Curso de Especialização de Inteligência Artificial Aplicada

Setor de Educação Profissional e Tecnológica - SEPT

Universidade Federal do Paraná - UFPR

IAA003 - Linguagem de Programação Aplicada

Prof. Alexander Robert Kutzke

Estudante: Clístenes Grizafis Bento

Implementação com Scikit-Learn

Utilizando a base de dados presente no repositório:

1. Escreva *pipeline de classificação de texto* para classificar reviews de filmes como positivos e negativos;
2. Encontre um bom conjunto de parâmetros utilizando `GridSearchCV` ;
3. Avalie o classificador utilizando parte do conjunto de dados (previamente separado para testes).
4. Repita os passos 1, 2 e 3 utilizando um algoritmo de classificação diferente;
5. Escreva um pequeno texto comparando os resultados obtidos para cada algoritmo.

O texto pode ser escrito em um "Jupyter Notebook" juntamente com o código. Ou qualquer outro tipo de documento.

1 Escreva pipeline de classificação de texto para classificar reviews de filmes como positivos e negativos

1.1 Importando Bibliotecas

In [14]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_files
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import GaussianNB, MultinomialNB
import numpy as np
```

1.2 Iniciando o método principal e importando os dados

In [15]:
```
if __name__ == "__main__":
    movie_reviews_data_folder = r"./data"
    dataset = load_files(movie_reviews_data_folder, shuffle=False)
    print("n_samples: %d" % len(dataset.data))
```

```
n_samples: 2000
```

1.3 Separando dados de treino e de teste

In [16]:
```
docs_train, docs_test, y_train, y_test = train_test_split(
        dataset.data, dataset.target, test_size=0.25, random_state=None)
```

1.4 Criando Pipeline de classificação utilizando algoritmo MultinomialNB

In [17]:
```
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', MultinomialNB()),
    ])
```

2 Encontre um bom conjunto de parâmetros utilizando GridSearchCV

2.1 Aplicando GridSearchCV

In [18]:
```
parameters = {'vect__ngram_range': [(1, 1), (1, 2)],
                  'tfidf__use_idf': (True, False),
                  'clf__alpha': (1e-2, 1e-3),
                  }
```

In [19]:
```
gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1)
```

In [20]:
```
gs_clf.fit(docs_train, y_train)
```

Out[20]:
```
GridSearchCV(estimator=Pipeline(steps=[('vect', CountVectorizer()),
                                       ('tfidf', TfidfTransformer()),
                                       ('clf', MultinomialNB())]),
             n_jobs=-1,
             param_grid={'clf__alpha': (0.01, 0.001),
                         'tfidf__use_idf': (True, False),
                         'vect__ngram_range': [(1, 1), (1, 2)]})
```

2.2 Verificando o melhor resultado

In [21]:
```
gs_clf.best_score_
```

Out[21]:
```
0.818
```

2.3 verificando um bom conjunto de parâmetros

In [22]:
```
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, gs_clf.best_params_[param_name]))
```

```
clf__alpha: 0.01
tfidf__use_idf: False
vect__ngram_range: (1, 2)
```

3 Avalie o classificador utilizando parte do conjunto de dados (previamente separado para testes)

3.1 Realizando teste

In [23]:
```
predicted = gs_clf.predict(docs_test)
```

3.2 Imprimindo avaliação de classificação

In [24]:
```
print(metrics.classification_report(y_test, predicted,
                                    target_names=dataset.target_names))
```

```
              precision    recall  f1-score   support

         neg       0.86      0.84      0.85       262
         pos       0.82      0.84      0.83       238

    accuracy                           0.84       500
   macro avg       0.84      0.84      0.84       500
weighted avg       0.84      0.84      0.84       500
```

3.3 Imprimindo matrix de confusão

In [25]:
```
cm = metrics.confusion_matrix(y_test, predicted)
print(cm)
```

```
[[219  43]
 [ 37 201]]
```

4 Repita os passos 1, 2 e 3 utilizando um algoritmo de classificação diferente;

4.1 Importando novo algoritmo escolhido

In [26]:
```
from sklearn.linear_model import SGDClassifier
```

4.2 Criando Pipeline

In [27]:
```
text_clf2 = Pipeline([('vect', CountVectorizer()),
                      ('tfidf', TfidfTransformer()),
                      ('clf', SGDClassifier(loss='hinge', penalty='l2',
                                            alpha=1e-3, random_state=42,
                                            max_iter=5, tol=None)),
    ])
```

4.3 Aplicando GridSearchCV

In [28]:
```
gs_clf2 = GridSearchCV(text_clf2, parameters, n_jobs=-1)

gs_clf2.fit(docs_train, y_train)
```

Out[28]:
```
GridSearchCV(estimator=Pipeline(steps=[('vect', CountVectorizer()),
                                       ('tfidf', TfidfTransformer()),
                                       ('clf',
                                        SGDClassifier(alpha=0.001, max_iter=5,
                                                      random_state=42,
                                                      tol=None))]),
             n_jobs=-1,
             param_grid={'clf__alpha': (0.01, 0.001),
                         'tfidf__use_idf': (True, False),
                         'vect__ngram_range': [(1, 1), (1, 2)]})
```

4.4 Verificando o melhor resultado

In [29]:
```
gs_clf2.best_score_
```

Out[29]:
```
0.826
```

4.5 verificando um bom conjunto de parâmetros

In [30]:
```
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, gs_clf2.best_params_[param_name]))
```

```
clf__alpha: 0.001
tfidf__use_idf: True
vect__ngram_range: (1, 1)
```

4.6 Realizando teste

In [34]:
```
predicted2 = gs_clf2.predict(docs_test)
```

4.7 Imprimindo avaliação de classificação¶

In [35]:
```
print(metrics.classification_report(y_test, predicted2,
                                    target_names=dataset.target_names))
```

```
              precision    recall  f1-score   support

         neg       0.83      0.89      0.85       262
         pos       0.86      0.79      0.83       238

    accuracy                           0.84       500
   macro avg       0.84      0.84      0.84       500
weighted avg       0.84      0.84      0.84       500
```

4.8 Imprimindo matrix de confusão

In [37]:
```
cm = metrics.confusion_matrix(y_test, predicted2)
print(cm)
```

```
[[232  30]
 [ 49 189]]
```

5 Escreva um pequeno texto comparando os resultados obtidos para cada algoritmo.

Ao avaliar a classificação e a matrix de confusão dos dois algoritmos, foi possível perceber os resultados não foram muito diferentes para o treinamento de 75% dos dados deixando 25% para testes. Ainda não tenho conhecimento suficiente para avaliar se os resultados obtidos são satisfatórios, mas aparentemete os resultados foram bons.