twitter_hate_review_src

November 23, 2021

[ ]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.utils import class_weight
from collections import Counter
import re
import pandas as pd
from nltk.tokenize import TweetTokenizer
from nltk.corpus import stopwords
from string import punctuation

import warnings
warnings.filterwarnings("ignore")
warnings.filterwarnings(action='ignore', category=DeprecationWarning)
warnings.filterwarnings(action='ignore', category=FutureWarning)
```

[ ]:
```
pre_token_replace_vals = [(re.compile(r'@\w+'), ''),
                          (re.compile(r'http\S+'), '')]
stop_words_filename = 'english'
additional_stop_words = ['amp', 'rt']
```

[ ]:
```
def read_data_set(data_filename):
    '''function to read dataset and print some information about the dataset'''
    # read csv file into dataframe
    data_df = pd.read_csv(
        data_filename, delimiter=",", encoding="utf-8")

    # print the info of twitter data frame
    print(data_df.head())
    print(data_df.shape)
    print(data_df.columns)
    print(data_df.isnull().sum())
```

```
    return data_df
```

[ ]:
```
def get_stop_words(stop_words_filename, additional_stop_words):
    '''function create english language stop words and add additional required␣
↪terms'''
    stop_words = stopwords.words(stop_words_filename) # get stop words for␣
↪given language
    stop_words.extend(additional_stop_words) # add additional stop words

    return stop_words
```

[ ]:
```
def pre_token_cleanup(text, replace_vals):
    '''function to pre-process the tweets'''
    text = text.lower()  # convert to lower case

    # text = replace_with(text, [('&amp;', 'and'), ('&gt;', '>'), ('&lt;',␣
↪'<')])
    for replace in replace_vals:
        text = re.sub(replace[0], replace[1], text)

    text = text.strip()  # remove leading and trailing whitespace
    return text
```

[ ]:
```
def remove_stop_words(text, stop_words):
    '''function to remove stop words for given text'''
    text = [word for word in text if word not in stop_words]

    return text
```

[ ]:
```
def remove_hashtags(text, tokenizer):
    '''function to remove hashtags from the tweets'''
    pattern = r'#'
    joined_text = ' '.join(text)
    cleaned_text = re.sub(pattern, '', joined_text)

    return tokenizer.tokenize(cleaned_text)
```

[ ]:
```
def remove_length_one_words(text):
    '''function to remove length one words'''
    text = [word for word in text if len(word) > 1]

    return text
```

[ ]:
```
def remove_punctuation(text):
    '''function to remove punctuation from the tweets'''
    text = [word for word in text if word not in punctuation]
```

```
    return text
```

[ ]:
```
def remove_nonalpha(text):
    '''function to remove non alpha characters'''
    text = [word for word in text if word.isalpha()]

    return text
```

[ ]:
```
def read_twitter_data_set():
    '''function to read the train dataset and print some information about the␣
 ↪dataset amd rempove unneccessary columns'''
    # read csv file into dataframe
    twitter_hate_df = read_data_set('../data/twitter_hate.csv')
    twitter_hate_df.drop(["id"], axis=1, inplace=True)  # remove id column

    return twitter_hate_df
```

[ ]:
```
def clean_tweets(twitter_hate_df):
    '''function to cleanup teh tweets to prepare for training'''
    # Normalize the casing, remove user handles and URLs.
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        pre_token_cleanup, args=(pre_token_replace_vals,))
    print(twitter_hate_df.head())

    # Using TweetTokenizer from NLTK, tokenize the tweets into individual terms.
    tokenizer = TweetTokenizer()
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        tokenizer.tokenize)
    print(twitter_hate_df.head())

    # Remove stop words and redundant terms like ‘amp’, ‘rt’, etc..
    stop_words = get_stop_words(
        stop_words_filename, additional_stop_words)
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        remove_stop_words, args=(stop_words,))
    print(twitter_hate_df.head())

    # Remove ‘#’ symbols from the tweet while retaining the term.
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        remove_hashtags, args=(tokenizer,))
    print(twitter_hate_df.head())

    # Removing tweets with a length of 1.
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        remove_length_one_words)
    print(twitter_hate_df.head())
```

```
    # Removing punctuation.
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        remove_punctuation)
    print(twitter_hate_df.head())

    # Removing non-alpha characters.
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        remove_nonalpha)
    print(twitter_hate_df.head())

    return twitter_hate_df
```

[ ]:
```
def find_top_terms_in_tweets(twitter_hate_df):
    '''function to find top 10 terms in tweets'''
    # find top 10 terms in tweets
    top_terms = Counter()
    for tweet in twitter_hate_df["tweet"]:
        top_terms.update(tweet)

    print(top_terms.most_common(10))
```

[ ]:
```
def format_for_training(twitter_hate_df):
    '''function to format the data for training'''
    # join the tokens back to string
    twitter_hate_df["tweet"] = twitter_hate_df["tweet"].apply(
        lambda x: ' '.join(x))
    print(twitter_hate_df.head())

    X = twitter_hate_df["tweet"]
    y = twitter_hate_df["label"]

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42)
    print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

    return X_train, X_test, y_train, y_test
```

[ ]:
```
def transform_to_feature_vector(X_train, X_test):
    '''function to transform the data to feature vector'''
    # Create the TF-IDF vectorizer with the top 5000 terms.
    vectorizer = TfidfVectorizer(max_features=5000)
    X_train_vectorized = vectorizer.fit_transform(X_train)
    X_test_vectorized = vectorizer.transform(X_test)

    print(X_train_vectorized.shape, X_test_vectorized.shape)

    return X_train_vectorized, X_test_vectorized, vectorizer
```

[ ]:
```
def train_model(X_train_vectorized, y_train):
    '''function to train the model'''
    # Train the model using the training data.
    model = LogisticRegression()
    model.fit(X_train_vectorized, y_train)

    return model
```

[ ]:
```
def make_predictions(model, X_train_vectorized, X_test_vectorized):
    '''function to make predictions'''
    # Make predictions on the train data.
    y_train_pred = model.predict(X_train_vectorized)
    y_test_pred = model.predict(X_test_vectorized)

    return y_train_pred, y_test_pred
```

[ ]:
```
def evaluate_model(y_train_pred, y_test_pred, y_train, y_test):
    '''function to evaluate the model'''
    # Evaluate the model using the training data.
    print("Accuracy on training data:", accuracy_score(y_train_pred, y_train))
    print("Accuracy on test data:", accuracy_score(y_test_pred, y_test))

    print(classification_report(y_train, y_train_pred))

    return None
```

[ ]:
```
def adjusting_class_imbalance_for_balance_model(X_train_vectorized, y_train):
    '''function to adjust class imbalance'''
    # Adjust class imbalance.
    balanced_model = LogisticRegression(class_weight='balanced')
    balanced_model.fit(X_train_vectorized, y_train)

    return balanced_model
```

[ ]:
```
def make_predictions_with_balanced_model(balanced_model, X_train_vectorized,␣
↪X_test_vectorized):
    '''function to make predictions with balanced model'''
    # Make predictions on the train data.
    y_train_pred = balanced_model.predict(X_train_vectorized)
    y_test_pred = balanced_model.predict(X_test_vectorized)

    return y_train_pred, y_test_pred
```

[ ]:
```
def evaluate_model_with_balanced_model(y_train_pred, y_test_pred, y_train,␣
↪y_test):
    '''function to evaluate the model with balanced model'''
    # Evaluate the model using the training data.
```

```
    print("Accuracy on training data:", accuracy_score(y_train_pred, y_train))
    print("Accuracy on test data:", accuracy_score(y_test_pred, y_test))

    print(classification_report(y_train, y_train_pred))

    return None
```

[ ]:
```
def get_parameters_for_grid_search():
    '''function to get parameters for grid search'''
    # Get the parameters for grid search.
    # parameters = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    #               'penalty': ['l1', 'l2']}

    parameters = {'C': [0.01, 0.1, 1, 10, 100],
                  'penalty': ['l1', 'l2']}
    return parameters
```

[ ]:
```
def get_grid_search_model(X_train_vectorized, y_train, parameters):
    '''function to grid search model'''
    # Grid search model.
    grid_search =␣
 ↪GridSearchCV(estimator=LogisticRegression(class_weight='balanced',␣
 ↪max_iter=1000),
                               param_grid=parameters,
                               cv=StratifiedKFold(4),
                               n_jobs=-1,
                               verbose=1,
                               scoring='recall')

    grid_search.fit(X_train_vectorized, y_train)

    print(grid_search.best_params_)
    # print(grid_search.best_score_)
    print(grid_search.best_estimator_)
    # print(grid_search.cv_results_)

    return grid_search
```

[ ]:
```
def make_predictions_with_grid_search_model(grid_search_model,␣
 ↪X_train_vectorized, X_test_vectorized):
    '''function to make predictions with grid search model'''
    # Make predictions on the train data.
    y_train_pred = grid_search_model.predict(X_train_vectorized)
    y_test_pred = grid_search_model.predict(X_test_vectorized)

    return y_train_pred, y_test_pred
```

[ ]:
```
twitter_hate_df = read_twitter_data_set()
```

[ ]:
```
twitter_hate_df = clean_tweets(twitter_hate_df)
```

[ ]:
```
find_top_terms_in_tweets(twitter_hate_df)
```

[ ]:
```
X_train, X_test, y_train, y_test = format_for_training(twitter_hate_df)
X_train_vectorized, X_test_vectorized, vectorizer =␣
↪transform_to_feature_vector(X_train, X_test)
```

[ ]:
```
model = train_model(X_train_vectorized, y_train)
```

[ ]:
```
y_train_pred, y_test_pred = make_predictions(model, X_train_vectorized,␣
↪X_test_vectorized)
```

[ ]:
```
evaluate_model(y_train_pred, y_test_pred, y_train, y_test)
```

[ ]:
```
balanced_model =␣
↪adjusting_class_imbalance_for_balance_model(X_train_vectorized, y_train)
```

[ ]:
```
y_train_pred, y_test_pred =␣
↪make_predictions_with_balanced_model(balanced_model, X_train_vectorized,␣
↪X_test_vectorized)
```

[ ]:
```
evaluate_model_with_balanced_model(y_train_pred, y_test_pred, y_train, y_test)
```

[ ]:
```
grid_search_model = get_grid_search_model(X_train_vectorized, y_train,␣
↪get_parameters_for_grid_search())
```

[ ]:
```
y_train_pred, y_test_pred =␣
↪make_predictions_with_grid_search_model(grid_search_model,␣
↪X_train_vectorized, X_test_vectorized)
```

[ ]:
```
print(classification_report(y_test, y_test_pred))
```

[ ]:
```

```