coursework_2

April 24, 2023

1 Coursework IDA

1.1 Task 1

1.2 1.1.

Implement and train a method for automatically classifying texts in the FiQA sentiment analysis dataset as positive, neutral or negative. Refer to the labs, lecture materials and textbook to identify a suitable method. In your report: • Briefly explain how your chosen method works and its main strengths and limitations; • Describe the preprocessing steps and the features you use to represent each text instance; • Explain why you chose those features and preprocessing steps and hypothesise how they will affect your results; • Briefly describe your software implementation. (10 marks)

[2]:
```
%load_ext autoreload
%autoreload 2

# Use HuggingFace's datasets library to access the financial_phrasebank dataset
from datasets import load_dataset

import numpy as np
```

[3]:
```
train_files = [
    # 'data_cache/FiQA_ABSA_task1/task1_headline_ABSA_train.json',
    'data_cache/FiQA_ABSA_task1/task1_post_ABSA_train.json'
]
```

[30]:
```
import json

def load_fiqa_sa_from_json(json_files):
    train_text = []
    train_labels = []

    # iterate through each tweet file
    for file in json_files:
        # open file in read mode, with method closes file after getting data␣
↪stream
        with open(file, 'r', encoding = 'utf8') as handle:
            # load file object and convert into json object
            dataf = json.load(handle)
```

```python
        dataf_text = [dataf[k]["sentence"] for k in dataf.keys()]
        # print(len(dataf_text))
        train_text.extend(dataf_text)

        dataf_labels = [float(dataf[k]["info"][0]["sentiment_score"]) for k in␣
 ↪dataf.keys()]
        # print(len(dataf_labels))
        train_labels.extend(dataf_labels)

    train_text = np.array(train_text)
    train_labels = np.array(train_labels)

    return train_text, train_labels


def threshold_scores(scores):
    """
    Convert sentiment scores to discrete labels.
    0 = negative.
    1 = neutral.
    2 = positive.
    """
    labels = []
    for score in scores:
        if score < -0.2:
            labels.append(0)
        elif score > 0.2:
            labels.append(2)
        else:
            labels.append(1)

    return np.array(labels)


all_text, all_labels = load_fiqa_sa_from_json(train_files)

print(f'Number of instances: {len(all_text)}')
print(f'Number of labels: {len(all_labels)}')

all_labels = threshold_scores(all_labels)
print(f'Number of negative labels: {np.sum(all_labels==0)}')
print(f'Number of neutral labels: {np.sum(all_labels==1)}')
print(f'Number of positive labels: {np.sum(all_labels==2)}')
```

```
Number of instances: 675
Number of labels: 675
```

```
Number of negative labels: 203
Number of neutral labels: 74
Number of positive labels: 398
```

[41]:
```python
type(load_fiqa_sa_from_json(train_files))
```

[41]:
```
tuple
```

[47]:
```python
print(len(load_fiqa_sa_from_json(train_files)[0]))
```

```
675
```

[6]:
```python
from sklearn.model_selection import train_test_split

# Split test data from training data
train_documents, test_documents, train_labels, test_labels = train_test_split(
    all_text,
    all_labels,
    test_size=0.2,
    stratify=all_labels  # make sure the same proportion of labels is in the␣
 ↪test set and training set
)

# Split validation data from training data
train_documents, val_documents, train_labels, val_labels = train_test_split(
    train_documents,
    train_labels,
    test_size=0.15,
    stratify=train_labels  # make sure the same proportion of labels is in the␣
 ↪test set and training set
)

print(f'Number of training instances = {len(train_documents)}')
print(f'Number of validation instances = {len(val_documents)}')
print(f'Number of test instances = {len(test_documents)}')
```

```
Number of training instances = 459
Number of validation instances = 81
Number of test instances = 135
```

[7]:
```python
print(f'What does one instance look like from the training set?␣
 ↪\n\n{train_documents[233]}')
print(f'...and here is its corresponding label \n\n{train_labels[233]}')
```

```
What does one instance look like from the training set?

$TWTR The best scenario going forward is this stock slowly falling
everyday…Which is quite probable…
…and here is its corresponding label
```

0

[9]:
```
from sklearn.feature_extraction.text import CountVectorizer
from nltk import word_tokenize

# CountVectorizer can do its own tokenization, but for consistency we want to
# carry on using WordNetTokenizer. We write a small wrapper class to enable␣
↪this:
class Tokenizer(object):
    def __call__(self, tweets):
        return word_tokenize(tweets)

vectorizer = CountVectorizer(tokenizer=Tokenizer())  # construct the vectorizer

vectorizer.fit(train_documents)  # Learn the vocabulary
X_train = vectorizer.transform(train_documents)  # extract training set bags of␣
↪words
X_val = vectorizer.transform(val_documents)  # extract test set bags of words
X_test = vectorizer.transform(test_documents)  # extract test set bags of words
```

```
C:\Users\laure\anaconda3\envs\data_analytics\lib\site-
packages\sklearn\feature_extraction\text.py:516: UserWarning: The parameter
'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

[ ]:
```

```

1.3 Naive Bayes Classifier

[10]:
```
# WRITE YOUR CODE HERE

from sklearn.naive_bayes import MultinomialNB

classifier = MultinomialNB()
classifier.fit(X_train, train_labels)
```

[10]:
```
MultinomialNB()
```

[14]:
```
y_val_pred = classifier.predict(X_val)
```

[16]:
```
# WRITE YOUR CODE HERE
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, classification_report

acc = accuracy_score(val_labels, y_val_pred)
print(f'Accuracy = {acc}')
```

```
prec = precision_score(val_labels, y_val_pred, average='macro')
print(f'Precision (macro average) = {prec}')

rec = recall_score(val_labels, y_val_pred, average='macro')
print(f'Recall (macro average) = {rec}')

f1 = f1_score(val_labels, y_val_pred, average='macro')
print(f'F1 score (macro average) = {f1}')

# We can get all of these with a per-class breakdown using␣
↪classification_report:
print(classification_report(val_labels, y_val_pred))
```

```
Accuracy = 0.6419753086419753
Precision (macro average) = 0.42821606254442074
Recall (macro average) = 0.4236111111111116
F1 score (macro average) = 0.4071700991609459
              precision    recall  f1-score   support

           0       0.64      0.38      0.47        24
           1       0.00      0.00      0.00         9
           2       0.64      0.90      0.75        48

    accuracy                           0.64        81
   macro avg       0.43      0.42      0.41        81
weighted avg       0.57      0.64      0.58        81


C:\Users\laure\anaconda3\envs\data_analytics\lib\site-
packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\laure\anaconda3\envs\data_analytics\lib\site-
packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\laure\anaconda3\envs\data_analytics\lib\site-
packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\laure\anaconda3\envs\data_analytics\lib\site-
packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[20]:
```
import numpy as np

vocabulary = vectorizer.vocabulary_

### CHANGE THE NAME OF THE CLASSIFIER VARIABLE BELOW TO USE YOUR TRAINED
↪CLASSIFIER
feat_likelihoods = np.exp(classifier.feature_log_prob_)  # Use exponential to
↪convert the logs back to probabilities
###

# WRITE YOUR CODE HERE
print(feat_likelihoods[:, vocabulary['a']])
print(feat_likelihoods[:, vocabulary['it']])
```

```
[0.0059322  0.00476049 0.00970874]
[0.00360169 0.00327284 0.00347776]
```

[ ]:

2 Logistic Regression Classifier

[27]:
```
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression()
classifier.fit(X_train, train_labels)
```

[27]:
```
LogisticRegression()
```

[ ]:

[28]:
```
y_val_pred = classifier.predict(X_val)
```

[ ]:

2.1 1.2. Evaluate Method

Evaluate your method, then interpret and discuss your results. Include the following points: • Define your performance metrics and state their limitations; • Describe the testing procedure (e.g., how you used each split of the dataset); • Show your results using suitable plots or tables; • How could you improve the method or experimental process? Consider the errors that your method makes.
(9 marks)

[ ]:

[29]:

```
# WRITE YOUR CODE HERE
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, classification_report

acc = accuracy_score(val_labels, y_val_pred)
print(f'Accuracy = {acc}')

prec = precision_score(val_labels, y_val_pred, average='macro')
print(f'Precision (macro average) = {prec}')

rec = recall_score(val_labels, y_val_pred, average='macro')
print(f'Recall (macro average) = {rec}')

f1 = f1_score(val_labels, y_val_pred, average='macro')
print(f'F1 score (macro average) = {f1}')

# We can get all of these with a per-class breakdown using␣
↪classification_report:
print(classification_report(val_labels, y_val_pred))
```

```
Accuracy = 0.6790123456790124
Precision (macro average) = 0.6187739463601533
Recall (macro average) = 0.5092592592592592
F1 score (macro average) = 0.5195857950574932
              precision    recall  f1-score   support

           0       0.67      0.58      0.62        24
           1       0.50      0.11      0.18         9
           2       0.69      0.83      0.75        48

    accuracy                           0.68        81
   macro avg       0.62      0.51      0.52        81
weighted avg       0.66      0.68      0.65        81
```

[ ]:

[ ]:

[ ]:

[ ]:

3 1.3 Common Themes & Topics

1.3. Can you identify common themes or topics associated with negative sentiment or positive sentiment in this dataset? • Explain the method you use to identify themes or topics; • Show your results (e.g., by listing or visualising example topics or themes); • Interpret the results and summarise the limitations of your approach. (12 marks)

[30]:
```python
n_feats_to_show = 10

# Flip the index so that values are keys and keys are values:
keys = vectorizer.vocabulary_.values()
values = vectorizer.vocabulary_.keys()
vocab_inverted = dict(zip(keys, values))

for c, weights_c in enumerate(classifier.coef_):
    print(f'\nWeights for class {c}:\n')
    strongest_idxs = np.argsort(weights_c)[-n_feats_to_show:]

    for idx in strongest_idxs:
        print(f'{vocab_inverted[idx]} with weight {weights_c[idx]}')
```

```
Weights for class 0:

been with weight 0.5012708848845197
bearish with weight 0.525706926913856
weak with weight 0.5308854467781363
recall with weight 0.5468370276745733
sbux with weight 0.5980631806752739
spy with weight 0.611452390341226
downside with weight 0.6743346625903969
lower with weight 0.7068927456779212
down with weight 0.9456657412047452
short with weight 1.0461304693067406

Weights for class 1:

soon with weight 0.46699155102808954
was with weight 0.4917629729258395
aapl with weight 0.49624868831367863
rating with weight 0.5175530058816274
have with weight 0.6119601108805572
but with weight 0.6319378746229999
today with weight 0.6529675399272293
sideways with weight 0.7209651479558604
nvda with weight 0.7209651479558604
not with weight 0.7554033901276137

Weights for class 2:

positive with weight 0.4865184129196852
bounce with weight 0.4961202985972643
```

```
high with weight 0.5240015256010992
run with weight 0.5371066592684799
calls with weight 0.6906878948511527
bullish with weight 0.701412411149405
up with weight 0.7124077727648621
higher with weight 0.7184575426067357
buy with weight 0.8678716809761933
long with weight 1.0372874435680175
```

[ ]:

```

```

3.0.1 Topics

[ ]:

```

```

[100]:

```python
pos_index = all_labels == 2  # compare predictions to gold labels
neg_index = all_labels == 0  # compare predictions to gold labels
# get the text of tweets where the classifier made an error:
pos_tweets = np.array(all_text)[pos_index]
neg_tweets = np.array(all_text)[neg_index]
```

[101]:

```python
#type(pos_tweets)
print(pos_tweets[0])
print(neg_tweets[0])
```

```
Slowly adding some $FIO here but gotta be careful. This will be one of biggest
winners in 2012
I am not optimistic about $amzn both fundementals and charts look like poopoo
this quarter.
```

[ ]:

```python
processed_pos = []
processed_neg = []
```

[105]:

```python
from nltk.stem import WordNetLemmatizer
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS # find stopwords

np.random.seed(400)  # We fix the random seed to ensure we get consistent␣
 ↪results when we repeat the lab.

# Tokenize and lemmatize
def preprocess(text):
    result=[]
    for token in simple_preprocess(text) :  # Tokenize, remove very short and␣
 ↪very long words, convert to lower case, remove words containing non-letter␣
 ↪characters
        if token not in STOPWORDS:
            result.append(WordNetLemmatizer().lemmatize(token, 'v'))
```

```python
    return result

# Create lists of preprocessed documents
for tweet in pos_tweets:
    processed_pos.append(preprocess(tweet))

for tweet in neg_tweets:
    processed_neg.append(preprocess(tweet))
```

[107]:
```python
print(processed_pos[0])
print(processed_neg[0])
```

```
['slowly', 'add', 'fio', 'gotta', 'careful', 'biggest', 'winners']
['optimistic', 'amzn', 'fundementals', 'chart', 'look', 'like', 'poopoo',
'quarter']
```

[ ]:
```python

```

[112]:
```python
from gensim.corpora import Dictionary

dictionary_pos = Dictionary(processed_pos) # construct word<->id mappings - it␣
 ↪does it in alphabetical order
print(dictionary_pos)

pos_bow_corpus = [dictionary_pos.doc2bow(tweet) for tweet in processed_pos]

dictionary_neg = Dictionary(processed_neg) # construct word<->id mappings - it␣
 ↪does it in alphabetical order
print(dictionary_neg)

neg_bow_corpus = [dictionary_neg.doc2bow(tweet) for tweet in processed_neg]
```

```
Dictionary(1514 unique tokens: ['add', 'biggest', 'careful', 'fio', 'gotta']…)
Dictionary(887 unique tokens: ['amzn', 'chart', 'fundementals', 'like',
'look']…)
```

[113]:
```python
len(pos_bow_corpus)
```

[113]: 796

[114]:
```python
len(neg_bow_corpus)
```

[114]: 203

[117]:
```python
from gensim.models import LdaModel

lda_pos_model =  LdaModel(pos_bow_corpus,
```

```
                          num_topics=10,
                          id2word=dictionary_pos,
                          passes=10,
                        )

lda_neg_model =  LdaModel(neg_bow_corpus,
                          num_topics=10,
                          id2word=dictionary_neg,
                          passes=10,
                        )
```

[118]:
```
'''
For each topic, we will explore the words occuring in that topic and its␣
 ↪relative weight
'''
for idx, topic in lda_pos_model.print_topics(-1):
    print("Pos Topic: {} \nWords: {}".format(idx, topic ))
    print("\n")

for idx, topic in lda_neg_model.print_topics(-1):
    print("Neg Topic: {} \nWords: {}".format(idx, topic ))
    print("\n")
```

```
Pos Topic: 0
Words: 0.059*"https" + 0.016*"buy" + 0.014*"upgrade" + 0.012*"googl" +
0.011*"fb" + 0.009*"tsla" + 0.009*"price" + 0.009*"sell" + 0.009*"trade" +
0.009*"time"


Pos Topic: 1
Words: 0.030*"http" + 0.024*"stks" + 0.023*"buy" + 0.021*"bullish" +
0.020*"aapl" + 0.019*"breakout" + 0.018*"stock" + 0.012*"high" + 0.012*"double"
+ 0.010*"long"


Pos Topic: 2
Words: 0.024*"good" + 0.021*"long" + 0.016*"close" + 0.016*"short" + 0.015*"buy"
+ 0.014*"aapl" + 0.014*"dividend" + 0.011*"hold" + 0.010*"https" + 0.010*"look"


Pos Topic: 3
Words: 0.023*"aapl" + 0.023*"http" + 0.023*"stks" + 0.020*"today" + 0.017*"look"
+ 0.017*"strong" + 0.013*"bounce" + 0.012*"nice" + 0.011*"buy" + 0.011*"break"


Pos Topic: 4
Words: 0.043*"long" + 0.017*"aapl" + 0.015*"hod" + 0.013*"pop" + 0.013*"tsla" +
0.013*"small" + 0.012*"buy" + 0.011*"stks" + 0.011*"http" + 0.010*"https"
```

Pos Topic: 5
Words: 0.039*"http" + 0.038*"stks" + 0.013*"https" + 0.013*"today" + 0.013*"new" + 0.012*"stock" + 0.011*"higher" + 0.009*"chart" + 0.009*"come" + 0.009*"volume"

Pos Topic: 6
Words: 0.032*"http" + 0.030*"stks" + 0.023*"call" + 0.018*"tsla" + 0.016*"buy" + 0.014*"signal" + 0.013*"https" + 0.012*"long" + 0.010*"pt" + 0.009*"fb"

Pos Topic: 7
Words: 0.035*"stks" + 0.035*"http" + 0.022*"go" + 0.013*"bounce" + 0.011*"long" + 0.011*"green" + 0.011*"apple" + 0.010*"company" + 0.009*"aapl" + 0.008*"time"

Pos Topic: 8
Words: 0.030*"https" + 0.017*"stks" + 0.017*"http" + 0.015*"long" + 0.014*"ma" + 0.011*"increase" + 0.010*"look" + 0.009*"good" + 0.009*"base" + 0.009*"strategy"

Pos Topic: 9
Words: 0.022*"like" + 0.021*"look" + 0.020*"long" + 0.014*"day" + 0.014*"https" + 0.012*"amzn" + 0.012*"run" + 0.012*"low" + 0.011*"stks" + 0.011*"http"

Neg Topic: 0
Words: 0.033*"short" + 0.031*"stks" + 0.031*"http" + 0.020*"aapl" + 0.020*"look" + 0.017*"like" + 0.011*"fb" + 0.011*"lower" + 0.011*"sell" + 0.009*"rejoice"

Neg Topic: 1
Words: 0.020*"short" + 0.017*"market" + 0.014*"downgrade" + 0.013*"aapl" + 0.011*"tsla" + 0.010*"share" + 0.010*"time" + 0.010*"day" + 0.010*"lower" + 0.010*"stks"

Neg Topic: 2
Words: 0.011*"weak" + 0.011*"gs" + 0.011*"short" + 0.011*"good" + 0.011*"look" + 0.011*"downside" + 0.011*"bay" + 0.011*"million" + 0.006*"break" + 0.006*"target"

Neg Topic: 3
Words: 0.021*"sell" + 0.021*"stks" + 0.021*"http" + 0.016*"min" + 0.011*"aapl" + 0.011*"qqq" + 0.011*"spy" + 0.011*"signal" + 0.011*"rt" + 0.011*"retest"

```
Neg Topic: 4
Words: 0.029*"tsla" + 0.029*"https" + 0.018*"short" + 0.018*"get" + 0.018*"sell"
+ 0.010*"model" + 0.010*"recall" + 0.009*"new" + 0.009*"authentication" +
0.009*"jump"


Neg Topic: 5
Words: 0.057*"https" + 0.050*"tsla" + 0.049*"recall" + 0.039*"model" +
0.035*"tesla" + 0.017*"seat" + 0.017*"suvs" + 0.013*"stks" + 0.013*"http" +
0.013*"fb"


Neg Topic: 6
Words: 0.023*"http" + 0.022*"spy" + 0.019*"stks" + 0.015*"buy" + 0.011*"bearish"
+ 0.011*"chart" + 0.011*"support" + 0.008*"fb" + 0.008*"gap" + 0.008*"day"


Neg Topic: 7
Words: 0.041*"https" + 0.029*"sbux" + 0.020*"deutsche" + 0.020*"bank" +
0.020*"downgrade" + 0.018*"starbucks" + 0.016*"short" + 0.011*"miss" +
0.011*"ntap" + 0.011*"stks"


Neg Topic: 8
Words: 0.024*"short" + 0.017*"https" + 0.012*"aapl" + 0.012*"fall" +
0.012*"today" + 0.008*"sell" + 0.008*"spy" + 0.008*"go" + 0.008*"low" +
0.008*"hit"


Neg Topic: 9
Words: 0.020*"short" + 0.020*"spy" + 0.020*"close" + 0.010*"week" +
0.010*"continue" + 0.010*"think" + 0.010*"gain" + 0.010*"market" + 0.010*"ma" +
0.010*"resistance"
```

3.0.2 Individual Topic Distribution

[121]:
```python
test_document_idx = 10
unseen_document = pos_tweets[test_document_idx]
print(unseen_document)

#print(f' This document is from newsgroup {newsgroups_test.
 ↪target_names[newsgroups_test.target[test_document_idx]]}')
```

```
# Data preprocessing step for the unseen document - It is the same␣
 ↪preprocessing we have performed for the training data
bow_vector = dictionary.doc2bow(preprocess(unseen_document))

for idx, count in bow_vector:
    print(f'{dictionary[idx]}: {count}')
```

```
$TZOO a close above 28.64 and we are ready to rock and roll
close: 1
ready: 1
rock: 1
roll: 1
tzoo: 1
```

[122]:
```
topic_distribution = lda_model[bow_vector]

for index, probability in sorted(topic_distribution, key=lambda tup: -1*tup[1]):
    print("Index: {}\nProbability: {}\t Topic: {}".format(index, probability,␣
 ↪lda_model.print_topic(index, 5)))
```

```
Index: 14
Probability: 0.4022486209869385  Topic: 0.016*"low" + 0.015*"breakout" +
0.011*"long" + 0.011*"supply" + 0.011*"master"
Index: 6
Probability: 0.2308562695980072  Topic: 0.038*"buy" + 0.023*"https" +
0.019*"stks" + 0.019*"http" + 0.019*"long"
Index: 17
Probability: 0.22521528601646423         Topic: 0.035*"https" + 0.023*"time" +
0.018*"dip" + 0.018*"upside" + 0.018*"move"
```

[ ]:
```
# make list of tuples ready for model training

train_set = list(zip(list_a, list_b))
```

[ ]:

[ ]:

[ ]:

[ ]:

3.1 Task 2: Named Entity Recognition (max. 19%)

In scientific research, information extraction can help researchers to discover relevant findings from across a wide body of literature. As a first step, your task is to build a tool for named entity recognition in scientific journal article abstracts. We will be working with the BioNLP 2004 dataset of abstracts from MEDLINE, a database containing journal articles from fields including medicine and pharmacy. The data was collected by searching for the terms ‘human’, ‘blood cells’ and 'transcription factors', and then annotated with five entity types: DNA, protein, cell type, cell line, RNA.

More information can be found in the paper: https://aclanthology.org/W04-1213.pdf . We provide a cache of the data and code for loading the data in 'data_loader_demo' in our Github repository, https://github.com/uob-TextAnalytics/intro-labs-public. This script downloaded the data from HuggingFace, where you can also find more information about the dataset: https://huggingface.co/datasets/tner/bionlp2004 .

The data is presented in this paper: Nigel Collier, Tomoko Ohta, Yoshimasa Tsuruoka, Yuka Tateisi, and Jin-Dong Kim. 2004. Introduction to the Bio-entity Recognition Task at JNLPBA. In Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP), pages 73–78, Geneva, Switzerland. COLING.

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```