' u'scanners' u'steve' u'doodlebops' u'casper' u'sasquatch'
u'darkman' u'demons' u'xica' u'smallville' u'weller' u'sucks'
u'lennon' u'cypher' u'zatoichi' u'ichi' u'janeane' u'botched'
u'gadget' u'joan' u'priya' u'ants' u'naschy' u'worms' u'muppet'
u'zizek' u'blah' u'wrestlemania' u'pack' u'tanner' u'hackenstein'
u'ranma' u'seagal' u'keaton' u'gamera' u'khouri' u'alvin' u'lexi'
u'othello' u'beatles']
```

Análise dos Resultados

Representação tf-idf

O método tf-idf associa considera importantes palavras que aparecem muitas vezes em poucos documentos, e associa a estas um valor elevado. Podemos inspecionar quais palavras no caso da IMDb são as mais importantes. Convém realçar que a técnica tf-idf é não supervisionada, e o que considera "importante" não está necessariamente relacionado com críticas positivas ou negativas. De notar igualmente que a conversão dos textos para a representação tf-idf deve ser feita depois da limpeza dos documentos.

```
# Dados previamente limpos (sem stemming - total de 25626 palavras)
>>> tp=r'\b\w\w\w\w+\b'
>>> tfidf=TfidfVectorizer(min_df=5,stop_words='english',token_pattern=tp)
>>> vocab=tfidf.get_feature_names(text_train)
>>> Xtrain=tfidf.transform(text_train)
>>> max_x=Xtrain.max(axis=0).toarray().ravel()
>>> idx=np.argsort(-max_x)
```

De notar que muitas das palavras que o método tf-idf considerou "ìmportantes" estão relacionadas com títulos de filmes, o que por si não ajuda na tarefa de classificação de críticas em positivas ou negativas, mas contudo revela algumas características destes documentos.

Análise dos Resultados

Representação tf-idf

O método tf-idf associa considera importantes palavras que aparecem muitas vezes em poucos documentos, e associa a estas um valor elevado. Podemos inspecionar quais palavras no caso da IMDb são as mais importantes. Convém realçar que a técnica tf-idf é não supervisionada, e o que considera "importante" não está necessariamente relacionado com críticas positivas ou negativas. De notar igualmente que a conversão dos textos para a representação tf-idf deve ser feita depois da limpeza dos documentos.

```
# Dados previamente limpos (sem stemming - total de 25626 palavras)
>>> tp=r'\b\w\w\w\w+\b'
>>> tfidf=TfidfVectorizer(min_df=5,stop_words='english',token_pattern=tp)
>>> vocab=tfidf.get_feature_names(text_train)
>>> Xtrain=tfidf.transform(text_train)
>>> max_x=Xtrain.max(axis=0).toarray().ravel()
>>> idx=np.argsort(-max_x)
>>> print(vocab[idx[-40:]])
[u'immunity' u'swells' u'distort' u'ancestral' u'administered'
u'mistreatment' u'reverting' u'mclaughlin' u'basking' u'temperamental'
u'orientated' u'spacious' u'vertical' u'booed' u'backfire' u'slyly'
u'confidante' u'pressuring' u'manically' u'alloy' u'attained'
u'sylvain' u'hypocrites' u'nyree' u'galadriel' u'livelier' u'gliding'
u'auspicious' u'oncoming' u'coaxing' u'ware' u'inconsiderate'
u'uphold' u'emerald' u'cataclysmic' u'oversee' u'songwriting'
u'thieving' u'gauche' u'suplexes']
```

Análise dos Resultados

Representação tf-idf

O método tf-idf associa considera importantes palavras que aparecem muitas vezes em poucos documentos, e associa a estas um valor elevado. Podemos inspecionar quais palavras no caso da IMDb são as mais importantes. Convém realçar que a técnica tf-idf é não supervisionada, e o que considera "importante" não está necessariamente relacionado com críticas positivas ou negativas. De notar igualmente que a conversão dos textos para a representação tf-idf deve ser feita depois da limpeza dos documentos.

```
# Dados previamente limpos (sem stemming - total de 25626 palavras)
>>> tp=r'\b\w\w\w\w+\b'
>>> tfidf=TfidfVectorizer(min_df=5,stop_words='english',token_pattern=tp)
>>> vocab=tfidf.get_feature_names(text_train)
>>> Xtrain=tfidf.transform(text_train)
>>> max_x=Xtrain.max(axis=0).toarray().ravel()
>>> idx=np.argsort(-max_x)
```

Estas são as palavras que o método tf-idf considerou menos "ìmportantes" e tipicamente são aquelas que são habitualmente usadas num grande número de documentos ou que aparecem esparsamente em documentos compridos.

Análise dos Resultados

Representação tf-idf

O método tf-idf associa considera importantes palavras que aparecem muitas vezes em poucos documentos, e associa a estas um valor elevado. Podemos inspecionar quais palavras no caso da IMDb são as mais importantes. Convém realçar que a técnica tf-idf é não supervisionada, e o que considera "importante" não está necessariamente relacionado com críticas positivas ou negativas. De notar igualmente que a conversão dos textos para a representação tf-idf deve ser feita depois da limpeza dos documentos.

Podemos verificar quais são as palavras que ocorrem num maior número de documentos examinando o valor idf (inverse document frequency). As que apresentam um valor mais baixo são as menos importantes visto serem usadas em várias críticas.

```
>>> vocab=tfidf.get_feature_names(text_train)
>>> idf=tfidf.idf_
>>> idx=np.argsort(idf)
>>> print([vocab[i] for i in idx[:40]])
[u'movie' u'film' u'like' u'just' u'good' u'time' u'really' u'story'
u'great' u'people' u'make' u'watch' u'think' u'acting' u'movies'
u'seen' u'characters' u'plot' u'best' u'little' u'character' u'know'
u'better' u'life' u'films' u'does' u'love' u'scenes' u'watching'
u'scene' u'thing' u'real' u'years' u'doesn' u'actors' u'director'
u'makes' u'work' u'didn' u'look']
```

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Vários modelos de classificação e regressão implementam os tipos de regularização *Ridge* e *Lasso*. A escolha é feita através do parâmetro `penalty`. O peso dado ao termo de regularização é controlado pelo parâmetro `C`.
  - `penalty='l1'` regularização *Lasso*
  - `penalty='l2'` regularização *Ridge*
- Quanto maior for o valor de `C`, **menor** será a regularização.

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Ridge*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n]-\hat{y}[n])^2 + \lambda \mathbf{w}^\top \mathbf{w}$

```
# Dados previamente limpos (sem stemming - total de 25626 palavras)
>>> DesLog=LogisticRegression(penalty='l2',C=1,max_iter=5000,tol=1e-2)
>>> DesLog.fit(Xtrain,class_train)
>>> print('Acertos %.2f'%(DesLog.score(Xtrain,class_train)*100))
>>> print('Acertos %.2f'%(DesLog.score(Xtest,class_test)*100))
Acertos:  93.25 (treino)
Acertos:  87.54 (teste)
```

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Ridge*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n]-\hat{y}[n])^2 + \lambda \mathbf{w}^\top \mathbf{w}$

Ordenar as palavras pelo valor dos coeficientes.

```
>>> w=DesLog.coef_.ravel()
>>> idx=np.argsort(w)
```

Palavras associadas aos coeficientes com o menor valor:

```
>>> for i in idx[:30]:
        print('(%.2f) %s'%(w[i],vocab[i]))
```

```
1.(-9.72) worst
2.(-6.82) waste
3.(-6.78) awful
4.(-5.79) boring
5.(-5.47) poor
6.(-5.28) worse
7.(-4.98) terrible
8.(-4.56) poorly
9.(-4.53) horrible
10.(-4.28) dull
11.(-4.14) unfortunately
12.(-4.02) disappointment
13.(-3.96) annoying
14.(-3.93) script
15.(-3.86) fails
16.(-3.82) ridiculous
17.(-3.81) avoid
18.(-3.81) supposed
19.(-3.77) instead
20.(-3.77) minutes
21.(-3.64) save
22.(-3.63) stupid
23.(-3.63) mess
24.(-3.53) disappointing
25.(-3.50) badly
26.(-3.48) lame
27.(-3.43) pointless
28.(-3.24) unless
29.(-3.19) crap
30.(-3.13) just
```

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Ridge*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n]-\hat{y}[n])^2 + \lambda \mathbf{w}^\top \mathbf{w}$

Ordenar as palavras pelo valor dos coeficientes.

```
>>> w=DesLog.coef_.ravel()
>>> idx=np.argsort(w)
```

Palavras associadas aos coeficientes com o maior valor:

```
>>> for i in idx[-30:]:
        print('(%.2f) %s'%(w[i],vocab[i]))
```

```
30.(2.50) world
29.(2.51) heart
28.(2.52) hilarious
27.(2.52) unique
26.(2.55) funniest
25.(2.55) especially
24.(2.69) perfectly
23.(2.77) entertaining
22.(2.77) wonderfully
21.(2.91) rare
20.(2.94) enjoy
19.(3.00) liked
18.(3.03) enjoyable
17.(3.23) beautiful
16.(3.29) fantastic
15.(3.36) definitely
14.(3.48) superb
13.(3.50) highly
12.(3.57) brilliant
11.(3.69) today
10.(3.70) love
9.(3.71) enjoyed
8.(4.10) loved
7.(4.26) favorite
6.(4.34) amazing
5.(4.84) perfect
4.(4.94) wonderful
3.(5.20) best
2.(6.45) excellent
1.(7.29) great
```

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Lasso*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n]-\hat{y}[n])^2 + \lambda\sum_{i=0}^{d}|w_i|$
- Comparativamente ao método *Ridge*, o método de regularização *Lasso* tem a vantagem de eliminar as palavras menos discriminativas, pondo a zero o valor dos pesos correspondentes.

```
# Dados previamente limpos (sem stemming - total de 25626 palavras)
>>> DesLog=LogisticRegression(penalty='l1',solver='saga',C=10,max_iter=5000,tol=1e-2)
>>> DesLog.fit(Xtrain,class_train)
>>> print('Acertos %.2f'%(DesLog.score(Xtrain,class_train)*100))
>>> print('Acertos %.2f'%(DesLog.score(Xtest,class_test)*100))
Acertos:  99.56 (treino)
Acertos:  85.09 (teste)
```

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Lasso*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n] - \hat{y}[n])^2 + \lambda\sum_{i=0}^{d}|w_i|$
- Comparativamente ao método *Ridge*, o método de regularização *Lasso* tem a vantagem de eliminar as palavras menos discriminativas, pondo a zero o valor dos pesos correspondentes.

Palavras associadas aos coeficientes com o menor valor:

```
1.(-24.97) worst
2.(-21.97) octane
3.(-20.90) waste
4.(-19.99) lifeless
5.(-19.96) halloran
6.(-18.89) disappointment
7.(-18.65) poorly
8.(-18.24) awful
9.(-18.23) unremarkable
10.(-17.42) deprecating
11.(-17.18) unfunny
12.(-16.36) mess
13.(-16.35) forgettable
14.(-16.28) unlikeable
15.(-16.02) laughable
16.(-15.99) dabney
17.(-15.61) lacks
18.(-15.60) worse
19.(-15.41) pointless
20.(-15.40) thunderbirds
21.(-15.12) alleged
22.(-15.00) solely
23.(-14.78) fails
24.(-14.73) alberta
25.(-14.54) disappointing
26.(-14.35) mildly
27.(-14.20) dreck
28.(-14.15) refer
29.(-13.80) believer
30.(-13.72) schwimmer
```

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Lasso*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n] - \hat{y}[n])^2 + \lambda\sum_{i=0}^{d}|w_i|$
- Comparativamente ao método *Ridge*, o método de regularização *Lasso* tem a vantagem de eliminar as palavras menos discriminativas, pondo a zero o valor dos pesos correspondentes.

Palavras associadas aos coeficientes com o maior valor:

```
30.(13.18) extravagant
29.(13.20) superbly
28.(13.35) great
27.(13.35) rare
26.(13.37) piggy
25.(13.41) ridiculed
24.(13.50) cynics
23.(13.61) wiser
22.(13.68) captures
21.(13.91) meddling
20.(14.17) vengeance
19.(14.20) joyous
18.(14.21) commenters
17.(14.87) wonderfully
16.(14.89) photograph
15.(15.13) excellent
14.(15.76) shrieks
13.(16.03) perceptive
12.(16.17) flawless
11.(16.24) delightfully
10.(16.49) finely
9.(16.59) insides
8.(16.85) frills
7.(17.20) poisoning
6.(17.93) nitpick
5.(18.04) refreshing
4.(19.86) publish
3.(20.22) moodiness
2.(23.52) appreciable
1.(25.17) endearingly
```

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Lasso*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n]-\hat{y}[n])^2 + \lambda\sum_{i=0}^{d}|w_i|$
- Comparativamente ao método *Ridge*, o método de regularização *Lasso* tem a vantagem de eliminar as palavras menos discriminativas, pondo a zero o valor dos pesos correspondentes.

Número de coeficientes com valor diferente de zero:

```
>>> DesLog=LogisticRegression(penalty='l1',solver='saga',C=10,max_iter=5000,tol=1e-2)
>>> DesLog.fit(Xtrain,class_train)
>>> print('%d'%np.sum(DesLog.coef_!=0))
6663
```

- O vocabulário inicial contém 25626 palavras mas para este modelo só são necessárias 6663 palavras.
- Convém reduzir o valor do parâmetro `C` para aumentar a regularização.

Análise dos Resultados

Análise dos coeficientes dum discriminante logístico

Uma das vantagens de utilizar modelos lineares ou discriminantes logísticos em representações BoW é poder investigar quais palavras são mais relevantes para discriminar entre classes. Cada coeficiente do discriminante está associado a uma palavra e podemos inspecionar quais coeficientes têm o maior valor em termos absolutos. Neste contexto faz igualmente sentido usar um termo de regularização dos coeficientes, para minorar a possibilidade dos modelos entrarem em sobre-aprendizagem.

- Discriminante logístico com regularização *Lasso*: $\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(y[n]-\hat{y}[n])^2 + \lambda\sum_{i=0}^{d}|w_i|$
- Comparativamente ao método *Ridge*, o método de regularização *Lasso* tem a vantagem de eliminar as palavras menos discriminativas, pondo a zero o valor dos pesos correspondentes.

Número de coeficientes com valor diferente de zero:

```
>>> DesLog=LogisticRegression(penalty='l1',solver='saga',C=0.5,max_iter=5000,tol=1e-2)
>>> DesLog.fit(Xtrain,class_train)
>>> print('%d'%np.sum(DesLog.coef_!=0))
524
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Notar que a inclusão de n-gramas pode levar a um acréscimo significativo no número de entradas no vocabulário. Em principio, o número máximo de bi-gramas é igual ao número de palavras isoladas do vocabulário ao quadrado, o número de tri-gramas igual ao número de palavras isoladas ao cubo, e assim por diante. No entanto, devido à estrutura e características da linguagem escrita, o número de n-gramas é bastante inferior ao número máximo atingível, mas mesmo assim, continua a ser muito elevado.

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Podemos definir o número *n* dos n-gramas a considerar pelo modelo BoW com o parâmetro `ngram_range` das funções `CountVectorizer` ou `TfidfVectorizer`. O parâmetro `ngram_range` é um tuplo consistindo no comprimento mínimo e máximo da sequência de palavras a considerar.

```
# Dados previamente limpos (sem stemming - total de 26970 palavras isoladas)
>>> tp=r'\b\w\w+\b'
>>> tfidf=TfidfVectorizer(min_df=5,ngram_range=(1,2),token_pattern=tp)
>>> vocab=tfidf.get_feature_names(text_train)
>>> print(len(vocab))
154513
# contar o número de palavras em cada entrada do vocabulário
>>> wC=np.array([len(w.split() for w in vocab])
>>> print('Número de uni-gramas %d\n Número de bi-gramas'%(np.sum(wC==1),np.sum(wC==2)))
Número de uni-gramas:  26970
Número de bi-gramas:  127543
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Podemos definir o número *n* dos n-gramas a considerar pelo modelo BoW com o parâmetro `ngram_range` das funções `CountVectorizer` ou `TfidfVectorizer`. O parâmetro `ngram_range` é um tuplo consistindo no comprimento mínimo e máximo da sequência de palavras a considerar.

```
>>> tp=r'\b\w\w+\b'
>>> tfidf=TfidfVectorizer(min_df=5,ngram_range=(1,3),token_pattern=tp)
# contar o número de palavras em cada entrada do vocabulário
Tamanho vocabulário:  249609
Número de uni-gramas:  26970
Número de bi-gramas:  127543
Número de tri-gramas:  95096
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Podemos definir o número *n* dos n-gramas a considerar pelo modelo BoW com o parâmetro `ngram_range` das funções `CountVectorizer` ou `TfidfVectorizer`. O parâmetro `ngram_range` é um tuplo consistindo no comprimento mínimo e máximo da sequência de palavras a considerar.

```
>>> tp=r'\b\w\w+\b'
>>> tfidf=TfidfVectorizer(min_df=5,ngram_range=(1,4),token_pattern=tp)
# contar o número de palavras em cada entrada do vocabulário
Tamanho vocabulário:  280689
Número de uni-gramas:  26970
Número de bi-gramas:  127543
Número de tri-gramas:  95096
Número de 4-gramas:  31080
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Discriminante logístico com regularização *Lasso*.

```
# Vocabulário construido com uni-gramas e bi-gramas - total de 154513 entradas
>>> DesLog=LogisticRegression(penalty='l1',solver='saga',C=20,max_iter=5000,tol=1e-2)
>>> DesLog.fit(Xtrain,class_train)
>>> print('Acertos %.2f'%(DesLog.score(Xtrain,class_train)*100))
>>> print('Acertos %.2f'%(DesLog.score(Xtest,class_test)*100))
Acertos:  100.00 (treino)
Acertos:  88.58 (teste)
```

Número de coeficientes diferentes de 0:

```
>>> print('%d'%np.sum(DesLog.coef_!=0))
8998
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Uni-gramas e bi-gramas associados aos coeficientes de menor valor:

```
1.(-31.54) not worth
2.(-27.79) blue and
3.(-27.29) awful
4.(-27.29) not recommended
5.(-25.91) disappointment
6.(-25.33) poorly
7.(-23.96) had high
8.(-23.51) the introduction
9.(-22.34) let down
10.(-21.60) laughable
11.(-21.37) disappointing
12.(-21.21) forgettable
13.(-21.16) lacks
14.(-20.88) mess
15.(-20.22) not recommend
16.(-19.97) boring
17.(-19.90) might enjoy
18.(-19.50) it tries
19.(-19.47) wooden
20.(-19.46) pointless
21.(-19.30) unwatchable
22.(-19.27) uneducated
23.(-18.95) half hearted
24.(-18.65) not good
25.(-18.49) unfunny
26.(-18.34) sound format
27.(-18.33) worst
28.(-17.97) lousy
29.(-17.93) alright
30.(-17.93) wouldn recommend
31.(-17.82) are shot
32.(-17.63) marital
33.(-17.62) fu manchu
34.(-17.35) wayans
35.(-17.35) does look
36.(-17.29) dreck
37.(-17.23) horrible
38.(-17.18) doesn it
39.(-17.06) only good
40.(-17.03) incoherent
41.(-17.02) an unbelievable
42.(-16.99) fails
43.(-16.75) obnoxious
44.(-16.68) waste of
45.(-16.57) just wasn
46.(-16.56) badly
47.(-16.56) avoid
48.(-16.47) baldwin
49.(-16.30) jessica simpson
50.(-16.15) lifeless
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Uni-gramas e bi-gramas associados aos coeficientes de maior valor:

```
1.(30.56) definitely worth
2.(26.72) evil breed
3.(22.35) bridge and
4.(21.96) this great
5.(21.92) refreshing
6.(21.65) surprisingly good
7.(20.62) fez
8.(20.35) well worth
9.(20.31) wonderfully
10.(20.21) perfect
11.(19.72) endearingly
12.(19.51) surpasses
13.(19.06) thumbs up
14.(18.85) very entertaining
15.(18.21) unusually
16.(17.98) you only
17.(17.78) only complaint
18.(17.68) flawless
19.(17.17) it every
20.(17.03) cerebral
21.(16.57) cynics
22.(16.49) what like
23.(16.22) delightful
24.(16.09) not disappointed
25.(16.03) is unusual
26.(15.90) captures
27.(15.82) zero day
28.(15.69) moodiness
29.(15.57) delightfully
30.(15.41) on here
31.(15.28) sensitive
32.(15.23) rare
33.(15.18) wonderful
34.(15.16) subtle
35.(15.09) excellent
36.(15.00) likable and
37.(14.94) bad thing
38.(14.93) greene
39.(14.89) enjoyed this
40.(14.83) kitty
41.(14.70) excellently
42.(14.69) whoopi
43.(14.61) superb
44.(14.59) the technology
45.(14.57) powerful
46.(14.40) favorite of
47.(14.39) incredible
48.(14.39) marvel
49.(14.17) great job
50.(14.15) recommended
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Discriminante logístico com regularização *Lasso*.

```
>>> DesLog=LogisticRegression(penalty='l1',solver='saga',C=2,max_iter=5000,tol=1e-2)
>>> DesLog.fit(Xtrain,class_train)
>>> print('Acertos %.2f'%(DesLog.score(Xtrain,class_train)*100))
>>> print('Acertos %.2f'%(DesLog.score(Xtest,class_test)*100)
Acertos:  91.91 (treino)
Acertos:  89.31 (teste)
```

Número de coeficientes diferentes de 0:

```
>>> print('%d'%np.sum(DesLog.coef_!=0))
1678
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Uni-gramas e bi-gramas associados aos coeficientes de menor valor:

```
1.(-28.89) worst
2.(-26.07) awful
3.(-24.81) waste
4.(-22.54) poorly
5.(-20.77) disappointment
6.(-19.26) boring
7.(-19.21) not worth
8.(-18.20) lacks
9.(-17.29) laughable
10.(-17.11) mess
11.(-17.06) dull
12.(-17.00) fails
13.(-16.77) disappointing
14.(-16.50) poor
15.(-16.23) bad
16.(-16.19) worse
17.(-15.99) pointless
18.(-15.77) horrible
19.(-15.56) unfunny
20.(-15.55) at best
21.(-14.74) annoying
22.(-14.30) badly
23.(-14.26) terrible
24.(-14.05) avoid
25.(-14.01) unfortunately
26.(-13.42) forgettable
27.(-13.29) save
28.(-13.01) ridiculous
29.(-12.62) lame
30.(-12.28) not good
31.(-12.27) nothing
32.(-12.07) redeeming
33.(-11.73) oh
34.(-11.62) than this
35.(-11.24) wooden
36.(-10.83) supposed
37.(-10.77) basically
38.(-10.41) way too
39.(-10.35) mst
40.(-10.33) pathetic
41.(-10.30) lousy
42.(-10.18) script
43.(-10.16) not very
44.(-10.08) not recommend
45.(-10.01) stupid
46.(-9.94) pretentious
47.(-9.82) instead
48.(-9.70) let down
49.(-9.63) mildly
50.(-9.51) tedious
```

n-gramas

Representação BoW com mais de uma palavra

Uma das limitações da representação BoW é que descarta a informação sobre a ordem das palavras. As frases “não é bom, é mau” e “não é mau, é bom” têm a mesma representação apesar de terem significados opostos. Uma maneira de capturar (parte) da informação contextual é incluir na representação BoW sequências de duas ou mais palavras que habitualmente apareçam nos documentos. Conjuntos de duas palavras são denominados bi-gramas, de três palavras tri-gramas, e em geral, sequências de várias palavras são denominadas n-gramas.

Uni-gramas e bi-gramas associados aos coeficientes de maior valor:

```
1.(20.08) excellent
2.(17.94) perfect
3.(15.28) wonderfully
4.(15.18) wonderful
5.(14.73) great
6.(14.69) refreshing
7.(14.65) well worth
8.(14.20) amazing
9.(14.10) definitely worth
10.(13.23) favorite
11.(11.90) today
12.(11.85) rare
13.(11.67) must see
14.(11.66) enjoyable
15.(11.39) best
16.(11.16) brilliant
17.(10.83) superb
18.(10.14) funniest
19.(10.03) perfectly
20.(9.72) fantastic
21.(9.60) noir
22.(9.28) incredible
23.(9.10) touching
24.(9.09) fun
25.(9.06) loved this
26.(8.78) better than
27.(8.67) bit
28.(8.54) on dvd
29.(8.53) my only
30.(8.48) simple
31.(8.45) delightful
32.(8.41) surprisingly good
33.(8.25) loved
34.(8.23) highly
35.(8.13) fascinating
36.(7.92) subtle
37.(7.87) definitely
38.(7.82) enjoyed this
39.(7.74) gem
40.(7.67) very good
41.(7.67) love this
42.(7.66) captures
43.(7.64) beautiful
44.(7.59) believable
45.(7.54) very well
46.(7.50) pleasantly
surprised
47.(7.46) to all
48.(7.45) beautifully
49.(7.42) is great
50.(7.40) recommended
```