Trabalho Final Aprendizagem Automática

JOÃO BELAS N44105

PEDRO HENRIQUES N45415

1

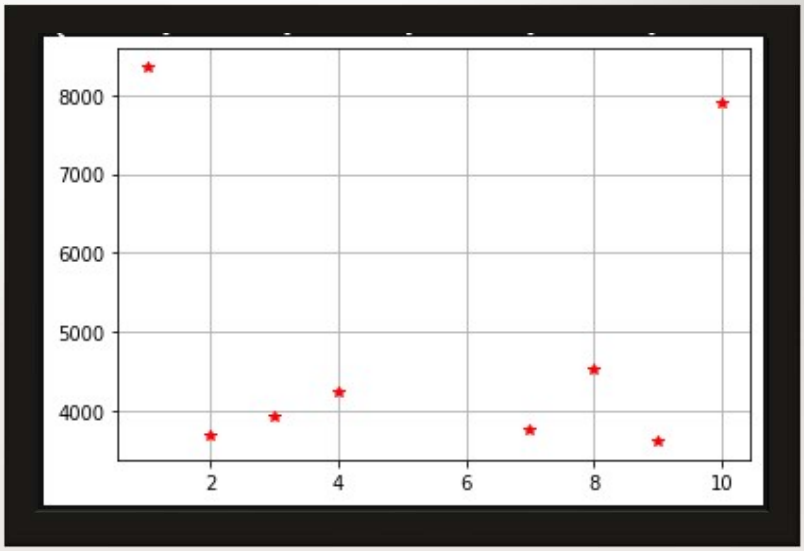
Introdução

- Este trabalho lida com textos de críticas de cinema do IMDb, e está dividido em duas tarefas de classificação. Baseado nos textos dos documentos, pretende-se:
  1. Determinar se a crítica é positiva ou negativa (classificação binária).
  2. Prever a pontuação da crítica (classificação multi-classe). Para tal considere que existem 8 classes, compostas pelas críticas com pontuações de 1-4 e de 7-10.

2

Quantidade de reviews por opção de review

- Maior concentração de críticas nos extremos.
- Distribuição não uniforme de críticas



3

Stemmer

- Um Stemmer reduz uma palavra flexionada à sua raiz linguística.
- Palavras tais como "universal", "university", e "universe" são reduzidas a "univers“. Pode levar a erros se as palavras não tiverem o mesmo sentido.
- Reduz a complexidade ortográfica dos documentos e elimina a maioria dos erros de escrita, atribuindo um único significado a múltiplas palavras.

```
Original
 Zero Day leads you to think, even re-think why two boys/young men would do what they did - commit mutual suicide via slaughtering their c
lassmates. It captures what must be beyond a bizarre mode of being for two humans who have decided to withdraw from common civility in ord
er to define their own/mutual world via coupled destruction.<br /><br />It is not a perfect movie but given what money/time the filmmaker
and actors had - it is a remarkable product. In terms of explaining the motives and actions of the two young suicide/murderers it is bette
r than 'Elephant' - in terms of being a film that gets under our 'rationalistic' skin it is a far, far better film than almost anything yo
u are likely to see. <br /><br />Flawed but honest with a terrible honesty.

Stemmed
 zero day lead you to think ev re think why two boy young men would do what they did commit mut suicid via slaught their classm it capt wh
at must be beyond a bizar mod of being for two hum who hav decid to withdraw from common civil in ord to defin their own mut world via cou
pl destruct it is not a perfect movy but giv what money tim the filmmak and act had it is a remark produc in term of explain the mot and a
ct of the two young suicid murd it is bet than eleph in term of being a film that get und our rat skin it is a far far bet film than almos
t anyth you ar lik to see flaw but honest with a terr honesty
```

4

Teste de stemmer Dimensão do vocabulário

```
print('R' ,40269/0.915175)
print('P' ,26773/0.945525)
print('S' ,26394/0.94535)
print('L' ,21883/0.939425)
```

```
R 44001.42049334827
P 28315.486105602708
S 27919.818056804357
L 23294.036245575753
```

- CONSTANTES:
  - TFIDFVECTORIZER(MIN_DF=3, TOKEN_PATTERN=R'\B[A-ZA-Z]{3,}\B')
  - LOGISTICREGRESSION(MAX_ITER = 1000, C=3.3, TOL = 1E-3)
- RESULTADO: ATRAVÉS DOS RESULTADOS DA DIVISÃO LEN_TOKENS/SCORE (PRINT AO LADO)
- PODEMOS DEDUZIR QUE O STEMMER LANCASTER É O MELHOR POIS SÃO PRECISOS MENOS VALORES PARA CHEGAR AO UM SCORE DE 100%. NO ENTANTO O STEMMER É O MAIS COMPUTACIONALMENTE DISPENDIOSO.
- É TAMBÉM POSSÍVEL VERIFICAR QUE MESMO COM MENOS TOKENS, UM STEMMER AUMENTA O A PRECISÃO DE PREVISÃO DO CLASSIFICADOR

5

Funções preProcessingDocs, text2vector, binClassify, multiClassify

```
def preProcessDoc(Doc, stemmer = 'lancaster', decode = False):
    stem = {
        'porter'   : PorterStemmer(),
        'snowball' : SnowballStemmer('english'),
        'lancaster': LancasterStemmer()
    }
    stemFunc = stem.get(stemmer, LancasterStemmer())
    if(decode):
        Doc = Doc.decode('UTF-8')
    Doc = Doc.replace('<br />', ' ')
    Doc = re.sub(r'[^a-zA-Z\u00C0\u00FF]+', ' ', Doc)
    Doc = ' '.join([stemFunc.stem(w) for w in Doc.split()])
    return Doc

def preProcessDocs(Docs, stemmer='lancaster', decode = False):
    return [preProcessDoc(doc, stemmer, decode) for doc in Docs]
```

- PreProcessingDocs: Limpa os documentos, aplicando os stemmers e removendo caracteres irrelevantes tais como acentos e tags html.
- Reduz a dimensão de tokens presentes nos documentos de textos não filtrados

6

Funções preProcessingDocs, text2vector, binClassify, multiClassify

- Text2vector:Função que converte uma lista de strings (variável Docs) na representação tf-idf.
- binClassify: Função que classifica a matriz X da função anterior em positivos e negativos.
- multiClassify: Função que classifica a matriz X devolvida pela função(text2vector) numa de oito classes (1-4 e 7-10).

```
def text2vector(Docs, tfidf=None,preProcess = False, stemmer='lancaster', decode=False):
    if(preProcess):
        Docs = preProcessDocs(Docs, stemmer=stemmer, decode=decode)

    if(tfidf==None):
        tfidf = pickle.load(open('tfidf_dump.p','rb'))

    return tfidf.transform(Docs)

def binClassify(X, clf=None):
    if(clf==None):
        clf = pickle.load(open('svc_bool_dump.p' , 'rb'))

    return clf.predict(X)

def multiClassify(X, clf=None):
    if(clf==None):
        clf = pickle.load(open('RidgeMultidump.p', 'rb'))
    return clf.predict(X)
```

7

Metodologia de Teste – Pipeline/GridSearch

- Pipeline é um modulo/classe do SKLearn que permite aplicar transformações sequencialmente a um conjunto de dados e estima a precisão das previsões com o classificador final. Essencialmente cria um classificador que pode também transformar e pré-processar os dados
- GridSearch é uma classe da biblioteca SKLearn que testa classificadores tendo em conta diversos parâmetros fornecidos para teste. Possui métodos que indicam tanto o tempo de execução de cada um dos classificadores testados, como a precisão de cada um destes.
- GridSearchCV é uma classe do SKLearn que modifica um pouco o comportamento de GridSearch. Nomeadamente executa os testes por Cross Validation e calcula a melhor pontuação para o classificador.
- RandomizedSearchCV é semelhante ao GridSearchCV, no entanto não testa todas as combinações possíveis e sorteia quais irá testar a partir das possíveis. Permite controlar o tempo de execução do programa, reduzindo o custo computacional. Por esse motivo foi o método de procura selecionado para modelos com diversos parâmetros a serem testados.

8

Metodologia de Teste - Pipeline

- Utilizamos a ferramenta pipeline para aplicar tanto o *tfidfVectorizer* como o *LogisticRegressionCV*. E determinar quais os melhores parâmetros para os nossos dados.
- Os parâmetros testados estão incluídos na variável grid_param e incluem se devemos incluir stop words, e n-gramas.
- Ao fazer fit, primeiro os documentos são transformados e depois é usado o LogisticRegression como classificador

```
pipeline =Pipeline([
    ('tfidf' , TfidfVectorizer()),
    ('clf' , LogisticRegressionCV(max_iter = 1000, tol=1e-3))
], memory=tmpdir, verbose=True)

grid_param ={
    'tfidf__strip_accents' :[None, 'unicode'],
    'tfidf__stop_words' : [None, 'english'],
    'tfidf__token_pattern' : [r'\b\w{3,}\b', r'\b[a-zA-Z]{3,}\b'],
    'tfidf__min_df' : np.arange(3, 6, 1),
    'tfidf__max_df' : [0.25, 0.5, 0.75],
    'tfidf__ngram_range' : [(1,1), (1,2), (1,3) , (2,2)] ,
    'tfidf__norm' : ['l1', 'l2'],

    #'clf__C' : np.linspace(0.1,10,100),
    'clf__Cs' : [1,3,10,30,100],
    #'clf__C' : [3.3],
    'clf__solver' : ['sag', 'saga'],
    #'clf__tol' : (1e-3, 1e-4, 1e-5)
}
t0 =time.localtime()
print('Started at'  , time.strftime("%H:%M:%S", t0))
## Se tiveres mais memoria que eu(8Gb), aumenta o pre_dispatch para um valor maior
grid_search = RandomizedSearchCV(pipeline, grid_param, cv = 5, n_jobs=-1, verbose=3,\
                                 pre_dispatch=8,n_iter = 30).fit(X, y_boolean)

t1 = time.localtime()
print('Done at' , time.strftime("%H:%M:%S", t1))
```


N-gramas

- Um N-grama corresponde a um token que é constituído por mais que 1 única palavra.
- A inclusão de n-gramas leva a uma explosão do número de tokens (De cerca de 25000 tokens para 175000)
- No entanto, também origina resultados consideravelmente melhores com os dados de teste

TFIDF

- Dado um conjunto de documentos (Strings), a matriz TF-IDF é uma matriz de (n_documentos, n_tokens) em que cada documento é representado pelo conjunto dos pesos nessa linha.
- Dado o número de amostras e quantidade de tokens, é representado por uma matriz esparsa.
- A matriz TF-IDF é automaticamente normalizada com valores entre [0,1] pelo que não é necessário proceder a outras normalizações.

11

Tfidf escolhido

- O tfidf escolhido teve em conta diversos critérios. Nomeadamente o resultado do GridSearch, número de tokens e precisão dos classificadores resultantes.

TFIDF optimizado

Não é exatamente o mesmo que foi eleito pelo pipeline, mas este remove stop words e palavras redundantes

```
7]: with open('tfidf_dump.p', 'wb') as f:
        tfidf = TfidfVectorizer(max_df=0.5 , min_df=4,ngram_range=(1, 2), norm='l2', token_pattern=r'\b\w{3,}\b'\
                                ,stop_words='english').fit(X)
        pickle.dump(tfidf, f)
```

12

Logistic Regression

- A regressão logística é um modelo matemático que procura produzir, a partir de um conjunto de treino, um modelo que seja capaz de prever uma variável para novos conjuntos de teste.
- No SKLearn estão implementadas as classes LogisticRegression, e LogisticRegressionCV. Este último é idêntico ao anterior, no entanto efetua o algoritmo *cross validation* para testar o classificador.
- Apesar de ter no nome Regression, comporta-se como um classificador pois as suas previsões são valores discretos contidos nos labels do conjunto de treino.

13

Logistic Regression CV

- O Logistic Regression é um classificador que aplica uma regressão logística a uma matriz de formato (n_amostras, n_componentes)
- Resultados train e teste:

```
1.0
0.9
```

- Resultados train e teste (multiclass):

```
0.7892333333333333
0.4406
```

É um classificador bastante bom para os dados em questão, tendo obtido resultados bastante semelhantes aos melhores resultados obtidos.
No entanto, não é tão rápido de executar, pelo que não foi escolhido.

14

Classificadores extra: LinearSVC

- Linear SVC é uma implementação do SKLearn de uma máquina de suporte vetorial, adaptado a um classificador.
- Uma máquina de suporte vetorial lida bastante bem com dados de elevada dimensão e com elevado número de amostras.
- A utilização LinearSVC é semelhante à utilização da classe SVC com kernel *'linear'*, a escolha do LinearSVC recaiu numa recomendação do SKLearn devido ao tipo de dados presentes.

15

Classificadores extra:

- Linear SVC:
  - dual=True (Por recomendação False se n_samples>n_features, tal não se verifica)
  - C=1 (parâmetro de regularização)
- Score train teste:
  - Multiclasse:

```
Linear SVC Score train:  0.85334375
Linear SVC Score test:  0.398375
```

  - Binária:

```
Linear SVC Score test:  0.9993
Linear SVC Score test:  0.9048
```

O LinearSVC foi o classificador que obteve melhores resultados na classificação binária. Além disso foi também dos classificadores mais rápidos de executar e com menor custo computacional.

16

Ridge Classifier

- O ridge classifier funciona de maneira semelhante ao Logistic Regression no sentindo em que formula um modelo matemático que tenta criar um modelo capaz de prever uma variável dependente através da introdução de variáveis independentes.
- O classificador normaliza as classes de treino para um espaço contido entre [-1, 1] e aplica uma regressão ridge que tenta minimizar a soma residual dos quadrados.

17

Classificadores extra:

- Ridge:
  - alpha=1 (força de regularização)
- Scores train e teste:
  - Multiclasse:

```
Ridge Score train:  0.8778524048714442
Ridge Score test:  0.6617789821386195
```

  - Binária:

```
Ridge Score train:  0.8697126432829163
Ridge Score test:  0.6363946566089572
```

18

Comparação Classificadores

- Diferentes classificadores adequam-se a diferentes tipos de dados
- Os nossos dados são esparsos e de elevada dimensão, assim sendo o classificador que melhor se adequa é o SVM. No entanto este não foi o que obteve melhores resultados. Após a normalização por recurso à classe Normalizer o SVM apresentou resultados um pouco melhores, na ordem dos 1% a 2%
- Ainda assim devido à velocidade de execução do algoritmo este foi o escolhido.

19

Análise de Componentes Principais (PCA)

- Análise em componentes principais é um procedimento matemático onde uma matriz é transformada de modo a que a sua primeira coluna possua os dados com a maior variância do conjunto.
- PCA permite reduzir um conjunto de dados com N fatores para um conjunto com uma dimensão muito mais reduzida. Se bem que a custo de alguma variância dos dados originais.

20

Adicional (PCA):

- TruncatedSVD:
  - n_components=100 (nº componentes desejados no output)
    - Variância total = sum(pca.explained_variance_ratio_): `0.0530847474352232`
    - Escolhemos os 100 valores por recomendação do SKLearn para tratamento de dados de texto.
  - Ciclo (n_components= [10,20,30,50,100,200,300,500], n_iter=2):

```
Num Componentes        10 ||  Variance  0.0145753481
Num Componentes        20 ||  Variance  0.0210284684
Num Componentes        30 ||  Variance  0.0261841245
Num Componentes        50 ||  Variance  0.0347922198
Num Componentes       100 ||  Variance  0.0521353079
Num Componentes       200 ||  Variance  0.0791129334
Num Componentes       300 ||  Variance  0.1008270827
Num Componentes       500 ||  Variance  0.1364963751
```


Adicional (PCA):

- LogisticRegressionCV(Cs=3, max_iter=1000, multi_class='auto', n_jobs=-1, penalty='l2', solver='saga', tol=0.001):
  - Multiclasse:

```
0.43515
Class 1
Scores
[[0.208875 0.417125 0.419875]
 [0.208875 0.4245   0.4295  ]
 [0.20875  0.42525  0.427125]
 [0.20875  0.421125 0.43175 ]
 [0.20875  0.422    0.42975 ]]

Class 4
Scores
[[0.208875 0.417125 0.419875]
 [0.208875 0.4245   0.4295  ]
 [0.20875  0.42525  0.427125]
 [0.20875  0.421125 0.43175 ]
 [0.20875  0.422    0.42975 ]]

Class 7
Scores
[[0.208875 0.417125 0.419875]
 [0.208875 0.4245   0.4295  ]
 [0.20875  0.42525  0.427125]
 [0.20875  0.421125 0.43175 ]
 [0.20875  0.422    0.42975 ]]

Class 10
Scores
[[0.208875 0.417125 0.419875]
 [0.208875 0.4245   0.4295  ]
 [0.20875  0.42525  0.427125]
 [0.20875  0.421125 0.43175 ]
 [0.20875  0.422    0.42975 ]]
```

  - Binária:

```
0.86865
Class 1
Score [[0.50525  0.859125 0.864875]
 [0.50525  0.864625 0.864875]
 [0.50525  0.863375 0.86525 ]
 [0.50525  0.870375 0.871125]
 [0.50525  0.863    0.865125]]
```

TruncatedSVD

- Um número maior de componentes exige um esforço computacional maior, mas origina melhores resultados.
- Por recomendação presente na documentação do SKLearn, escolhemos manter apenas os 100 componentes principais, apesar de possuir apenas 5% da variância total dos dados.

```
pca = TruncatedSVD(n_components=100).fit(vector)
```

```
sum(pca.explained_variance_ratio_)
```

```
0.05308359524560884
```

```
with open('pca_dump.p' ,'wb') as f:
    pickle.dump(pca , f)
```

```
list = [10,20,30,50,100,200,300,500]

for n_comps in list:
    pca = TruncatedSVD(n_components=n_comps, n_iter=2).fit(vector)
    print(f'Num Componentes {n_comps : >10} ||  Variance {pca.explained_variance_ratio_.sum() : .10f}')
```

```
Num Componentes         10 ||  Variance  0.0099217084
Num Componentes         20 ||  Variance  0.0163094359
Num Componentes         30 ||  Variance  0.0214984490
Num Componentes         50 ||  Variance  0.0305247506
Num Componentes        100 ||  Variance  0.0489808437
Num Componentes        200 ||  Variance  0.0780221481
Num Componentes        300 ||  Variance  0.1020260951
Num Componentes        500 ||  Variance  0.1415033827
```

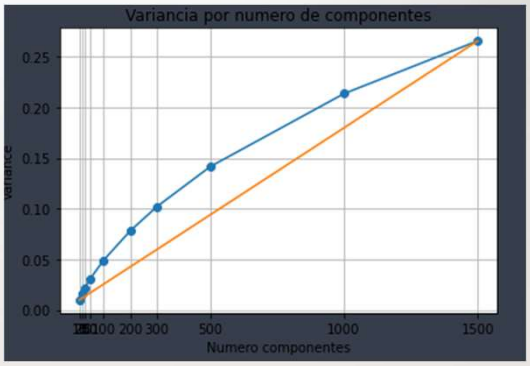

Comparação Resultados PCA – Sem PCA

- Tendo apenas 100 componentes o classificador teve uma precisão de ~87% (classificação booleana conjunto de teste, classificador logístico)
- Com todos os 178 000 componentes o classificador obteve uma precisão de ~90% (classificação booleana conjunto de teste, classificador logístico)

- Uma fração de 1/1780 no número de componentes leva a resultados semelhantes aos originais com um custo computacional muito menor.
- Assim podemos admitir que a introdução de erros na previsão é compensada pela velocidade de execução e menor complexidade computacional.

Aproximação de número ótimo de componentes

- Através dos testes efetuados anteriormente obtivemos os resultados presentes à direita.
- O número máximo de componentes corresponde ao ponto anterior aquele em que o programa deixaria de funcionar por falta de memória.
- Tendo em conta a curva obtida o ponto onde se encontra um equilíbrio computacional e os melhores resultados é aquele que possui maior distância à reta a laranja (método elbow).
- Número de componentes ótimos é igual a cerca de 500.



25

Clustering - KMeans

- Clustering é um técnica de aprendizagem automática não supervisionada que divide um vetor em grupos (clusters) com semelhanças entre si.
- O KMeans é um método de Clustering que procura separar os dados em grupos com igual variância entre eles, minimizando a inércia (soma dos quadrados) através da formula $\sum_{i=0}^{n} min_{\mu_j \in C}(|x_i - \mu|)$
- O KMeans sofre bastante para conjuntos com dimensões elevadas, pelo que é necessário reduzir as dimensões através de PCA.

26

Clustering – Descobrir o número de clusters

- Na imagem ao lado vemos o plot da inércia dos dados em função do número de clusters
- Segundo o "*Elbow method*", o número ideal de cluster, corresponde ao ponto que possua a maior distância à linha a laranja.
- O "cotovelo" da curva corresponde ao ponto em que os ganhos, nomeadamente na inércia, já não justificam o custo adicional.
- O número escolhido foi 37 clusters, para um conjunto de dados com 100 componentes

27

Clusters

- Os clusters correspondem a tópicos comuns presentes nos documentos.
- Podemos verificar que alguns clusters que têm tópicos bem definidos tais como:
  - Cluster 1 – Ambiente/Tema geral do filme
  - Cluster 3 – Recomendar a visualização
  - Cluster 30 – Emoções contidas na crítica
- Outros possuem tópicos mais gerais, fornecendo contexto ao documento

28

Conclusões

- Melhor Stemmer e o utilizado:Lancaster
- Tfidf otimizado: TfidfVectorizer(max_df=0.5 , min_df=4,ngram_range=(1, 2), norm='l2', token_pattern=r'\b\w{3,}\b' ,stop_words='english')
- Logistic Regression:LogisticRegressionCV(max_iter = 1000, Cs=3, tol = 1e-3, cv=5,penalty='l2', solver='saga')
- Classificador escolhido entre os testados: LinearSVC
- PCA: reduzido o número de componentes para 100 com uma escala de 1/1780 pois os resultados são semelhantes(100->87%,178000->90%)
- Aproximação do nº ótimo de componentes:500
- Para um conjunto de 100 componentes foram criados 37 clusters

29

Bibliografia

- Slides fornecidos pelo docente.
- Documentação oficial SKLearn disponível em :
  - https://scikit-learn.org/stable/user_guide.html
  - Notavelmente os seguintes
    - https://scikit-learn.org/stable/modules/grid_search.html
    - https://scikit-learn.org/stable/modules/compose.html#combining-estimators
- Blogs Sobre o tema, notavelmente:
  - - https://medium.com/@wenxuan0923/feature-extraction-from-text-using-countvectorizer-tfidfvectorizer-9f74f38f86cc
  - - https://www.datascienceassn.org/sites/default/files/users/user1/lsa_presentation_final.pdf
  - - https://medium.com/hanman/data-clustering-what-type-of-movies-are-in-the-imdb-top-250-7ef59372a93b
  - - https://jtemporal.com/kmeans-and-elbow-method/

30