Classificação textual através da representação de Bag-of-Words e do algoritmo Naïve Bayes

André Scheibel de Almada - 9292952
Leonardo Alves Gomes - 9293178

Considerações Iniciais

Utilização da distribuição de ciência de dados **Anaconda**.

Uso das bibliotecas **scikit-learn**, **Pandas** e **NLTK**.

Weka.

Pré-Processamento

Retirada de caracteres não alfa-numéricos.

Caracteres na maiúscula passaram a estar na minúscula.

Tokenização (NLTK)

Retirada de stopwords (NLTK)

Stemização (NLTK)

```
stemmer = PorterStemmer()
# Lê o texto de um arquivo, fazendo os devidos pré-processamentos
def get_data (filename):
    with open(filename, 'r', encoding='latin1') as f:
        corpus = f.read() # leitura do corpus do arquivo
        corpus = re.sub('[^A-Za-z]', ' ', corpus) # Retira caracteres não alfanumericos
        corpus = corpus.lower() # Torna todas as palavras minúsculas

        corpus = word_tokenize(corpus) # tokenização do corpuso
        # remoção das stopwords
        for token in corpus:
            if token in stopwords.words('english'):
                corpus.remove(token)

        for i in range(len(corpus)): # Processo de Stemming
            corpus[i] = stemmer.stem(corpus[i])

        plain_corpus = " ".join(corpus) # Transforma o array de tokens em uma string única por artigo
        return plain_corpus

# Pega o tema do artigo através do nome do arquivo
def get_class (filename):
    return filename.split('-')[0].split('\\')[1]
```

Bag-of-Words

CountVectorizer (scikit-learn) com os 1000 termos mais frequentes.

Criação de uma matriz termo-documento (Pandas).

```
print("transformando textos em matriz termo-documento...")
# transformação em matriz de termo/documento
cv = CountVectorizer(max_features=1000, encoding='latin1')

X = cv.fit_transform(dataset.data).toarray() # Bag of words de cada artigo
words = cv.get_feature_names() # palavras presentes em X
y = dataset.theme # Classificação dos artigos

# matriz de termo/documento com coluna de classe
tdm = pd.concat([y, pd.DataFrame(X, columns=words)], axis=1)
```

Naïve-Bayes

Utilização do estimador de Laplace (com valor 0.1).

```python
# determina a classe de um documento utilizando naive bayes
def bayes(document, train_set, themes):
    # dicionário que guarda as probabilidades de cada tema
    probabilities = {}

    # calcula a probabilidade condicional (parcial) de cada tema
    for theme in themes:
        # separa a porção do dataset associada ao tema atual
        theme_set = train_set[train_set.theme == theme]
        # calcula a probabilidade do tema atual (P(A))
        theme_prob = (len(theme_set) / len(train_set))

        # calcula a probabilidade condicional para cada palavra (P(B|A))
        for word in theme_set.columns[1:]:
            theme_prob *= ((len(theme_set[theme_set[word] == document[word]]) + 0.1) /
                           (len(theme_set) + 0.1))

        # guarda o valor no dicionário de probabilidades
        probabilities[theme] = theme_prob

    # retorna a entrada no dicionário
    return max(probabilities, key=lambda k: probabilities[k])
```

Fase de Testes

Validação cruzada 10-folds (scikit-learn).

Embaralhamento das partições ativado.

```
# gerador de 'folds' normalizadas para um k-fold cross-validation
n_splits = 10 # quantidade de folds
skf = StratifiedKFold(n_splits=n_splits, shuffle=True)

accuracy = 0
# criação do conjunto de temas possíveis
themes = list(set(tdm.theme))
for train_index, test_index in skf.split(X, y):
    # separação entre teste e treinamento
    train_set = tdm.iloc[train_index]
    test_set = tdm.iloc[test_index]

    correct_count = 0 # contagem de exemplos corretamente previstos
    for index, doc in test_set.iterrows():
        # calcula a predição para um exemplo do conjunto de tetes
        pred = bayes(doc, train_set, themes)
        # incrementa contagem de previsões corretas caso esta seja correta
        correct = (pred == doc.theme)
        correct_count += 1 if correct else 0

    correct_count /= len(test_set)

    # adiciona à acurácia total a acurácia obtida neste fold
    accuracy += correct_count / len(test_set)

accuracy /= n_splits

print (accuracy)
```

Resultado

Caractere ‘x’ indica erro na predição.

Caractere ‘.’ indica acerto na predição.

```
 fold 0 (58 itens):
.....................................................
fold 0 accuracy: 1.0
fold 1 (58 itens):
..x.........................x.....................x.x....
fold 1 accuracy: 0.9310344827586207
fold 2 (58 itens):
.................x..x.....................x....x..........
fold 2 accuracy: 0.9310344827586207

...


fold 7 (57 itens):
.x....................................................
fold 7 accuracy: 0.9824561403508771
fold 8 (57 itens):
.......................................xx..............
fold 8 accuracy: 0.9649122807017544
fold 9 (55 itens):
.....................................x..x.....x.....x.
fold 9 accuracy: 0.9272727272727272

0.95464499807
```

Comparações com outros algoritmos

Naïve-Bayes	J48 (Weka)	KNN (Weka)	SMO (Weka)	Naïve-Bayes (Weka)
95.4645 %	94.7735 %	92.6829 %	99.3031 %	98.4321 %