Course: A.I. & Machine Learning

Student: Paolo Bassi

Sentiment Analysis using a Multinomial Naïve Bayes classifier

Build a vocabulary

In [1]:

```
import collections
import os


def read_document(filename):
    """Read the file and returns a list of words."""
    f = open(filename, encoding="utf8")
    text = f.read()
    f.close()
    words = []
    # The three following lines replace punctuation symbols with
    # spaces.
    p = "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"
    table = str.maketrans(p, " " * len(p))
    text = text.translate(table)
    for w in text.split():
        if len(w) > 2:
            words.append(w.lower())
    return words


def write_vocabulary(voc, filename, n):
    """Write the n most frequent words to a file."""
    f = open(filename, "w")
    for word, count in sorted(voc.most_common(n)):
        print(word, file=f)
    f.close()


# The script reads all the documents in the smalltrain directory, uses
# the to form a vocabulary, writes it to the 'vocabulary.txt' file.
voc = collections.Counter()
for f in os.listdir("smalltrain/pos"):
    voc.update(read_document("smalltrain/pos/" + f))
for f in os.listdir("smalltrain/neg"):
    voc.update(read_document("smalltrain/neg/" + f))
write_vocabulary(voc, "vocabulary.txt", 1000)
```

Extract the features

In [14]:

```
import numpy as np
import os


def load_vocabulary(filename):
```

```
    """Load the vocabulary and returns it.

    The return value is a dictionary mapping words to numerical
indices (n).

    """
    f = open(filename)
    n = 0
    voc = {}
    for w in f.read().split():
        voc[w] = n
        n += 1
    f.close()
    return voc


def read_document(filename, voc):
    """Read a document and return its BoW representation."""
    f = open(filename, encoding="utf8")
    text = f.read()
    f.close()
    p = "!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~"
    table = str.maketrans(p, " " * len(p))
    text = text.translate(table)
    # Start with all zeros
    bow = np.zeros(len(voc))
    for w in text.split():
        # If the word is the vocabulary...
        if w in voc:
            # ...increment the proper counter.
            index = voc[w]
            bow[index] += 1
    return bow


# The script compute the BoW representation of all the training
# documents (the rows), and labels (class 0 = neg. documents, class 1 = pos documents).

#Train

voc = load_vocabulary("vocabulary.txt")
documents = []
labels = []
for f in os.listdir("smalltrain/pos"):
    documents.append(read_document("smalltrain/pos/" + f, voc))
    labels.append(1)
for f in os.listdir("smalltrain/neg"):
    documents.append(read_document("smalltrain/neg/" + f, voc))
    labels.append(0)

# np.stack transforms the list of vectors into a 2D array.
X = np.stack(documents)
Y = np.array(labels)

# The following line append the labels Y as additional column of the
# array of features so that it can be passed to np.savetxt.
data = np.concatenate([X, Y[:, None]], 1)
np.savetxt("train.txt.gz", data)
```

In [7]:
```
#Test

voc = load_vocabulary("vocabulary.txt")
documents = []
labels = []
for f in os.listdir("test/pos"):
    documents.append(read_document("test/pos/" + f, voc))
    labels.append(1)
for f in os.listdir("test/neg"):
    documents.append(read_document("test/neg/" + f, voc))
    labels.append(0)

# np.stack transforms the list of vectors into a 2D array.
X = np.stack(documents)
Y = np.array(labels)

# The following line append the labels Y as additional column of the
# array of features so that it can be passed to np.savetxt.
data = np.concatenate([X, Y[:, None]], 1)
np.savetxt("test.txt.gz", data)
```

In [8]:
```
#Validation

voc = load_vocabulary("vocabulary.txt")
documents = []
labels = []
for f in os.listdir("validation/pos"):
    documents.append(read_document("validation/pos/" + f, voc))
    labels.append(1)
for f in os.listdir("validation/neg"):
    documents.append(read_document("validation/neg/" + f, voc))
    labels.append(0)

# np.stack transforms the list of vectors into a 2D array.
X = np.stack(documents)
Y = np.array(labels)

# The following line append the labels Y as additional column of the
# array of features so that it can be passed to np.savetxt.
data = np.concatenate([X, Y[:, None]], 1)
np.savetxt("validation.txt.gz", data)
```

Train a classifier

In [16]:

```python
import numpy as np

# X = features
# Y = labels

def train_nb(X, Y): #Train a binary NB classifier.

    m = X.shape[0]
    n = X.shape[1]

    # + 1 for the Laplacian smoothing
    pos_c = X[Y == 1, :].sum(0)  # counter c
    pos_p = (pos_c + 1) / (pos_c.sum() + n)  # phi π

    neg_c = X[Y == 0, :].sum(0)
    neg_p = (neg_c + 1) / (neg_c.sum() + n)

    prior_pos = Y.sum() / m
    prior_neg = 1 - prior_pos

    w = np.log(pos_p) - np.log(neg_p)

    # Estimate P(0) and P(1) and compute b
    b = np.log(prior_pos) - np.log(prior_neg)
    return w, b


def inference_nb(X, w, b): #Prediction of a binary NB classifier.
    logits = X @ w + b
    return (logits > 0).astype(int)


# The script loads the training data and train a classifier.

data = np.loadtxt("train.txt.gz")
X = data[:, :-1]
Y = data[:, -1]
w, b = train_nb(X, Y)
predictions = inference_nb(X, w, b)
accuracy = (predictions == Y).mean()
print("Training accuracy:", accuracy * 100)


# This part detects the most relevant words for the classifier.

f = open("vocabulary.txt")
voc = f.read().split()
f.close()

indices = w.argsort()
print("NEGATIVE WORDS")
for i in indices[:20]:
    print(voc[i], w[i])

print()
print("POSITIVE WORDS")
for i in indices[-20:]:
    print(voc[i], w[i])
```

```
Training accuracy: 82.04
NEGATIVE WORDS
waste -2.642892054738665
worst -2.2508189906161267
awful -2.1284526670869433
poorly -1.9713692338276578
lame -1.7884801401285726
horrible -1.7847832782472448
mess -1.7612527808370526
crap -1.7582437230126322
terrible -1.6969670722403292
worse -1.6465217112473214
badly -1.6171678688429871
stupid -1.6171678688429854
boring -1.6170605208744444
ridiculous -1.591879612122753
dull -1.419592069277022
japanese -1.379318170139081
dumb -1.3410969573188822
bad -1.3374710602035806
cheap -1.2943476101843032
supposed -1.2579081380469024

POSITIVE WORDS
beautiful 1.0433018494420505
unique 1.1015151542612838
mary 1.1055884796489188
today 1.1055884796489197
incredible 1.1653077143505417
sweet 1.2003668859762087
loved 1.2715736171231784
favorite 1.2805618296157864
brilliant 1.3070292945176343
beauty 1.3210523442888995
perfectly 1.334774345419092
powerful 1.385779880371543
perfect 1.390060276470452
excellent 1.500445063930269
superb 1.5300294892160897
amazing 1.5440462709114327
wonderful 1.5793728317345597
disney 1.616414103414911
fantastic 1.6427314117322842
ben 1.7987356602088642
```

In [17]:

```
# The script loads the test data and train a classifier.

data = np.loadtxt("test.txt.gz")
X = data[:, :-1]
Y = data[:, -1]
w, b = train_nb(X, Y)
predictions = inference_nb(X, w, b)
accuracy = (predictions == Y).mean()
print("Test accuracy:", accuracy * 100)


# This part detects the most relevant words for the classifier.
f = open("vocabulary.txt")
voc = f.read().split()
f.close()

indices = w.argsort()
print("NEGATIVE WORDS")
for i in indices[:20]:
    print(voc[i], w[i])

print()
print("POSITIVE WORDS")
for i in indices[-20:]:
    print(voc[i], w[i])
```

```
Test accuracy: 82.91199999999999
NEGATIVE WORDS
waste -2.8018297115190283
worst -2.499869527202959
awful -2.3875743348299867
lame -2.0863673814266432
poorly -2.076511239184782
terrible -1.9588798910523924
crap -1.8502245755687943
fails -1.7043529398105175
horrible -1.6970428038083423
mess -1.641895408824415
stupid -1.5783595768005219
dull -1.5718448957793267
worse -1.5696453662201355
zombie -1.5601165792102467
badly -1.4808573704100585
annoying -1.4718294343664917
ridiculous -1.4576473784855395
boring -1.3976428178508735
bad -1.3763205379296002
cheap -1.3583924203901985

POSITIVE WORDS
scott 1.1178561666546862
favorite 1.1555076728583744
greatest 1.1865993190742916
perfectly 1.2206159006124544
today 1.2417014969866145
loved 1.2466890384976548
unique 1.2539883409792658
michael 1.272006846481947
american 1.272006846481947
brilliant 1.2922095537994647
perfect 1.322017267056606
amazing 1.3336901549201832
incredible 1.338096958310933
powerful 1.417562432073269
beauty 1.4435978009308137
excellent 1.5625419879161333
wonderful 1.5651683846983513
tom 1.6286817904206785
fantastic 1.7186300270836172
superb 1.8070271988349917
```

In [18]:

```
# The script loads the training data and train a classifier.
#It must be extended to evaluate the classifier on the test set.
data = np.loadtxt("validation.txt.gz")
X = data[:, :-1]
Y = data[:, -1]
w, b = train_nb(X, Y)
predictions = inference_nb(X, w, b)
accuracy = (predictions == Y).mean()
print("Validation accuracy:", accuracy * 100)


# This part detects the most relevant words for the classifier.
f = open("vocabulary.txt")
voc = f.read().split()
f.close()

indices = w.argsort()
print("NEGATIVE WORDS")
for i in indices[:20]:
    print(voc[i], w[i])

print()
print("POSITIVE WORDS")
for i in indices[-20:]:
    print(voc[i], w[i])
```

```
Validation accuracy: 82.504
NEGATIVE WORDS
waste -2.648414044415995
poorly -2.4477677151287214
worst -2.437259737530307
awful -2.3926158950536403
terrible -2.18497800155708
lame -1.9122494787723587
george -1.7580987989451007
horrible -1.7503166585030465
stupid -1.6596587261318474
mess -1.6591588510901971
worse -1.6170202006851966
boring -1.5881997621497028
crap -1.5615589931488651
dull -1.490417393199988
dumb -1.4665779501861538
badly -1.4438030779752307
cheap -1.4408843789283008
annoying -1.4398901358341627
fails -1.4341267250881184
supposed -1.3488095943985332

POSITIVE WORDS
greatest 1.1007742918916916
today 1.1023512790860153
loved 1.130344318044659
james 1.1322729589510647
richard 1.1322729589510647
robert 1.1322729589510647
masterpiece 1.1405717617657594
perfectly 1.1579153895644012
perfect 1.1750602341562733
brilliant 1.2024347376429976
unique 1.2041855405413502
favorite 1.2123156666246002
incredible 1.2295362643762715
fantastic 1.3583971384034115
amazing 1.4001958814223832
excellent 1.4849398814922594
peter 1.6430985827170552
wonderful 1.6724166434736665
superb 1.957347682553559
oscar 2.4315559430813245
```