Tut 6: Phân loại văn bản sử dụng Support Vector Machine (SVM)

Ứng dụng SVM vào việc phân loại văn bản dựa trên vector tf-idf

- Học viên sẽ được yêu cầu sử dụng SVM vào tập dữ liệu đã lấy về từ Lab 1.

Trong bài hướng dẫn này, chúng ta sẽ làm các bước sau

- Lấy nội dung tập dữ liệu và các thể loại
- Trích xuất các vecto đặc trưng phù hợp cho học máy
- Huấn luyện dữ liệu với SVM để thực hiện phân loại
- Xử dụng Gird Search để điều chỉnh hiệu xuất

Importing Dataset

- Trước tiên, chúng ta cần tải dữ liệu 20newsgroup về, dữ liệu này chúng ta đã làm quen ở những bài trước. Để lấy dữ liệu này, chúng ta thực hiện dòng lệnh sau:

In [1]:
```
# Import tập dữ liệu từ thư viện của sklearn

from sklearn.datasets import fetch_20newsgroups

# Sau đó ta tiến hành load dữ liệu, chia dữ liệu train, test, tạo vector class với
4 phần tử tương ứng với 4 chủ đề:
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train', random_state=42, categories=categ
ories)
train_label = twenty_train.target
twenty_test = fetch_20newsgroups(subset='test', random_state=42, categories=categor
ies)
test_label = twenty_test.target
len(twenty_train.data), len(twenty_test.data), len(test_label)
```

Out[1]:
```
(2257, 1502, 1502)
```

Extracting features from text files

- Ở bước tiếp theo, chúng ta cần trích xuất các tính năng từ văn bản (Extracting features from text files). Để thực hiện học máy trên các tài liệu văn bản, trước tiên chúng ta cần biến nội dung văn bản thành các vectơ đặc trưng số học.
- Chúng ta sử dụng thư viện TfidfVectorizer để tạo vector tf-idf mà ta đã làm quen ở Lab 1

TfidfVetorizer

- Chúng ta cần phải loại bỏ các stopword

In [2]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = tfidf_vectorizer.fit_transform(twenty_train.data)
X_train_tfidf.shape
```

Out[2]: (2257, 35482)

Building a Pipeline

- Để thực hiện quá trình vectorizer => transformer => classifier dễ hơn, scikit-learn cung cấp một lớp Pipeline hoạt động giống như một mô hình phân loại tổng hợp

In [3]:
```
# Tiến hành import các thư viện cần thiết
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
```

In [4]:
```
 # Tạo Pineline và fit model vào. Ở đây chúng ta sẽ dụng kernel là Radial Basis Fun
ction (rbf). Một trong các kernel thường được sử dụng của SVM

text_clf = Pipeline([('vect', TfidfVectorizer()),
('clf', SVC(kernel='rbf', random_state=0, gamma=1, C=1))]) # Khi sử dụng SVM với ke
rnel RBF có hai tham số cần được thiết lập trước đó là tham số C và tham số γ.
# Tham khảo tham số C và γ theo đường link sau https://chrisalbon.com/machine_learn
ing/support_vector_machines/svc_parameters_using_rbf_kernel/
```

In [5]:
```
text_clf.fit(twenty_train.data, twenty_train.target)
```

Out[5]:
```
Pipeline(memory=None,
     steps=[('vect', TfidfVectorizer(analyzer='word', binary=False, decode_error
='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
  ...bf',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False))])
```

Evaluation

In [6]:
```
# Kiểm tra bằng metric

from sklearn import metrics
predicted = text_clf.predict(twenty_test.data)
print(metrics.classification_report(twenty_test.target, predicted,
target_names=twenty_test.target_names))
```

```
                        precision    recall  f1-score   support

           alt.atheism       0.97      0.77      0.86       319
         comp.graphics       0.84      0.96      0.89       389
               sci.med       0.91      0.87      0.89       396
soc.religion.christian       0.88      0.94      0.91       398

           avg / total       0.90      0.89      0.89      1502
```

In [7]:
```
# Confusion matrix

metrics.confusion_matrix(twenty_test.target, predicted)
```

Out[7]:
```
array([[247,  14,  16,  42],
       [  2, 374,   6,   7],
       [  2,  48, 343,   3],
       [  3,  11,  11, 373]], dtype=int64)
```

In [8]:
```
metrics.accuracy_score(twenty_test.target, predicted)
```

Out[8]:
```
0.8901464713715047
```

Parameter tuning sử dụng Grid Search

- Chúng ta tạo một danh sách các tham số mà chúng ta muốn thực hiện điều chỉnh hiệu suất. Chúng ta sử dụng phương pháp tìm kiếm lưới (grid search) để xác định các tham số C và γ tối ưu cho từng tập dữ liệu cụ thể

In [9]:
```
# Import thư viện và tạo paramater
from sklearn.model_selection import GridSearchCV
parameters =  {'vect__ngram_range': [(1, 1), (1, 2)],
'vect__use_idf': (True, False),
'clf__C': (1,10,100,1000),'clf__gamma': (1,2,3,'auto')
}
```

In [10]:
```
# Khởi tạo model GridSearchCV

# Ta cung cấp cho tham số này giá trị -1, grid search sẽ phát hiện có bao nhiêu lõi
được cài đặt và sử dụng tất cả chúng

gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1)
```

In [11]:
```
# Fit training data vào:
gs_clf = gs_clf.fit(twenty_train.data, twenty_train.target)
```

In [12]:
```
# Các thuộc tính của best_score_ và best_params_ lưu trữ giá trị trung bình tốt nhấ
t và cài đặt tham số tương ứng với điểm số đó

gs_clf.best_params_
```

Out[12]:
```
{'clf__C': 10,
 'clf__gamma': 1,
 'vect__ngram_range': (1, 1),
 'vect__use_idf': True}
```

In [13]:
```
gs_clf.best_score_
```

Out[13]:
```
0.9579087284005317
```

In [14]:
```
# Lúc này, ta đã có model và có thể sử dụng để dự đoán

twenty_train.target_names[gs_clf.predict(['OpenGL on the GPU is fast'])[0]]
```

Out[14]:
```
'comp.graphics'
```

- Học viên thử thay đổi các tham số C và γ và sau đó so sánh kết quả.