naive_bayes

November 2, 2020

[1]:
```
import warnings
warnings.simplefilter('ignore')
```

[2]:
```
import os
import random
from string import punctuation
```

[3]:
```
import numpy as np
import pandas as pd
```

[4]:
```
data_dir = 'data/20_newsgroups/'
```

[5]:
```
def get_targets(data_dir):
    # Assign target values to each of the classes in the dataset
    targets = {}
    for i, newsgroup in enumerate(os.listdir(data_dir)):
        targets[newsgroup] = i
    return targets
```

1 Assigning a target value to each document class

[6]:
```
targets_dict = get_targets(data_dir)
targets_dict
```

[6]:
```
{'alt.atheism': 0,
 'rec.autos': 1,
 'comp.windows.x': 2,
 'sci.med': 3,
 'sci.crypt': 4,
 'comp.os.ms-windows.misc': 5,
 'talk.politics.mideast': 6,
 'talk.politics.misc': 7,
 'sci.electronics': 8,
 'rec.sport.baseball': 9,
 'rec.sport.hockey': 10,
 'comp.graphics': 11,
```

```
 'sci.space': 12,
 'talk.politics.guns': 13,
 'comp.sys.mac.hardware': 14,
 'misc.forsale': 15,
 'talk.religion.misc': 16,
 'rec.motorcycles': 17,
 'comp.sys.ibm.pc.hardware': 18,
 'soc.religion.christian': 19}
```

[7]:
```
def get_data_paths(data_dir):
    X_paths, Y = [], []
    targets_dict = get_targets(data_dir)
    for newsgroup_dir in os.listdir(data_dir):
        class_path = os.path.join(data_dir, newsgroup_dir)
        for text_file in os.listdir(class_path):
            file_path = os.path.join(class_path, text_file)
            try:
                with open(file_path, 'r') as fp:
                    x = fp.readlines()
            except UnicodeDecodeError:
                print(f'DecodeError, ignoring -- {file_path}')
                os.remove(file_path)
                continue
            X_paths.append(file_path)
            Y.append(targets_dict.get(newsgroup_dir))

    return X_paths, Y
```

[8]:
```
X_paths, Y = get_data_paths(data_dir)
```

[9]:
```
print(f'Total data samples: {len(Y)}')
```

```
Total data samples: 19924
```

Randomly checking if the data is correct or not

[10]:
```
random.sample(X_paths, 5)
```

[10]:
```
['data/20_newsgroups/misc.forsale/74749',
 'data/20_newsgroups/talk.politics.guns/53324',
 'data/20_newsgroups/talk.politics.misc/178729',
 'data/20_newsgroups/comp.graphics/38571',
 'data/20_newsgroups/comp.os.ms-windows.misc/10072']
```

[11]:
```
random.sample(Y, 5)
```

[11]:
```
[1, 19, 7, 11, 3]
```

[12]:
```
def split_train_test(X, y, test_pct=0.5):
    total_len = len(y)
    train_len = int(test_pct*total_len)
    train_indices = random.sample(range(total_len), train_len)
    test_indices = [k for k in range(total_len) if k not in train_indices]
    X_train, y_train, X_test, y_test = [], [], [], []
    for i in train_indices:
        X_train.append(X[i])
        y_train.append(y[i])

    for i in test_indices:
        X_test.append(X[i])
        y_test.append(y[i])

    return X_train, y_train, X_test, y_test
```

Stop Words taken from NLTK corpora

- These words are very common and do not contribute much to the semantic meaning of a text document
- So, I am filtering out these words from the documents

[14]:
```

```

Remove headers from the document text

- There are a couple of line breaks after header information in each file
- So, check for that and remove everything above that

[16]:
```
def remove_headers(lines):
    for i, line in enumerate(lines):
```

```
        # First make sure that the bytecodes read is decoded
        line = line.decode(encoding='utf-8')
        if line == '\n':
            break
    return lines[i+1:]
```

Remove whitespaces and stop words from every line

[17]:
```
def remove_digits(word):
    for i in range(10):
        word = word.replace(str(i), '')
    return word
```

[18]:
```
def remove_punctuations(word):
    all_punctuations = punctuation.replace("'", "")
    # Also, add tabs
    all_punctuations += '\t'
    table = str.maketrans('', '', all_punctuations)
    return word.translate(table)
```

[19]:
```
def pre_process(words):
    """
    Takes in a list of words and applies some preprocessing
    1. Remove numbers from string
    2. Remove punctuations
    3. Remove quotes from words if present
    """
    processed_words = []
    for word in words:
        # Remove numbers from words
        word = remove_digits(word)

        # Remove punctuations
        word = remove_punctuations(word)

        # Do not process empty or one character strings
        if len(word) < 2:
            continue

        # Also check for quoted words and remove the quotes
        if word[0] in ["'", '"']:
            word = word[1:]
        if word[-1] in ["'", '"']:
            word = word[:-1]

        processed_words.append(word)
```

```
    return processed_words
```

[20]:
```
def validate_line(line):
    # Return a list of valid words
    words = line.replace('\n', '').strip().split(' ')
    words = pre_process(words)
    return words
```

[21]:
```
def read_file(file_path):
    try:
        with open(file_path, 'rb') as file:
            lines = file.readlines()
        valid_lines = remove_headers(lines)
        valid_words = []
        for line in valid_lines:
            # Decode byte words to string on each line
            line = line.decode(encoding='utf-8')
            processed_line = validate_line(line)
            for word in processed_line:
                word = word.lower()
                if len(word) > 1 and word not in stop_words:
                    valid_words.append(word)

    except Exception as error:
        # print(f'ERROR: {error} || FILE_NAME: {file_path}')
        return [], 1

    return valid_words, 0
```

[22]:
```
read_file('data/20_newsgroups/alt.atheism/54238')[0][:10]
```

[22]:
```
['article',
 'cvmrzdarksideosrheuoknoredu',
 'bilokcforumosrheedu',
 'conner',
 'writes',
 'myth',
 'refer',
 'convoluted',
 'counterfeit',
 'athiests']
```

1.1 Selecting features for the dataset

[23]:
```
def get_features(X, n_features=4000, reject_words=0):
    """Goes through the entire training set and gets top "n_features" words␣
↪appeared in the documents along with their frequencies"""
    all_words = []
    file_errors = 0
    for file_path in X:
        words, has_error = read_file(file_path)
        file_errors += has_error
        for w in words:
            all_words.append(w)

    words, counts = np.unique(np.array(all_words), return_counts=True)
    freq, words = (list(i) for i in zip(*sorted(zip(counts, words),␣
↪reverse=True)))
    # print(len(words), words[:10], freq[:10])
    # print(f'Total file encoding errors: {file_errors}')

    # Return the 4000 words removing the first reject_words (as they are very␣
↪common and won't be useful in differentiating among the documents)
    # in the whole dataset
    return words[reject_words:n_features]
```

[24]:
```
def doc_word_freq(X):
    """
        Returns a list of dictionaries that contain the frequencies of words in␣
↪each document
        --> [{'word1': 3, ...} ...]
    """
    word_freq = []
    for file_path in X:
        words, has_error = read_file(file_path)
        words, counts = np.unique(np.array(words), return_counts=True)
        word_counts = {}
        for i, word in enumerate(words):
            word_counts[word] = counts[i]

        word_freq.append(word_counts)
    return word_freq
```

[25]:
```
def create_data(X, feature_words):
    X_data = []
    word_freq = doc_word_freq(X)
    for doc_words in word_freq:
        # doc_words is a dict that contains words in that document along with␣
↪their number of appearences
```

```
        doc_data = []
        for f_word in feature_words:
            if f_word in doc_words.keys():
                # Add the frequency for the word to create a feature vector for␣
↪training set
                doc_data.append(doc_words[f_word])
            else:
                doc_data.append(0)
        X_data.append(doc_data)
    return np.array(X_data)
```

[26]:
```
def get_train_test(X_train, y_train, X_test, y_test, n_features=4000,␣
↪reject_words=0):
    feature_words = get_features(X_train, n_features, reject_words)
    X_train = create_data(X_train, feature_words)
    X_test = create_data(X_test, feature_words)
    y_train = np.array(y_train)
    y_test = np.array(y_test)
    print(f'Train samples: {len(X_train)} || Test samples: {len(X_test)}')
    return X_train, y_train, X_test, y_test
```

2 Text Classification with Naive Bayes

[27]:
```
class NaiveBayes:
    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.prior = None
        self._is_trained = False

    def fit(self, X_train, y_train):
        n = X_train.shape[0]

        # Separate data in X_train by its classes
        X_by_class = np.array([X_train[y_train == c] for c in np.
↪unique(y_train)])
        self.prior = np.array([len(X_class)/n for X_class in X_by_class])

        # Get word counts
        self.word_counts = np.array([row.sum(axis=0) for row in X_by_class]) +␣
↪self.alpha
        self.lk_word = self.word_counts / self.word_counts.sum(axis=1).
↪reshape(-1, 1)

        self._is_trained = True
        return self
```

```
    def predict(self, X):
        return self.predict_prob(X).argmax(axis=1)

    def score(self, X_test, y_test):
        y_pred = self.predict_prob(X_test).argmax(axis=1)
        return np.mean(y_pred == y_test)

    def predict_prob(self, X):
        if not self._is_trained:
            print('Model not trained yet!!')

        # Go through each input vector to calculate the conditional␣
↪probabilities
        class_nums = np.zeros(shape=(X.shape[0], self.prior.shape[0]))
        for i, x in enumerate(X):
            word_exists = x.astype(bool)
            lk_words_present = self.lk_word[:, word_exists] ** x[word_exists]
            lk_message = (lk_words_present).prod(axis=1)
            class_nums[i] = lk_message * self.prior

        normalize_term = class_nums.sum(axis=1).reshape(-1, 1)
        conditional_probs = class_nums / normalize_term
        return conditional_probs
```

[31]:
```
def k_fold(X_paths, Y, k=5, n_features=5000, increment=1000, reject_words=0):
    for i in range(k):
        print(f'\nfold: {i} || n_features: {n_features} || reject_words:␣
↪{reject_words}')
        X_train_list, y_train_list, X_test_list, y_test_list =␣
↪split_train_test(X_paths, Y, test_pct=0.5)
        X_train, y_train, X_test, y_test = get_train_test(X_train_list,␣
↪y_train_list, X_test_list, y_test_list, n_features, reject_words)
        alpha = random.uniform(0.5, 1.0)
        print(f'Alpha chosen: {alpha}')
        clf = NaiveBayes(alpha=alpha)
        clf.fit(X_train, y_train)
        print(f'Acc: {clf.score(X_test, y_test)}')
        #n_features += increment
```

3 Test some hyperparameters

[29]:
```
k_fold(X_paths, Y, k=10, reject_words=900)
```

```
fold: 0 || n_features: 5000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
```

```
Alpha chosen: 0.8131513439076901
Acc: 0.6844007227464365

fold: 1 || n_features: 6000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.8182395813367933
Acc: 0.6923308572575788

fold: 2 || n_features: 7000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.7050027565002497
Acc: 0.7020678578598675

fold: 3 || n_features: 8000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.8772637911906764
Acc: 0.6997590845211805

fold: 4 || n_features: 9000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.7751594113962157
Acc: 0.7087934149769123

fold: 5 || n_features: 10000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.8205057389889763
Acc: 0.7035735796024895

fold: 6 || n_features: 11000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.8617624079385859
Acc: 0.7046777755470789

fold: 7 || n_features: 12000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.7138341043845708
Acc: 0.7074884561333066

fold: 8 || n_features: 13000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.519860364482198
Acc: 0.7136117245533026

fold: 9 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.9691349812451657
Acc: 0.7197349929732986
```

4 k-Fold Cross Validation

[32]:
```
k_fold(X_paths, Y, k=10, n_features=14000, reject_words=900)
```

```
fold: 0 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.72379781297586
Acc: 0.7114033326641237

fold: 1 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.9437328497264166
Acc: 0.7073880746837984

fold: 2 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.5413693747693333
Acc: 0.7186307970287091

fold: 3 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.5612603118958358
Acc: 0.7143143946998595

fold: 4 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.5382103066097286
Acc: 0.711704477012648

fold: 5 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.669042708208162
Acc: 0.7115037141136318

fold: 6 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.9129900331480532
Acc: 0.7074884561333066

fold: 7 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.725490893816866
Acc: 0.71321019875527

fold: 8 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.6191943269659299
```

```
Acc: 0.7115037141136318

fold: 9 || n_features: 14000 || reject_words: 900
Train samples: 9962 || Test samples: 9962
Alpha chosen: 0.79587695498053
Acc: 0.7084922706283878
```

[ ]:

5 For comparison

[33]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
```

[35]:
```
X_train_list, y_train_list, X_test_list, y_test_list =␣
↪split_train_test(X_paths, Y, test_pct=0.5)
X_train, y_train, X_test, y_test = get_train_test(X_train_list, y_train_list,␣
↪X_test_list, y_test_list, n_features=14000, reject_words=900)
```

```
Train samples: 9962 || Test samples: 9962
```

[36]:
```
clf = MultinomialNB()
clf.fit(X_train, y_train)
y_predict = clf.predict(X_test)
print(clf.score(X_test, y_test))
```

```
0.782975306163421
```

[ ]: