File overview

This notebook implements **Support Vector Machines (SVM)** classification for the subreddit prediction dataset. Hyperparameter tuning is performed, and the model's accuracy is evaluated using **10-fold cross-validation**.

Load modules

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore", category=UserWarning)  # This will
suppress UserWarnings



import time

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline


import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords, words

# Ensure required NLTK resources are downloaded
try:
    nltk.download('punkt')
    nltk.download('stopwords')
    nltk.download('words')

except Exception as e:
    print(f"Error downloading NLTK resources: {e}")

# Define stopwords list
specific_stopwords = ["https", "subreddit", "www", "com"] ## some
specific words for the given dataset
stopwords_list = stopwords.words('english') +specific_stopwords +
stopwords.words('french') # dataset is both in english and in french
```

```
[nltk_data] Downloading package punkt to /home/clatimie/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/clatimie/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package words to /home/clatimie/nltk_data...
[nltk_data]   Package words is already up-to-date!
```

Load training dataset

```
# Define the path to the training data file
path_training = "../datasets/Train.csv"

# Read the CSV file into a pandas DataFrame
training_data = pd.read_csv(path_training, delimiter=',')

# Set column names explicitly for better readability
training_data.columns = ['text', 'subreddit']

# Shuffle dataset
training_data = training_data.sample(frac=1,
random_state=42).reset_index(drop=True)

# Separate the training data into two series: texts and subreddit
labels
x_train = training_data['text']          # Contains the Reddit posts
or comments
y_train = training_data['subreddit'] # Contains the subreddit each
post originates from

# Get unique subreddit labels
unique_labels = np.unique(y_train)   # List of unique subreddits in
the dataset

n_samples_training = x_train.shape[0]
n_classes = unique_labels.shape[0]

print(f"Training dataset has {n_samples_training} examples and there
are {n_classes} classes")
```

```
Training dataset has 1399 examples and there are 4 classes
```

Load test dataset

```
# Define the path to the training data file
path_test = "../datasets/Test.csv"

# Read the CSV file into a pandas DataFrame
x_test = pd.read_csv(path_test, delimiter=',')["body"]
```

```python
n_samples_test = x_test.shape[0]
print(f"Test dataset has {n_samples_test} examples")
```

```
Test dataset has 600 examples
```

Lemma Tokenizer from NLTK

```python
class LemmaTokenizer:
    def __init__(self, stopwords=None):
        self.wnl = WordNetLemmatizer()
        self.stop_words = stopwords

    def __call__(self, doc):
        # Tokenize the document and apply lemmatization and filtering
        return [
            self.wnl.lemmatize(t, pos="v") for t in word_tokenize(doc)
            if t.isalpha() and t.lower() not in self.stop_words]
```

Hyperparameters search

```python
""" # Define the parameter grid for hyperparameter search
param_grid = {
    'svc__kernel': ['linear', 'rbf', 'poly'],  # Different kernel
options
    'select__k': [1000, 2000, 3000, 4000],  # Different values for top
k features
    'svc__C': [0.1, 0.2],  # Different values for C (controls slack in
SVM)
    'svc__gamma': ['scale', 0.001, 0.01, 0.1]  # Gamma values for RBF
and poly kernels
}

# Define the pipeline
pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(
        lowercase=True,
        tokenizer=LemmaTokenizer(stopwords=stopwords_list)
    )),
    ('select', SelectKBest(mutual_info_classif)),  # Placeholder for k
parameter
    ('scaler', StandardScaler(with_mean=False)),  # Use
with_mean=False for sparse data
    ('svc', SVC())  # SVM classifier
])

# Use GridSearchCV to find the best combination of hyperparameters
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=5,  # 10-fold cross-validation
```

```
    scoring='accuracy',
    verbose=3,  # To display progress
    n_jobs=-1  # Use all available cores
)

# Fit the model to the training data and search for best parameters
grid_search.fit(x_train, y_train)

# Get the best parameters and corresponding score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print(f"Best Cross-Validated Accuracy: {best_score:.4f}") """
```

```
' # Define the parameter grid for hyperparameter search\nparam_grid =
{\n    \'svc__kernel\': [\'linear\', \'rbf\', \'poly\'],  # Different
kernel options\n    \'select__k\': [1000, 2000, 3000, 4000],  #
Different values for top k features\n    \'svc__C\': [0.1, 0.2],  #
Different values for C (controls slack in SVM)\n    \'svc__gamma\':
[\'scale\', 0.001, 0.01, 0.1]  # Gamma values for RBF and poly
kernels\n}\n\n# Define the pipeline\npipeline = Pipeline([\n
(\'vectorizer\', TfidfVectorizer(\n        lowercase=True,\n
tokenizer=LemmaTokenizer(stopwords=stopwords_list)\n    )),\n
(\'select\', SelectKBest(mutual_info_classif)),  # Placeholder for k
parameter\n    (\'scaler\', StandardScaler(with_mean=False)),  # Use
with_mean=False for sparse data\n    (\'svc\', SVC())  # SVM
classifier\n])\n\n# Use GridSearchCV to find the best combination of
hyperparameters\ngrid_search = GridSearchCV(\n    estimator=pipeline,\
n    param_grid=param_grid,\n    cv=5,  # 10-fold cross-validation\n
scoring=\'accuracy\',\n    verbose=3,  # To display progress\n
n_jobs=-1  # Use all available cores\n)\n\n# Fit the model to the
training data and search for best parameters\ngrid_search.fit(x_train,
y_train)\n\n# Get the best parameters and corresponding score\
nbest_params = grid_search.best_params_\nbest_score =
grid_search.best_score_\n\nprint("Best Parameters:", best_params)\
nprint(f"Best Cross-Validated Accuracy: {best_score:.4f}") '
```

10-fold cross validation

```
vectorizer = TfidfVectorizer(
    lowercase=True,
    tokenizer=LemmaTokenizer(stopwords=stopwords_list)
)

x_train_tfidf = vectorizer.fit_transform(x_train)

selector = SelectKBest(mutual_info_classif, k=3000)
x_train_mi = selector.fit_transform(x_train_tfidf, y_train)
```

```
scaler = StandardScaler()
x_train_svc = scaler.fit_transform(np.asarray(x_train_mi.todense()))

classifier = SVC(kernel="rbf",gamma='scale', C=1)


accuracies = []
class_accuracies = {class_name: [] for class_name in set(y_train)}  #
To store accuracy for each class
kf = KFold(n_splits=10, shuffle=True, random_state=42)
fold = 0

# Start measuring time
start_time = time.time()

accuracies = []
training_accuracies = []
class_accuracies = {class_name: [] for class_name in set(y_train)}  #
To store accuracy for each class
kf = KFold(n_splits=10, shuffle=True, random_state=42)
fold = 0

for train_index, val_index in kf.split(x_train_svc):
    fold += 1
    X_train_fold, X_val_fold = x_train_svc[train_index],
x_train_svc[val_index]
    y_fold_train, y_fold_val = y_train[train_index],
y_train[val_index]

    # Train the classifier
    classifier.fit(X_train_fold, y_fold_train)

    # Predict and evaluate on the validation set
    y_pred = classifier.predict(X_val_fold)
    y_pred_training = classifier.predict(X_train_fold)

    # Display results for each fold
    print(f"\nFold n°{fold}:")

    # Get accuracy per class
    class_accuracy = classification_report(y_fold_val, y_pred,
output_dict=True)
    print("Classification Report:\n",
classification_report(y_fold_val, y_pred))

    accuracy = accuracy_score(y_fold_val, y_pred)
    accuracies.append(accuracy)

    accuracy_training = accuracy_score(y_pred_training, y_fold_train)
    training_accuracies.append(accuracy_training)
```

```python
    for label, metrics in class_accuracy.items():
        if label != 'accuracy' and label!="macro avg" and label!=
"weighted avg":
            class_accuracies[label].append(metrics['precision'])

# Compute total time
end_time = time.time()
total_time = end_time - start_time
print(f"\nTotal computing time for 10 folds: {total_time:.2f}
seconds")

# Mean accuracy across 10 folds
mean_accuracy = np.mean(accuracies)
print(f"Mean Accuracy across 10 folds for SVM classifier:
{mean_accuracy:.4f}")

# Average accuracy for each class
print("\nAverage Accuracy per Class:")
for label, accuracies in class_accuracies.items():
    avg_class_accuracy = np.mean(accuracies)
    print(f"Class {label}: {avg_class_accuracy:.4f}")

# Mean training accuracy across 10 folds
mean_training_accuracy = np.mean(accuracy_training)
print(f"Mean training accuracy across 10 folds for SVM classifier:
{mean_training_accuracy:.4f}")
```

```
Fold n°1:
Classification Report:
              precision    recall  f1-score   support

    Brussels       0.69      0.87      0.77        38
      London       0.71      0.84      0.77        32
    Montreal       0.90      0.59      0.72        32
     Toronto       0.85      0.74      0.79        38

    accuracy                           0.76       140
   macro avg       0.79      0.76      0.76       140
weighted avg       0.79      0.76      0.76       140
```