HW1_BOW

October 17, 2024

1 Bag of Words Model

Dataset: NYT dataset contains a **text** column consisting of news articles and a **label** column indicating the category to which this article belongs.

Goal: Train a text classifier using the following document representation techniques and report accuracy, macro-f1 score, and micro-f1 score on the test set.

- Each document is represented as a **binary-valued** vector of dimension equal to the size of the vocabulary. The value at an index is 1 if the word corresponding to that index is present in the document, else 0.
- A document is represented by a vector of dimension equal to the size of the vocabulary where the value corresponding to each word is its **frequency** in the document.
- Each document is represented by a vector of dimension equal to the size of the vocabulary where the value corresponding to each word is its **tf-idf** value.

Use the **logistic regression** classifier.

[1]:
```
# import required libraries
import re
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

from tqdm import tqdm
from pprint import pprint
from scipy import sparse
from collections import defaultdict, Counter

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪PredefinedSplit
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score, f1_score
```

```
# custom visualisation styling
custom = {"axes.edgecolor": "red", "grid.linestyle": "dashed", "grid.color":␣
 ↪"black"}
sns.set_style("darkgrid", rc=custom)
```

```
<Figure size 800x600 with 0 Axes>
```

[2]:
```
# load NYT dataset
data = pd.read_csv("nyt.csv")
print(data.shape)
```

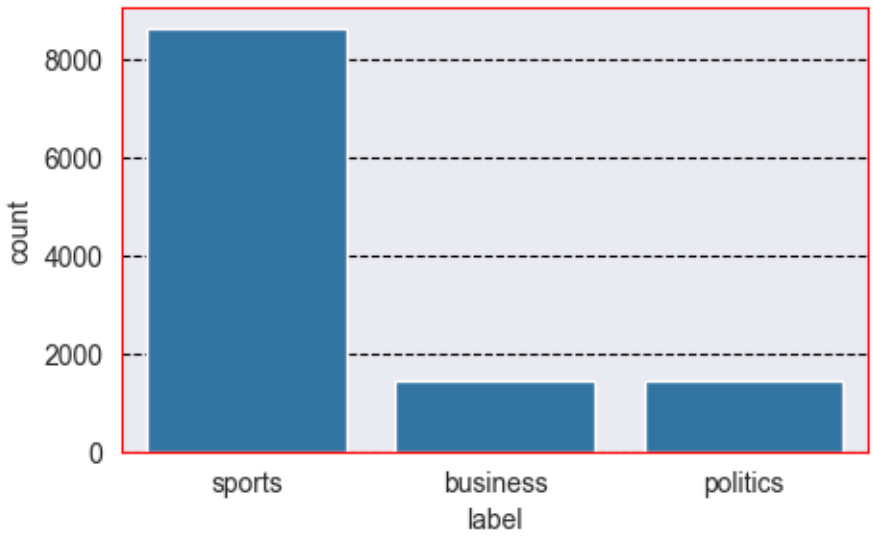
```
(11519, 2)
```

[3]:
```
# check available columns
df = data.copy()
print(df.columns)

plt.figure(figsize=(5,3))
sns.countplot(data=df, x='label')
```

```
Index(['text', 'label'], dtype='object')
```

[3]:
```
<Axes: xlabel='label', ylabel='count'>
```



[4]:
```
# check random datapoint
idx = np.random.randint(len(df))
print(df.text[idx])
```

```
manchester, england - wayne rooney scored twice against bayer leverkusen on
tuesday to reach 200 manchester united goals as he enjoys a new lease of life
```

under david moyes who thinks he can become one of the club's greatest predators.the striker's commitment to old trafford had been questioned and he was left out of key games last season under alex ferguson, who also took to playing him in more of a midfield role as robin van persie took the goalscoring glory.rooney is fourth on united's all-time leading scorers' list, behind bobby charlton, denis law (237) and jack rowley (211) and his manager can see him going further."i remember coming in and saying you've got a real chance to be one of the all-time leading goalscorers at this club and i tell you what if he keeps playing like he did tonight he will get there," moyes told a news conference .along with the rousing reception rooney has been getting from the old trafford crowd, despite a turbulent few months, moyes' faith in him seems to have lifted the england striker's mood and his game. ferguson had said in may that rooney wanted to leave the club, triggering a close season of speculation over his future and an ultimately unsuccessful but very public pursuit by chelsea and jose mourinho for his signature.as the rumours swirled, rooney himself never spoke publicly on the matter and on tuesday when asked about it he was still keen to avoid the issue of whether he wanted to stay or go."listen, i will concentrate on my football as i have done all summer," he told itv after a 4-2 victory over leverkusen in their champions league opener. "i got my head down in the summer, worked hard and i'm concentrating on my football ."the fans here have been fantastic with me and the reception i get here is great. hopefully i can reward them with goals and performances like tonight." moyes had long said rooney was training well and was in a good frame of mind and slowly but surely he has been proven right as the striker - deployed in his more traditional role - has enjoyed a good start to the season.having been united's best player in last month's 0-0 draw with chelsea in the premier league and having scored from a free kick in saturday's 2-0 win over crystal palace, rooney took his performance to a new level against leverkusen .he volleyed in a 22nd-minute opener, netted the third with a right- footed shot to bring up 200 for united and then passed superbly to assist the fourth scored by antonio valencia .his goals have come since he started wearing a headband to protect his forehead after suffering a deep gash and, whether that is a lucky charm or a mere coincidence, rooney is delighted to have reached another milestone."it's something i'm very proud of, i'm pleased to score 200 goals for a club like manchester united and hopefully there is more to come," he said."i'm delighted to be back playing and scoring and this was a good result. the first game in the champions league is always important and thankfully we got the victory."it has taken rooney 406 appearances to notch his double century, meaning he averages nearly a goal a game, and being still only 27 years old, he has time to overtake charlton."it (the record) was something we made him aware of," moyes said. "more importantly, i wanted to get him back in a good condition and mentally correct when he was ready to play,"i think you see (that, he is moving as well as he has done, he's in a good place himself at the moment, any centre forward who is scoring goals feels good about himself."rooney's latest goals took him to second in united's champions league scorers chart, past ryan giggs . he now has 30 goals, eight shy of ruud van nistelrooy's record tally. leverkusen manager sami hyypia could only watch in awe as rooney tore his side apart."he showed he can score a lot of goals," the former liverpool defender, who knows from his playing

```
days how hard it is to come up against rooney, told a news conference ."he still
has a few years left, to score 200 goals is a good achievement… it looks so
easy how he is playing, he showed today he is a quality player."
```

[5]:
```
# check num classes in label column -> multi class classification
df.label.value_counts()
```

[5]:
```
label
sports      8639
politics    1451
business    1429
Name: count, dtype: int64
```

[6]:
```
# convert the target variable data type from string to numeric
mapped_classes = df.label.astype('category')
hm_class = dict(enumerate(mapped_classes.cat.categories))
print(hm_class)

df['label'] = df.label.astype('category').cat.codes
```

```
{0: 'business', 1: 'politics', 2: 'sports'}
```

1.0.1 Preprocessing and vocab generation

[7]:
```
# required libraries from nltk for preprocessing
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize, sent_tokenize

ps = PorterStemmer()
stop = set(stopwords.words('english'))
```

[8]:
```
# common preprocessing function
def clean_text_and_tokenise(doc):
    # remove non-alpha numeric characters and strip off braces
    doc = re.sub(r'[^\w\s]', ' ', doc)
    doc = re.sub(r'[\{\}\[\]\(\)]', '', doc)

    # remove stopwords and apply stemming
    tokens = doc.lower().split(" ")
    tokens = [ps.stem(word) for word in tokens if word not in stop and 0 <␣
 ↪len(word) < 15]
    return tokens
```

[9]:
```
# helper function to get vocabulary from the corpus of docs
def get_frequency_count_from_docs(doc):
    # apply preprocessing and tokenisation
    tokens = clean_text_and_tokenise(doc)
```

```
    # get vocab with frequencies
    vocab_hm = Counter(tokens)

    return dict(vocab_hm)
```

[10]:
```
# helper function to get doc frequency
def get_word_frequency_across_doc(df, col, vocab_hm):
    DF = defaultdict(float)
    for doc in tqdm(df[col]):
        # preprocess and get tokens
        tokens = clean_text_and_tokenise(doc)
        for token in set(tokens):
            # check if the word exists in vocab
            if token in vocab_hm:
                DF[token] += 1
    return DF
```

[11]:
```
# get entire raw corpus and lower
corpus = ' '.join(list(df["text"])).lower()

# get vocab from corpus
vocab_hm = get_frequency_count_from_docs(corpus)

# get document frequency map
df_hm = get_word_frequency_across_doc(df, "text", vocab_hm)
```

```
100%|                    | 11519/11519 [00:36<00:00, 319.86it/s]
```

[12]:
```
# check total words in vocabulary and total docs
n_vocab = len(vocab_hm)
n_docs = len(df)
print(n_docs, n_vocab)
```

```
11519 45855
```

[13]:
```
# check 10 most and least frequent words for sanity
sorted_vocab = sorted(list(vocab_hm.items()), key = lambda x : -x[1])

print("Most Frequent words in Vocabulary: \n")
pprint(dict(sorted_vocab[:10]))
print("Least Frequent words in Vocabulary: \n")
pprint(dict(sorted_vocab[-10:]))
```

```
Most Frequent words in Vocabulary:

{'first': 24314,
 'game': 34839,
 'one': 22029,
```

```
 'play': 23563,
 'said': 59594,
 'season': 20036,
 'team': 18769,
 'time': 18867,
 'two': 22526,
 'year': 29132}
Least Frequent words in Vocabulary:

{'budson': 1,
 'economix': 1,
 'emerton': 1,
 'fatalist': 1,
 'kerrilyn': 1,
 'nrl': 1,
 'paducah': 1,
 'regionq': 1,
 'shesaidy': 1,
 'taronga': 1}
```

1.0.2 Bag of Word Representations

[14]:
```
# index vocab hashmap
indexed_vocab_hm = {key: index for index, key in enumerate(vocab_hm.keys())}
```

[15]:
```
# we can create a custom function to create binarised embeddings of a doc using␣
 ↪vocabulary
def get_doc_representation(doc, name="binary"):
    # assertion check
    assert name in ["binary", "count", "tfidf"]

    # initialise the vector represntation
    vector = np.zeros(n_vocab, dtype=np.int64)

    # get raw doc tokens
    doc_tokens = clean_text_and_tokenise(doc)

    # count of words in doc
    if name != "binary":
        doc_hm = dict(Counter(doc_tokens))

    for word in set(doc_tokens):
        if word in indexed_vocab_hm:
            idx = indexed_vocab_hm[word]
            if name == "binary":
                vector[idx] = 1
            elif name == "count":
```

```
                vector[idx] = doc_hm[word]
            else:
                # maximum frequency normalisation
                tf = 0.5 + 0.5 * doc_hm[word] / max(doc_hm.values())

                # long idf
                idf = 1 + np.log(n_docs / df_hm[word])

                # tfidf
                vector[idx] = tf * idf

    return list(vector)
```

1.0.3 Modeling

Train-Test-Validation splits

[16]:
```
# split training data into train(10%) and validation(10%)
train_ratio = 0.8
test_ratio = 0.1
validation_ratio = 0.1

def get_data_splits(df, name, use_sparse=True):
    # target
    Y = df["label"]
    df = df.drop(['label'], axis=1)

    if name in ["binary", "count"]:
        # apply vectorisation to the documents
        df["bow_vector"] = df['text'].apply(lambda x: get_doc_representation(x,␣
 ↪name))
        X = list(df["bow_vector"])
    else:
        # case of tfidf - we want train/test/val to be seperate first
        X = df

    # train test splits
    train_x, val_x, train_y, val_y = train_test_split(X, Y,␣
 ↪test_size=1-train_ratio, random_state=42)
    val_x, test_x, val_y, test_y = train_test_split(
        val_x,
        val_y,
        test_size=validation_ratio/(test_ratio + validation_ratio),
        random_state=42
    )

    # we apply tfidf seperately for each of the sets
    if name == "tfidf":
```

```
        train_x = list(train_x['text'].apply(lambda x:␣
↪get_doc_representation(x, name)))
        val_x = list(val_x['text'].apply(lambda x: get_doc_representation(x,␣
↪name)))
        test_x = list(test_x['text'].apply(lambda x: get_doc_representation(x,␣
↪name)))

    # convert to sparse matrix to speed up training
    if use_sparse:
        train_x = sparse.csr_matrix(train_x)
        test_x = sparse.csr_matrix(test_x)
        val_x = sparse.csr_matrix(val_x)

    return train_x, val_x, test_x, train_y, val_y, test_y
```

Training

[25]:
```
# logistic regression model using "one-vs-rest" strategy and Grid Search cross␣
↪validation
# we have a predefined validation dataset
def get_best_LR(X, Y, split_index, param_grid):
    pds = PredefinedSplit(split_index)

    # hyperparameter tuning using grid search
    grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=pds,␣
↪scoring='accuracy')
    grid_search.fit(X, Y)

    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
    best_score = grid_search.best_score_

    print("Best Parameters:", best_params, "Best Accuracy on Validation set",␣
↪best_score)
    return best_model
```

[18]:
```
def print_metrics(y, y_pred):
    print("Accuracy:", accuracy_score(y, y_pred))
    print("Macro F1 Score:", f1_score(y, y_pred, average='macro'))
    print("Micro F1 Score:", f1_score(y, y_pred, average='micro'))
```

1.1 1a. Binarised Document Representation

[40]:
```
# get train/test/validation sets
train_x, val_x, test_x, train_y, val_y, test_y = get_data_splits(df, "binary",␣
↪use_sparse=False)
```

```
print(
    "train_data_size: {}%, validation_data_size: {}%, test_data_size: {}%".
 ↪format(
    round(100 * len(train_x)/len(df), 4),
    round(100 * len(val_x)/len(df), 4),
    round(100 * len(test_x)/len(df), 4))
)
```

```
train_data_size: 79.9983%, validation_data_size: 10.0009%, test_data_size:
10.0009%
```

[41]:
```
# concat train and val for predefined validation dataset
split_index = [-1]*len(train_x) + [0]*len(val_x)
X = np.concatenate((train_x, val_x), axis=0)
Y = np.concatenate((train_y, val_y), axis=0)

# get sparse representations
sparse_X = sparse.csr_matrix(X)
sparse_test_x = sparse.csr_matrix(test_x)
```

[42]:
```
# training and finding the best model
# hyperparameter space
param_grid = {
    'C': [0.001, 0.1, 1, 10, 100], # Regularization strength
    'max_iter': [100, 200],  # Maximum iterations
    'class_weight': [None, 'balanced']  # To handle class imbalance
}

model_binary = get_best_LR(sparse_X, Y, split_index, param_grid)
```

```
Best Parameters: {'C': 0.1, 'class_weight': 'balanced', 'max_iter': 100} Best
Accuracy on Validation set 0.9817708333333334
```

Inference on test set

[43]:
```
# infer using the best model on the test set
y_test_pred = model_binary.predict(sparse_test_x)

# calculate metrics on the test set
print_metrics(test_y, y_test_pred)
```

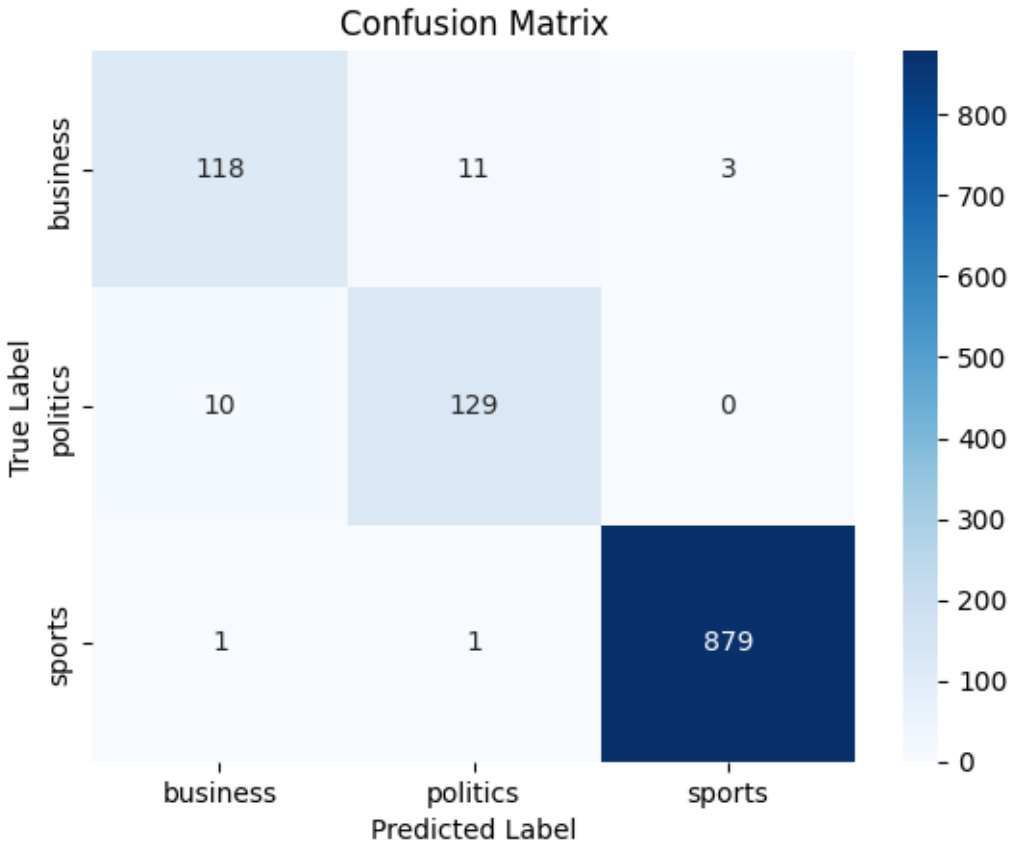
```
Accuracy: 0.9774305555555556
Macro F1 Score: 0.9409356853143893
Micro F1 Score: 0.9774305555555556
```

[44]:
```
# visualisation
# confusion matrix
def plot_confusion_matrix(test_y, y_test_pred):
    cm = confusion_matrix(test_y, y_test_pred)
```

```
    # create a heatmap using seaborn
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['business', 'politics', 'sports'],
                yticklabels=['business', 'politics', 'sports'])

    # Set labels and title
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.title('Confusion Matrix')
    plt.show()
```

[45]:
```
plot_confusion_matrix(test_y, y_test_pred)
```



1.2 1b. Count/Frequency based Document Representation

[46]:
```
# get sparse representations
train_x, val_x, test_x, train_y, val_y, test_y = get_data_splits(df, "count",␣
 ↪use_sparse=False)
```

```
# concat train and val for predefined validation dataset
split_index = [-1]*len(train_x) + [0]*len(val_x)
X = np.concatenate((train_x, val_x), axis=0)
Y = np.concatenate((train_y, val_y), axis=0)

# get sparse representations
sparse_X = sparse.csr_matrix(X)
sparse_test_x = sparse.csr_matrix(test_x)
```

[47]:
```
# training and finding the best model
model_count = get_best_LR(sparse_X, Y, split_index, param_grid)
```

```
Best Parameters: {'C': 1, 'class_weight': 'balanced', 'max_iter': 100} Best
Accuracy on Validation set 0.9887152777777778
```

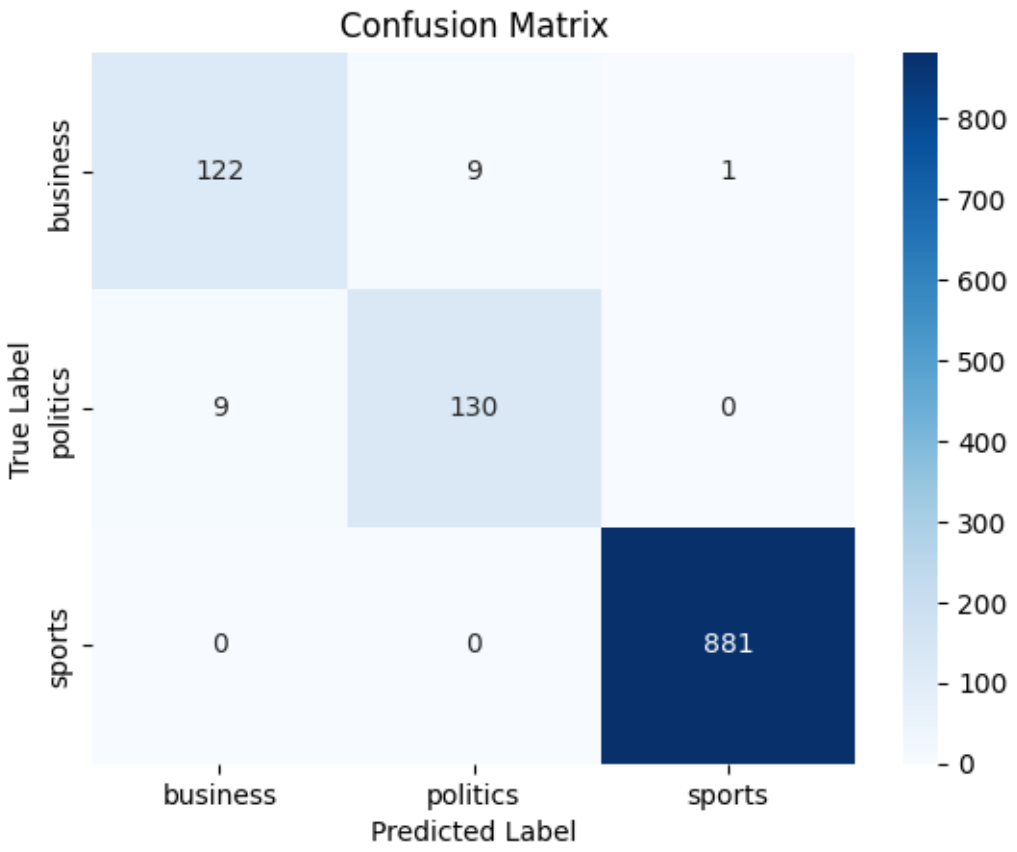
Inference on test set

[48]:
```
# infer using the best model on the test set
y_test_pred = model_count.predict(sparse_test_x)

# calculate metrics on the test set
print_metrics(test_y, y_test_pred)
```

```
Accuracy: 0.9835069444444444
Macro F1 Score: 0.9541470791930237
Micro F1 Score: 0.9835069444444444
```

[49]:
```
plot_confusion_matrix(test_y, y_test_pred)
```



1.3 1c. tf-idf Document Representation

[50]:
```
# get train/test/validation sets
train_x, val_x, test_x, train_y, val_y, test_y = get_data_splits(df, "tfidf",␣
 ↪use_sparse=False)

# concat train and val for predefined validation dataset
split_index = [-1]*len(train_x) + [0]*len(val_x)
X = np.concatenate((train_x, val_x), axis=0)
Y = np.concatenate((train_y, val_y), axis=0)

# get sparse representations
sparse_X = sparse.csr_matrix(X)
sparse_test_x = sparse.csr_matrix(test_x)
```

[51]:
```
# training and finding the best model
model_tfidf = get_best_LR(sparse_X, Y, split_index, param_grid)
```

```
Best Parameters: {'C': 10, 'class_weight': 'balanced', 'max_iter': 100} Best
Accuracy on Validation set 0.984375
```

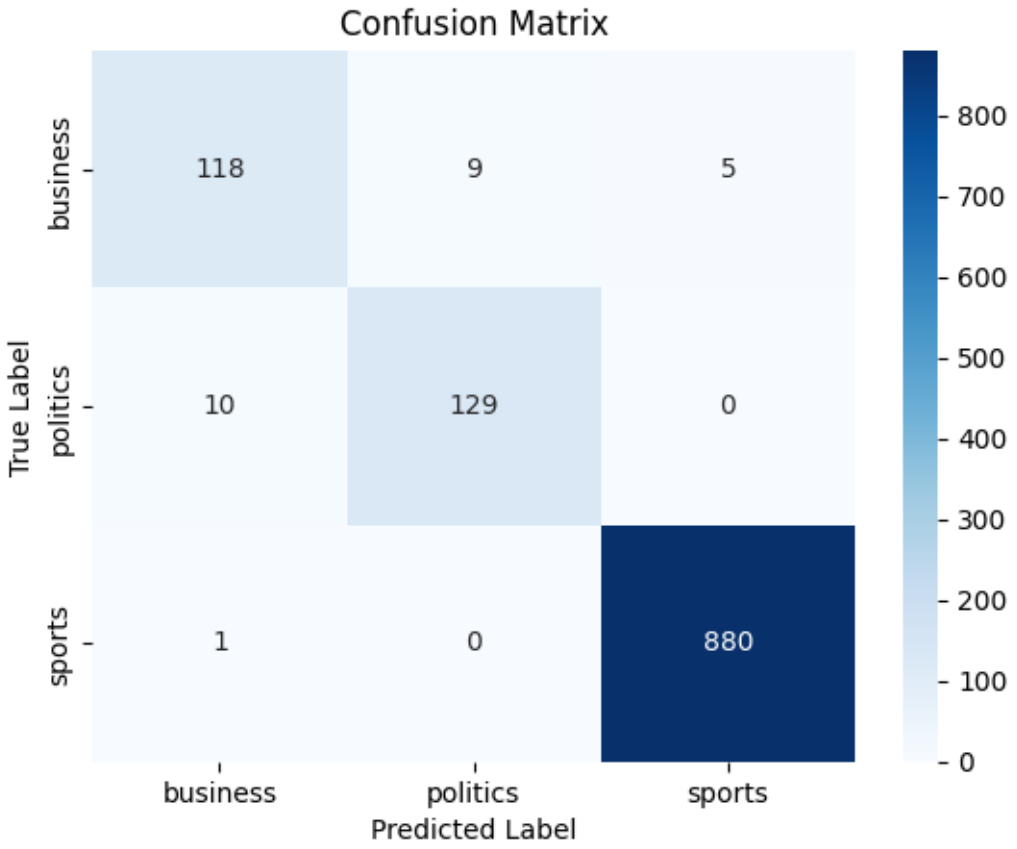
Inference on test set

[52]:
```
# infer using the best model on the test set
y_test_pred = model_tfidf.predict(sparse_test_x)

# calculate metrics on the test set
print_metrics(test_y, y_test_pred)
```

```
Accuracy: 0.9782986111111112
Macro F1 Score: 0.9440749977104897
Micro F1 Score: 0.9782986111111112
```

[53]:
```
plot_confusion_matrix(test_y, y_test_pred)
```



HW1_Word2Vec

October 17, 2024

1 Word2Vec model

(a) Train a text classifier using the following document representation techniques using 100-dimensional word vectors and report accuracy, macro-f1 score, and micro-f1 score on the test set. Compare and analyze their performance.

1. **Using publicly available pre-trained GloVe embeddings** as word vectors, a document vector is represented as an average of word vectors of its constituent words.
2. **Train Word2Vec** (e.g., use the `gensim` package) on AGNews text data and use them as word vectors to compute document vectors by averaging word vectors of its constituent words.
3. **Train Word2Vec on NYT text data** and use them as word vectors to compute document vectors by averaging word vectors of its constituent words.

(b) What are the disadvantages of averaging word vectors for the document representation? Describe an idea to overcome this.

The document vectors should be formed using word vectors.
Note: This is an open-ended question. Feel free to propose new ideas.

[3]:
```
# import necessary library
import re
import gensim
import warnings

import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from tqdm import tqdm
from pprint import pprint
from scipy import sparse
from collections import defaultdict, Counter

from gensim.models import KeyedVectors
from gensim.test.utils import get_tmpfile
from gensim.scripts.glove2word2vec import glove2word2vec
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪PredefinedSplit
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, f1_score

# custom visualisation styling
custom = {"axes.edgecolor": "red", "grid.linestyle": "dashed", "grid.color":␣
 ↪"black"}
sns.set_style("darkgrid", rc=custom)

warnings.simplefilter("ignore")
```

[4]:
```
# load NYT dataset
data = pd.read_csv("nyt.csv")
print(data.shape)
```

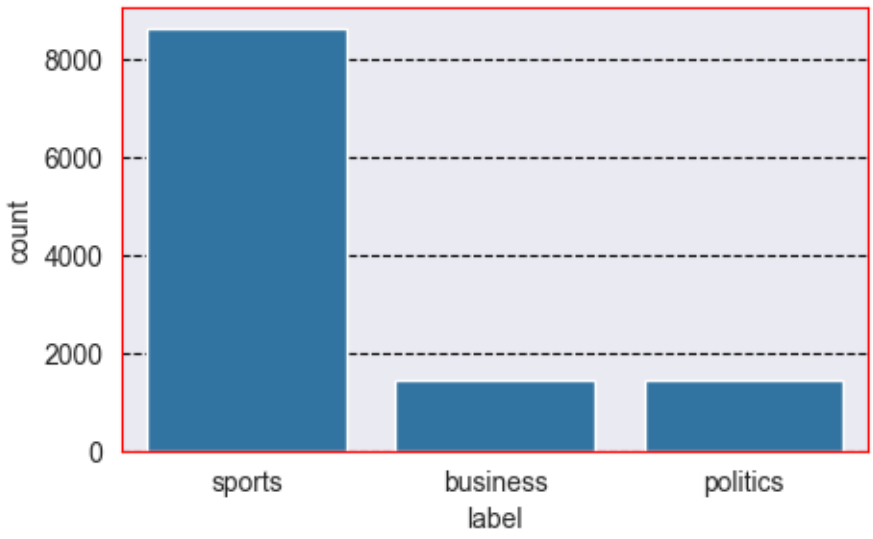
```
(11519, 2)
```

[5]:
```
# check available columns
df = data.copy()
print(df.columns)

plt.figure(figsize=(5,3))
sns.countplot(data=df, x='label')
```

```
Index(['text', 'label'], dtype='object')
```

[5]:
```
<Axes: xlabel='label', ylabel='count'>
```



[4]:
```
# convert the target variable data type from string to numeric
mapped_classes = df.label.astype('category')
hm_class = dict(enumerate(mapped_classes.cat.categories))
print(hm_class)

df['label'] = df.label.astype('category').cat.codes
```

```
{0: 'business', 1: 'politics', 2: 'sports'}
```

1.1 a1. Using GloVe Embeddings

[5]:
```
# gloVe file path containing pre-trained word vectors
glove_file = '../pretrained-models/glove.6B/glove.6B.100d.txt'

# temporary file path to store the converted Word2Vec format vectors
tmp_file = get_tmpfile("test_word2vec.txt")

# convert the gloVe file to word2vec format and save it to the temporary file
_ = glove2word2vec(glove_file, tmp_file)
model = KeyedVectors.load_word2vec_format(tmp_file)
```

[6]:
```
# check embeddings of a random word
random_vec = model['random']
print(len(random_vec), '\n', random_vec)
```

```
100
 [-0.34378   -0.13502   -0.43921    0.3171     0.45931    1.4118
  0.53641    1.072     -0.19217   -0.19073   -0.26035   -0.16939
  0.11266   -0.43995    0.15545    1.3412    -0.34172    0.94433
  0.062561   0.63704    0.5084    -0.4696     0.10751   -0.21524
  0.50907   -0.17371    0.94811    0.4571     0.40394   -0.12882
  0.50923   -0.058139  -0.55692   -0.51644    0.56536   -0.28991
 -0.081733  -0.1865     0.67905   -0.29877   -0.17778    0.42206
 -0.4408     0.2316    -0.95221    0.22149   -0.62444   -0.14468
 -0.37559   -0.4516     1.1225    -0.44304   -0.17111    0.058563
  0.44505   -1.2974     0.54388    0.49319    1.1714    -0.20397
  0.18537    0.11079    0.011758   0.33083    1.4132    -0.15832
  0.52176    0.050126   0.8741     0.16155   -0.99235   -0.034789
  0.43111    0.30439   -0.0060538 -0.10579    0.13443   -0.47229
 -1.179     -0.025391   0.97781    0.18939   -0.73967   -0.2017
 -0.84043   -0.017837   0.64232   -0.65417   -0.64107    0.27953
 -0.82348    1.0642     0.48362    0.066701  -0.26653    0.099707
  0.2748    -0.20851   -0.26602    0.058957 ]
```

1.1.1 Preprocessing

[7]:
```
# required libraries from nltk for preprocessing
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize, sent_tokenize

ps = PorterStemmer()
stop = set(stopwords.words('english'))
```

[8]:
```
# common preprocessing function
def clean_text_and_tokenise(doc):
    # remove non-alpha numeric characters and strip off braces
    doc = re.sub(r'[^\w\s]', ' ', doc)
    doc = re.sub(r'[\{\}\[\]\(\)]', '', doc)

    # remove stopwords and apply stemming
    tokens = doc.lower().split(" ")
    tokens = [ps.stem(word) for word in tokens if word not in stop and 0 <␣
 ↪len(word) < 16]
    return tokens
```

[9]:
```
# helper functions - used from Prof. Jingbo's notebook
def get_avg_word_vector(doc, model):
    vecs = []
    doc_tokens = clean_text_and_tokenise(doc)

    for token in doc_tokens:
        try:
            vecs.append(model[token])
        except KeyError:
            pass
    return list(np.mean(vecs, axis=0))
```

[10]:
```
# get average vector embeddings for each doc in dataset
df["glove_vector"] = df['text'].apply(lambda x: get_avg_word_vector(x, model))
```

1.1.2 Modeling

Train-Test-Validation splits

[11]:
```
# split training data into train(10%) and validation(10%)
train_ratio = 0.8
test_ratio = 0.1
validation_ratio = 0.1

def get_data_splits(df, col):
    # target
    Y = df["label"]
```

```
    df = df.drop(['label'], axis=1)

    # train test splits
    train_x, val_x, train_y, val_y = train_test_split(list(df[col]), Y,␣
 ↪test_size=1-train_ratio, random_state=42)
    val_x, test_x, val_y, test_y = train_test_split(
        val_x,
        val_y,
        test_size=validation_ratio/(test_ratio + validation_ratio),
        random_state=42
    )

    return train_x, val_x, test_x, train_y, val_y, test_y
```

[12]:
```
# get splits across dataset
train_x, val_x, test_x, train_y, val_y, test_y = get_data_splits(df,␣
 ↪col="glove_vector")

print(
    "train_data_size: {}%, validation_data_size: {}%, test_data_size: {}%".
 ↪format(
    round(100 * len(train_x)/len(df), 4),
    round(100 * len(val_x)/len(df), 4),
    round(100 * len(test_x)/len(df), 4))
)

# concat train and val for predefined validation dataset
split_index = [-1]*len(train_x) + [0]*len(val_x)
X = np.concatenate((train_x, val_x), axis=0)
Y = np.concatenate((train_y, val_y), axis=0)
```

```
train_data_size: 79.9983%, validation_data_size: 10.0009%, test_data_size:
10.0009%
```

1.1.3 Training

[13]:
```
# logistic regression model using "one-vs-rest" strategy and Grid Search cross␣
 ↪validation
# we have a predefined validation dataset
def get_best_LR(X, Y, split_index, param_grid):
    pds = PredefinedSplit(split_index)

    # hyperparameter tuning using grid search
    grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=pds,␣
 ↪scoring='accuracy')
    grid_search.fit(X, Y)
```

```
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_
    best_score = grid_search.best_score_

    print("Best Parameters:", best_params, "Best Accuracy on Validation set",␣
 ↪best_score)
    return best_model
```

[14]:
```
# hyperparameter space
param_grid = {
    'C': [0.001, 0.1, 1, 10, 100], # Regularization strength
    'max_iter': [100, 200, 500],  # Maximum iterations
    'penalty': ['l1', 'l2'], # Regularization
    'class_weight': [None, 'balanced']  # To handle class imbalance
}

model_glove = get_best_LR(X, Y, split_index, param_grid)
```

```
Best Parameters: {'C': 1, 'class_weight': None, 'max_iter': 100, 'penalty':
'l2'} Best Accuracy on Validation set 0.9817708333333334
```

Inference on test data

[15]:
```
def print_metrics(y, y_pred):
    print("Accuracy:", accuracy_score(y, y_pred))
    print("Macro F1 Score:", f1_score(y, y_pred, average='macro'))
    print("Micro F1 Score:", f1_score(y, y_pred, average='micro'))
```

[16]:
```
# infer using the best model on the test set
y_test_pred = model_glove.predict(test_x)

# calculate metrics on the test set
print_metrics(test_y, y_test_pred)
```

```
Accuracy: 0.9730902777777778
Macro F1 Score: 0.928047554460982
Micro F1 Score: 0.9730902777777778
```

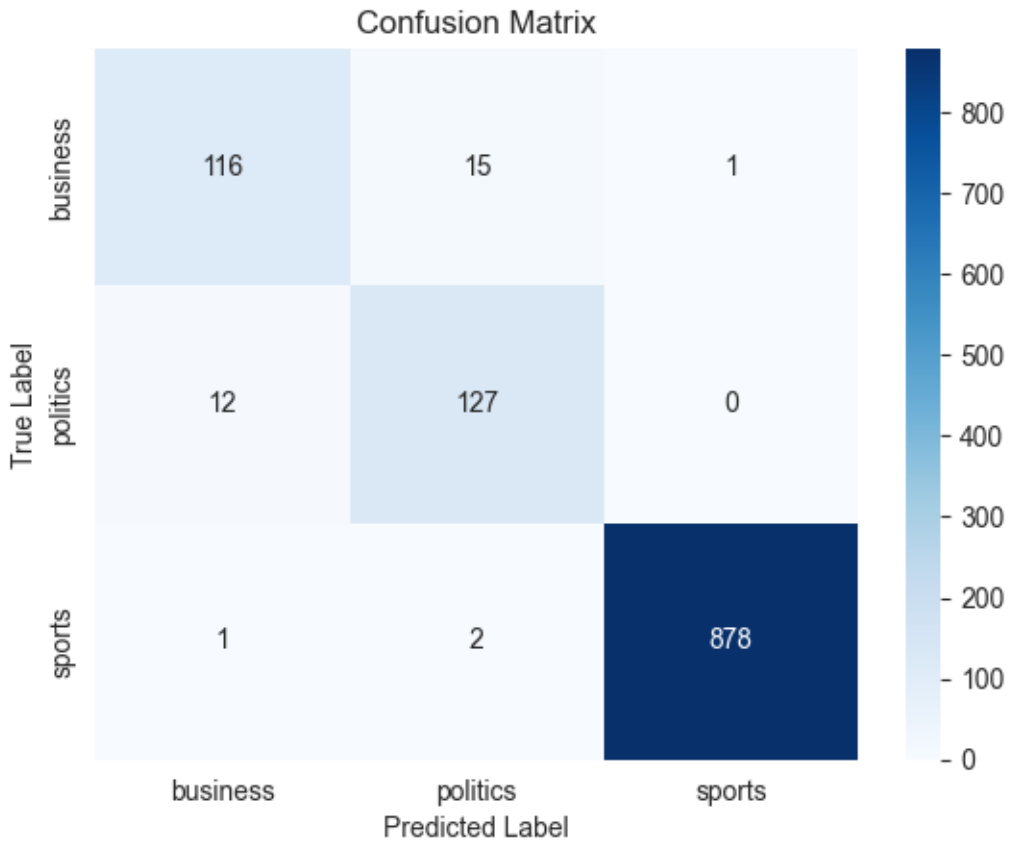
[17]:
```
# visualisation
# confusion matrix
def plot_confusion_matrix(test_y, y_test_pred):
    cm = confusion_matrix(test_y, y_test_pred)

    # create a heatmap using seaborn
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['business', 'politics', 'sports'],
                yticklabels=['business', 'politics', 'sports'])

    # Set labels and title
```

```
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.title('Confusion Matrix')
    plt.show()
```

[18]:
```
plot_confusion_matrix(test_y, y_test_pred)
```



1.2 a2. Training a Word2Vec on AGNews

[19]:
```
# AGNews Corpus
df_ag = pd.read_csv("ag.csv")
print(df_ag.shape)
```

```
(90000, 1)
```

[20]:
```
# sample text
df_ag['text'][0]
```

[20]:
```
"wall st. bears claw back into the black (reuters) . reuters - short-sellers,
wall street's dwindling band of ultra-cynics, are seeing green again."
```

[21]:
```
# curate input data for training the Word2Vec model, list of sentences
ag_news = list(df_ag['text'].apply(lambda x: clean_text_and_tokenise(x)))
```

1.2.1 initialising model and params

[22]:
```
from gensim.models import Word2Vec

# this will be a shallow deep learning model
wv_model = Word2Vec(
    sentences=ag_news,
    min_count=2, # ignores all words with a total frequency lower than this␣
 ↪value.
    window=10, # model will consider the n words before and n words after that␣
 ↪word as part of the context.
    negative=5, # k = the number of negative samples to use
    sg=1, # use Skip-Gram model.
    vector_size=100,
    workers=4
)
```

1.2.2 building vocab

[23]:
```
wv_model.build_vocab(ag_news, progress_per=10000)
print(wv_model.corpus_count)
```

```
90000
```

1.2.3 word2vec model training

[24]:
```
wv_model.train(
    ag_news,
    total_examples=wv_model.corpus_count,
    epochs=20,
    start_alpha=0.04,
    end_alpha=0.0001
)
```

[24]:
```
(47616660, 49169920)
```

[25]:
```
# normalize the word vectors and free up memory
wv_model.init_sims(replace=True)
```

1.2.4 embeddings generation

[26]:
```
# get average vector embeddings for each doc in dataset
df = data.copy()
```

```
df["wv_vector"] = df['text'].apply(lambda x: get_avg_word_vector(x, wv_model.
 ↪wv))
```

[27]:
```
# get splits across dataset
train_x, val_x, test_x, train_y, val_y, test_y = get_data_splits(df,␣
 ↪col="wv_vector")

print(
    "train_data_size: {}%, validation_data_size: {}%, test_data_size: {}%".
 ↪format(
    round(100 * len(train_x)/len(df), 4),
    round(100 * len(val_x)/len(df), 4),
    round(100 * len(test_x)/len(df), 4))
)

# concat train and val for predefined validation dataset
split_index = [-1]*len(train_x) + [0]*len(val_x)
X = np.concatenate((train_x, val_x), axis=0)
Y = np.concatenate((train_y, val_y), axis=0)
```

```
train_data_size: 79.9983%, validation_data_size: 10.0009%, test_data_size:
10.0009%
```

1.2.5 train LR and inference

[28]:
```
# train
model_lr_word2vec = get_best_LR(X, Y, split_index, param_grid)
```

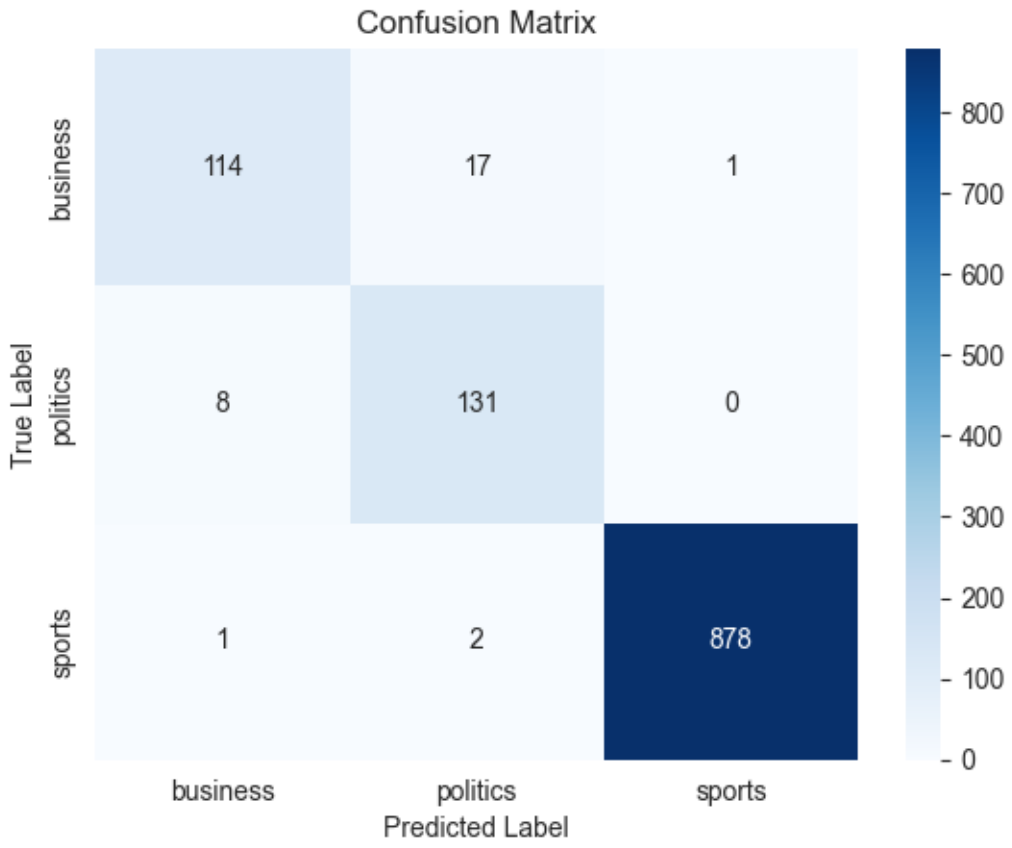
```
Best Parameters: {'C': 100, 'class_weight': None, 'max_iter': 100, 'penalty':
'l2'} Best Accuracy on Validation set 0.9765625
```

[29]:
```
# infer using the best model on the test set
y_test_pred = model_lr_word2vec.predict(test_x)

# calculate metrics on the test set
print_metrics(test_y, y_test_pred)
```

```
Accuracy: 0.9748263888888888
Macro F1 Score: 0.9328064380832547
Micro F1 Score: 0.9748263888888888
```

[30]:
```
plot_confusion_matrix(test_y, y_test_pred)
```



1.3 a3. Training a Word2Vec on NYU News dataset

[31]:
```
# nyt data
nyt = data.copy()

# curate input data for training the Word2Vec model, list of sentences
nyt_news = list(df_ag['text'].apply(lambda x: clean_text_and_tokenise(x)))
```

1.3.1 initialising model and params

[32]:
```
# this will be a shallow deep learning model
wv_model_nyt = Word2Vec(
    sentences=nyt_news,
    min_count=2, # ignores all words with a total frequency lower than this␣
 ↪value.
    window=7, # model will consider the n words before and n words after that␣
 ↪word as part of the context.
    negative=5, # k = the number of negative samples to use
    sg=0, # use CBOW model.
```

```
    vector_size=100,
    workers=4
)
```

1.3.2 building vocab

[33]:
```
wv_model_nyt.build_vocab(nyt_news, progress_per=10000)
print(wv_model_nyt.corpus_count)
```

```
90000
```

1.3.3 word2vec model training

[34]:
```
wv_model_nyt.train(
    nyt_news,
    total_examples=wv_model_nyt.corpus_count,
    epochs=10,
    start_alpha=0.1,
    end_alpha=0.0001
)
```

[34]:
```
(23807317, 24584960)
```

[35]:
```
# normalize the word vectors and free up memory
wv_model_nyt.init_sims(replace=True)
```

1.3.4 embeddings generation

[36]:
```
# get average vector embeddings for each doc in dataset
nyt["wv_vector"] = nyt['text'].apply(lambda x: get_avg_word_vector(x,␣
 ↪wv_model_nyt.wv))
```

[37]:
```
# get splits across dataset
train_x, val_x, test_x, train_y, val_y, test_y = get_data_splits(nyt,␣
 ↪col="wv_vector")

print(
    "train_data_size: {}%, validation_data_size: {}%, test_data_size: {}%".
 ↪format(
    round(100 * len(train_x)/len(df), 4),
    round(100 * len(val_x)/len(df), 4),
    round(100 * len(test_x)/len(df), 4))
)

# concat train and val for predefined validation dataset
split_index = [-1]*len(train_x) + [0]*len(val_x)
X = np.concatenate((train_x, val_x), axis=0)
```

```
Y = np.concatenate((train_y, val_y), axis=0)
```

```
train_data_size: 79.9983%, validation_data_size: 10.0009%, test_data_size:
10.0009%
```

1.3.5 train LR and inference

[38]:
```
# train
model_lr_word2vec = get_best_LR(X, Y, split_index, param_grid)
```

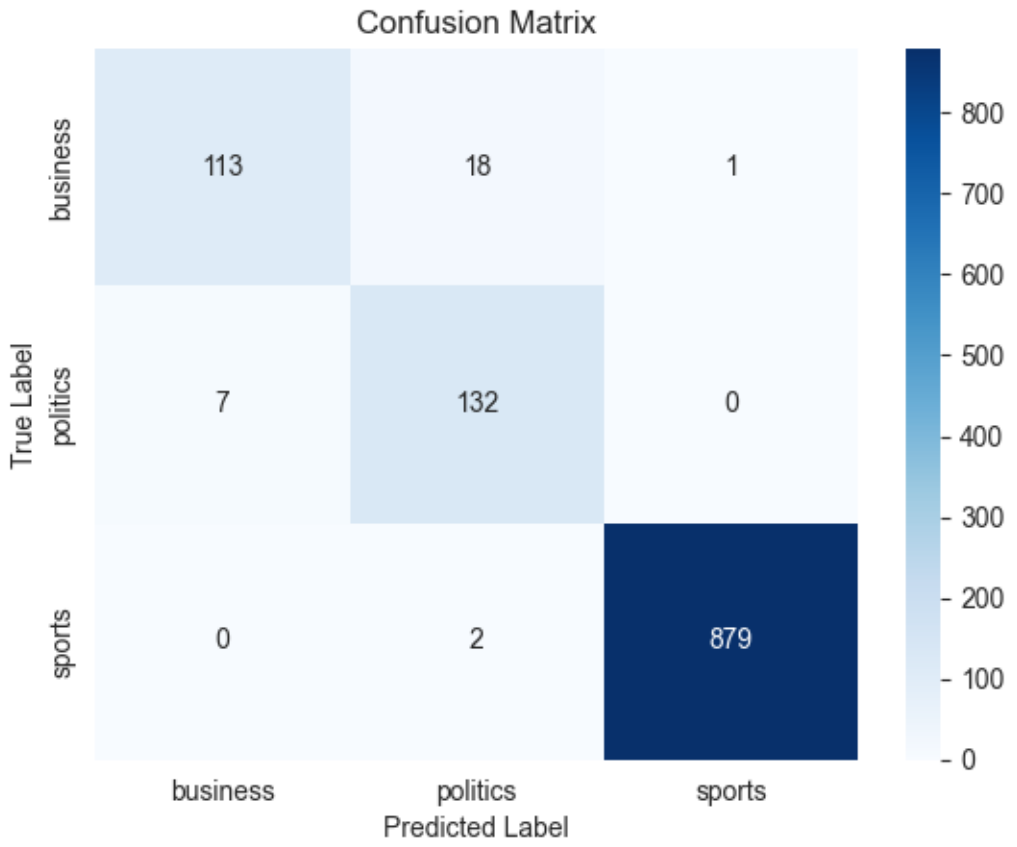
```
Best Parameters: {'C': 100, 'class_weight': None, 'max_iter': 100, 'penalty':
'l2'} Best Accuracy on Validation set 0.9791666666666666
```

[39]:
```
# infer using the best model on the test set
y_test_pred = model_lr_word2vec.predict(test_x)

# calculate metrics on the test set
print_metrics(test_y, y_test_pred)
```

```
Accuracy: 0.9756944444444444
Macro F1 Score: 0.9341127713859936
Micro F1 Score: 0.9756944444444444
```

[40]:
```
plot_confusion_matrix(test_y, y_test_pred)
```



1.4 Question

(b) What are the disadvantages of averaging word vectors for the document representation? Describe an idea to overcome this.

Disadvantages of average word vector representation

The major disadvantage of averaging word vectors arises when polar words (eg "good", "bad") appear in the same document. Word2Vec models (**word2vec-google-news-300.model**) achieves the following cosine similarity values for different polar words: * sim(**good, bad**) = 0.72 * sim(**bullish, bearish**) = 0.88 * sim(**long, short**) = 0.57

In tasks like sentiment analysis, this can lead to a loss of important information, as opposing sentiments (good, bad) may cancel each other out. Another example is automating stock trading using NLP, where financial losses could occur because words like "bullish" and "bearish" may be treated as similar, despite having opposite meanings.

Mitigation Strategies 1. We can use a more context aware representaion of words, such as advanced models like ELMo, Bert, which capture the meaning of words based on their context. 2. We can fine-tune or train word embeddings specifically for sentiment analysis or financial sentiment tasks, ensuring that words like "bullish" and "bearish" are assigned distinct vectors, even if they appear similar in general-purpose embeddings.

HW1_BERT_TensorFlow

October 17, 2024

0.1 BERT

Fine-tune the BERT (bert-base-uncased) for text classification and report accuracy, macro f1-score, and micro f1-score. If you are using PyTorch, hugging face transformers is highly recommended for this task. While tokenizing, set the maximum length to 64 and fine-tune for 3 epochs.

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from datasets import load_dataset, DatasetDict, Dataset
from transformers import AutoTokenizer,TFAutoModelForSequenceClassification
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, f1_score

# custom visualisation styling
custom = {"axes.edgecolor": "red", "grid.linestyle": "dashed", "grid.color":␣
 ↪"black"}
sns.set_style("darkgrid", rc=custom)
```

[2]:
```
# load dataset
data = pd.read_csv("nyt.csv")
print(data.shape)
```

```
(11519, 2)
```

[3]:
```
data.head()
```

[3]:

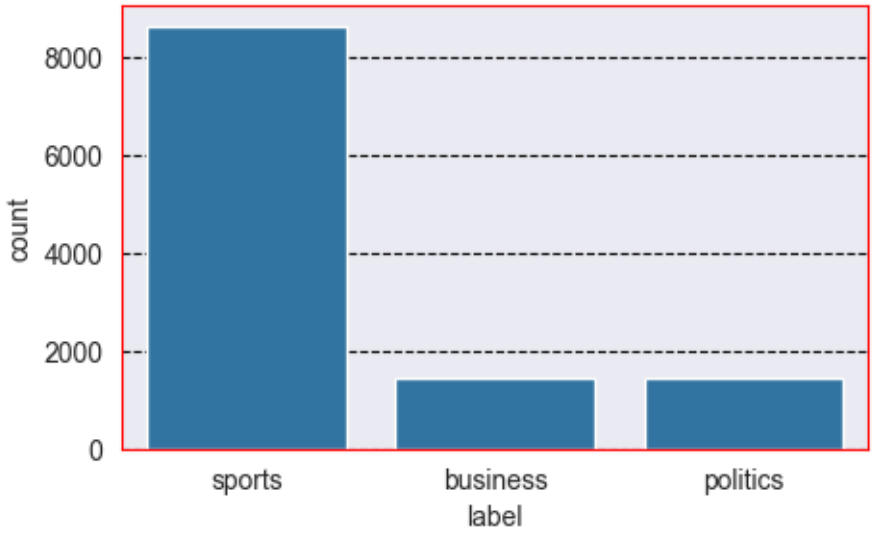
	text	label
0	(reuters) - carlos tevez sealed his move to ju…	sports
1	if professional pride and strong defiance can …	sports
2	palermo, sicily - roberta vinci beat top-seede…	sports
3	spain's big two soccer teams face a pair of it…	sports
4	the argentine soccer club san lorenzo complete…	sports

[4]:
```
# see class distribution - multiclass problem
df = data.copy()
print(df['label'].value_counts())

plt.figure(figsize=(5,3))
sns.countplot(data=df, x='label')
```

```
label
sports      8639
politics    1451
business    1429
Name: count, dtype: int64
```

[4]: <Axes: xlabel='label', ylabel='count'>



[5]:
```
# classes are imbalanced
from sklearn.utils.class_weight import compute_class_weight

class_weights = compute_class_weight(class_weight='balanced', classes=np.
 ↪unique(df['label']), y=df['label'].values)
class_weight_dict = {0: class_weights[0], 1: class_weights[1], 2:␣
 ↪class_weights[2]}
print(class_weight_dict)
```

```
{0: 2.6869605784931188, 1: 2.646220997013554, 2: 0.4444573060153567}
```

[6]:
```
# mapping of label to codes
mapped_classes = df.label.astype('category')
hm_class = dict(enumerate(mapped_classes.cat.categories))
```

```
print(hm_class)

df['label'] = df.label.astype('category').cat.codes

# 1 hot encoding
one_hot = pd.get_dummies(df['label'])
df['label'] = one_hot.apply(lambda row: row.values, axis=1)
```

```
{0: 'business', 1: 'politics', 2: 'sports'}
```

[7]:
```
# Note: in case of Bert, removing the stopwords might actually worsen the␣
 ↪metrics
df['text'] = df['text'].str.replace(r'[^\w\s]', ' ', regex=True)
df['text'] = df['text'].str.replace(r'[\{\}\[\]\(\)]', '', regex=True)
```

[8]:
```
# convert to huggingface dataset format
ds = Dataset.from_pandas(df)
ds
```

[8]:
```
Dataset({
    features: ['text', 'label'],
    num_rows: 11519
})
```

[9]:
```
# train test validation splits
train_test_splits = ds.train_test_split(test_size=0.2)
test_validation_splits = train_test_splits['test'].train_test_split(test_size=0.
 ↪5)

# collate all in a dict
tweet_dataset = DatasetDict({
    'train': train_test_splits['train'],
    'test': test_validation_splits['test'],
    'valid': test_validation_splits['train']
})

tweet_dataset
```

[9]:
```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 9215
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 1152
    })
    valid: Dataset({
```

```
        features: ['text', 'label'],
        num_rows: 1152
    })
})
```

0.1.1 Tokenizer

[10]:
```
# loading a pre-trained BERT tokenizer that corresponds to the␣
 ↪"bert-base-uncased" model.
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

[11]:
```
# preprocess
def preprocess(ds):
    return tokenizer(
        ds['text'],
        padding='max_length',
        truncation=True,
        max_length=64
    )

# When batched=True, the preprocess function will receive a batch of samples as␣
 ↪input instead of a single sample. Faster.
tokenized_dataset = tweet_dataset.map(preprocess, batched=True, batch_size=32,␣
 ↪remove_columns=["text"])

train_dataset = tokenized_dataset['train'].with_format('tensorflow')
eval_dataset = tokenized_dataset['valid'].with_format('tensorflow')
test_dataset = tokenized_dataset['test'].with_format('tensorflow')
```

```
Map:   0%|          | 0/9215 [00:00<?, ? examples/s]
Map:   0%|          | 0/1152 [00:00<?, ? examples/s]
Map:   0%|          | 0/1152 [00:00<?, ? examples/s]
```

[12]:
```
# batching
batch_size=16

def preprocess_batch_dataset(dataset):
    # When you pass in a tensor (or a tuple of tensors), from_tensor_slices␣
 ↪treats each entry (or "slice") in the tensor as a separate element in the␣
 ↪dataset.
    dataset = tf.data.Dataset.from_tensor_slices((
        {x: dataset[x] for x in tokenizer.model_input_names},
        dataset["label"]
        # tf.keras.utils.to_categorical(dataset["label"], num_classes=3)
    ))
```

```
    # shuffle and create batches
    # buffer_size=1000 specifies the number of elements from which to sample␣
 ↪when shuffling
    dataset = dataset.shuffle(buffer_size=len(dataset)).batch(batch_size)
    return dataset

# batch dataset
train_dataset_bt, eval_dataset_bt, test_dataset_bt =␣
 ↪[preprocess_batch_dataset(ds) for ds in [train_dataset, eval_dataset,␣
 ↪test_dataset]]
```

[13]:
```
# check if batch created
first_batch = next(iter(train_dataset_bt.take(1)))
print(first_batch[0]['input_ids'].shape)
```

```
(16, 64)
```

[14]:
```
# save ground truth y in test
labels_list = []
for batch in test_dataset_bt:
    labels_list.extend(batch[1].numpy())

test_truth = np.argmax(labels_list, axis=1)
```

```
2024-10-17 03:05:34.587556: I tensorflow/core/framework/local_rendezvous.cc:404]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

[15]:
```
from collections import Counter
Counter(test_truth)
```

[15]:
```
Counter({2: 864, 0: 146, 1: 142})
```

0.1.2 Define Model

[16]:
```
# load a pre-trained tf model
# hf trainer class - Trainer is primarily designed for PyTorch models, not␣
 ↪TensorFlow models.
model = TFAutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    problem_type="multi_label_classification",
    num_labels=3,
    id2label=hm_class,
    label2id={v: k for k, v in hm_class.items()}
)
```

```
All PyTorch model weights were used when initializing
TFBertForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were
```

```
not initialized from the PyTorch model and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

0.1.3 Training

[17]:
```python
def lr_scheduler(epoch):
    epoch_tensor = tf.convert_to_tensor(epoch, dtype=tf.float32)
    if epoch > 1:
        return 0.00005 * tf.math.exp(-epoch_tensor)
    else:
        return 0.00005
```

[18]:
```python
# metrics
def f1_micro(y_true, y_pred):
    y_true_indices = tf.argmax(y_true, axis=-1)
    y_pred_indices = tf.argmax(y_pred, axis=-1)
    return f1_score(y_true_indices.numpy(), y_pred_indices.numpy(),␣
 ↪average='micro')

def f1_macro(y_true, y_pred):
    y_true_indices = tf.argmax(y_true, axis=-1)
    y_pred_indices = tf.argmax(y_pred, axis=-1)
    return f1_score(y_true_indices.numpy(), y_pred_indices.numpy(),␣
 ↪average='macro')

# Wrapping them in a TensorFlow function for compatibility
@tf.function
def f1_micro_metric(y_true, y_pred):
    return tf.py_function(f1_micro, [y_true, y_pred], tf.double)

@tf.function
def f1_macro_metric(y_true, y_pred):
    return tf.py_function(f1_macro, [y_true, y_pred], tf.double)
```

[19]:
```python
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=5e-5),
    loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
    metrics=[tf.metrics.CategoricalAccuracy(), f1_micro_metric, f1_macro_metric]
)

history = model.fit(
    train_dataset_bt,
    epochs=3,
    validation_data=eval_dataset_bt,
    # class_weight=class_weight_dict,
```

```
    callbacks=[tf.keras.callbacks.LearningRateScheduler(lr_scheduler)],
    verbose=True
)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs
slowly on M1/M2 Macs, please use the legacy TF-Keras optimizer instead, located
at `tf.keras.optimizers.legacy.Adam`.

Epoch 1/3
576/576 [==============================] - 824s 1s/step - loss: 0.1495 -
categorical_accuracy: 0.9485 - f1_micro_metric: 0.9485 - f1_macro_metric: 0.8554
- val_loss: 0.0805 - val_categorical_accuracy: 0.9714 - val_f1_micro_metric:
0.9714 - val_f1_macro_metric: 0.9091 - lr: 5.0000e-05
Epoch 2/3
576/576 [==============================] - 807s 1s/step - loss: 0.0514 -
categorical_accuracy: 0.9844 - f1_micro_metric: 0.9844 - f1_macro_metric: 0.9544
- val_loss: 0.0761 - val_categorical_accuracy: 0.9748 - val_f1_micro_metric:
0.9748 - val_f1_macro_metric: 0.9418 - lr: 5.0000e-05
Epoch 3/3
576/576 [==============================] - 806s 1s/step - loss: 0.0193 -
categorical_accuracy: 0.9945 - f1_micro_metric: 0.9945 - f1_macro_metric: 0.9853
- val_loss: 0.0757 - val_categorical_accuracy: 0.9757 - val_f1_micro_metric:
0.9757 - val_f1_macro_metric: 0.9452 - lr: 6.7668e-06
```

[20]:
```
# visualisation
def plot_metric(ax, x, y_train, y_val, title, ylabel, xlabel):
    ax.plot(x, y_train, label='Train')
    ax.plot(x, y_val, label='Validation')
    ax.set_title(title)
    ax.set_ylabel(ylabel)
    ax.set_xlabel(xlabel)
    ax.legend(loc='upper left')

def plot_history(history):
    # 2x2 grid
    fig, axs = plt.subplots(2, 2, figsize=(7, 5))  # Adjust figsize for better␣
 ↪visibility

    # Epochs
    epochs = range(1, len(history.history['loss']) + 1)

    # Loss
    plot_metric(axs[0, 0], epochs, history.history['loss'], history.
 ↪history['val_loss'],
                'Model Loss', 'Loss', 'Epoch')
    # Accuracy
    plot_metric(axs[0, 1], epochs, history.history['categorical_accuracy'],␣
 ↪history.history['val_categorical_accuracy'],
```

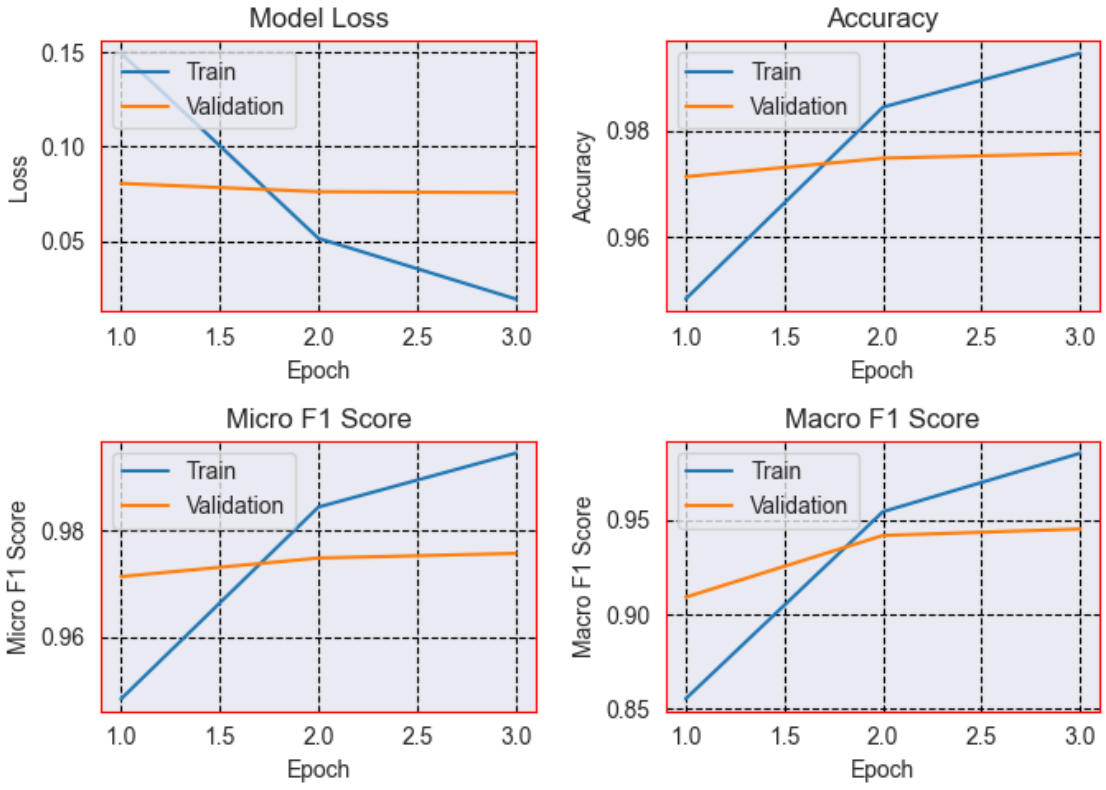
```
                'Accuracy', 'Accuracy', 'Epoch')

    # Micro F1 Score
    plot_metric(axs[1, 0], epochs, history.history['f1_micro_metric'], history.
 ↪history['val_f1_micro_metric'],
                'Micro F1 Score', 'Micro F1 Score', 'Epoch')

    # Macro F1 Score
    plot_metric(axs[1, 1], epochs, history.history['f1_macro_metric'], history.
 ↪history['val_f1_macro_metric'],
                'Macro F1 Score', 'Macro F1 Score', 'Epoch')

    # Adjust layout
    plt.tight_layout()
    plt.show()

plot_history(history)
```



[22]:
```
# inference on test set
test_loss, test_acc, test_f1_micro, test_f1_macro = model.
 ↪evaluate(test_dataset_bt, verbose=2)
```

```
print("Accuracy:", test_acc)
print("Micro F1 Score:", test_f1_micro)
print("Macro F1 Score:", test_f1_macro)
```

```
72/72 - 32s - loss: 0.0859 - categorical_accuracy: 0.9757 - f1_micro_metric:
0.9757 - f1_macro_metric: 0.9336 - 32s/epoch - 447ms/step
Accuracy: 0.9756944179534912
Micro F1 Score: 0.9756944179534912
Macro F1 Score: 0.9336167573928833
```