Movie reviews
Machine Learning assignment
Marija Maneva

Sentimental analysis of Movie Reviews

Abstract
Sentimental analysis is a useful technique to analyze feedback, reviews, social media posts, articles, opinions and preferences.
In this lab activity, I applied sentiment analysis to movie reviews written by users on the IMDB website using a multinomial Naive Bayes classifier and SVM classifier. The aim is to detect positive and negative comments in the reviews.
To achieve this, I performed three main tasks: building a vocabulary, extracting the features and training a classifier.

1.1 Build a vocabulary
First, I built a vocabulary by analyzing text files in the "smalltrain" directory.
The code defines three functions :
1. "**remove_puntuation(text)**" : this function takes a string of text as input, removes all punctuation and returns the modified text.
2. "**read_document(filename)**": this function takes a filename as input, reads the content of the file, converts all text to lowercase, removes punctuation using the "remove_punctuation" function and returns a list of words.
3. "**write_vocabulary(voc,filename,n)**": this function takes a vocabulary (a Counter object), a filename and a number "n" as input. It opens the specified file in writing mode, and prints the most common "n" words (long with their counts) in the vocabulary to the file.

In other words, the code defines a vocabulary that is updated with the words from each file using the "update" method. At the end, the most common words in the vocabulary are written to a file named "vocabulary.txt".

1.2 Extract the features
Next, I extracted the features by performing a bag-of-words (BoW) representation of a set of documents.
(Note: What is a BoW?
A bag of words is a way of representing a text document as a collection of words without considering their order or context. It treats each word as a separate feature and assigns a numerical value to each feature based on the frequency of that given word in the document.)
The code defined a function to read a file containing a list of words (the vocabulary) and map each word to a unique numerical index stored in a dictionary. Then, it used a function to remove punctuation marks and create a BoW representation of the text in the form of a numpy array.
The result of BoW representations and label vectors (1 for positive and 0 for negative) were combined into a numpy array and saved in a file called "train.txt.gz". The same operation was done for the test set data.

1.3 Train a classifier

In the script name "train_data" we trained a Naive Bayes classifier using the extracted features and class labels.
In the code, there are two functions that compute the probability of the positive and negative words and use them to compute the w and b values of the NB classifier.
Another function makes predictions on the training dataset and prints the accuracy of the classifier.

1.4 Variants

1. Make it so the model ignores very common words

```
# function that removes the stopwords
def remove_common(words):
    f = open("stopwords.txt", encoding ="utf-8")
    badword = f.read()
    f.close()
    j=0
    for i in range(len(words)-j):
        if words[i-j] in badword:
            words.remove(words[i-j])
            j += 1
    return words



# function that opens the file with encoding utf-8 and returns a list of words
def read_document(filename):
    f = open(filename, encoding = "utf-8")
    text = f.read()
    f.close()
    words = []
# to put all the words in lower case
    text = text.lower()
    text = remove_puntuation(text)
    for word in text.split():
        if len(word)>2:
            words.append(word)
    words = remove_common(words)
    return words
```

In this part of code of the script variants.txt I am defining two functions : "remove_common" and "read_document".

- "**remove_common**": the function takes a list of words as input, open a file named "stopwords.txt" and reads its content. It then compares the words found in the input list with the words found in the stopwords list and removes the words of the stop list found in the list of input words. At the end it just returns the modified list of words.
- "**read_document**": the function takes a filename as input, opens the file, reads the content and stores it in a variable named "text". Then it does other modifications like : removing the punctuation and removing common words.

Overall, this part of the code removes the stop words from a text document leaving only important words for analysis.

2. Make it so the model does not distinguish among words with the same root (use the technique called stemming).

```
# function that applies the Porter stemming algorithm to a word
def porter_stem(word):
    suffixes = {
        'sses': 'ss',
        'ies': 'i',
        'ss':'ss',
        's': ''
    }
    for suffix in suffixes:
        if word.endswith(suffix):
            return word[:-len(suffix)] + suffixes[suffix]
    return word
```

The "porter_stem" function implements the Porter stemming algorithm. It uses a dictionary to map common suffixes to their replacement. In the "read_document" function the porter_stem function is applied to every word before adding it to the list.

With modifications 1 and 2 , the accuracy of the model goes from 80.352% to 81.136%, so thanks to the removal of the stopwords and to the application of the stemming algorithm, the accuracy increases.

1.5 Analysis

- Identify the set of most impactful words on the predictions made by the model.

```
# detection of the most relevant words of the classifier
f = open("vocabulary.txt")
words = f.read().split()
f.close()

indices = np.argsort(w)
print('NEGATIVE')
for i in indices[:20]:
    print(words[i], w[i])

print()
print('POSITIVE')
for i in indices[-20:]:
    print(words[i],w[i])
```

The set of the most impactful words on the predictions mad by the model can be identified by the weight values of the words ("w" array). The words with bigger positive w can be more representative of positive reviews and the words with bigger negative w can be considered more indicative of negative reviews.

With the code above we have identified the 20 most relevant words of the classifier (negative and positive ones).
Output : the most impactful words with their correspondent weight

NEGATIVE	POSITIVE
waste -2.719539646001727	heart 0.8817073957479646
worst -2.326787460101217	oscar 0.8897395674452273
poorly -2.1001794709839734	memorable 0.8988751993703286
bad. -1.9423135001240874	episodes 0.9205071448874502
awful -1.9351972102663275	great 0.9270036706631402
worse -1.8283080860392005	greatest 0.9667006085813572
bad, -1.6714075040665701	beautiful 1.01238571939008
badly -1.6606101612533752	season 1.029627525824587
boring -1.6278918666304083	highly 1.0396387984151287
stupid -1.6133516255040536	unique 1.1673713040435079
horrible -1.5583738226599717	brilliant 1.2081039803804368
terrible -1.550785511355997	perfect 1.2207935720505372
crap -1.461548736977008	favorite 1.254382681033138
dull -1.4570167438619546	perfectly 1.2972228397096313
ridiculous -1.4532065843614408	loved 1.353746994478799
fails -1.3430144310454786	powerful 1.4244263832899087
poor -1.2789595858544587	amazing 1.43206385827059
supposed -1.2743521927240442	excellent 1.5237390755024798
avoid -1.2617917822320912	fantastic 1.5538011671100769
bad -1.2593256897368983	wonderful 1.5573409938152007

- Also identify the worst errors on the test set, that is, those that the model classifies with the highest confidence.

To identify the worst errors on the test set, we need to evaluate the model on the test set and look at the instances where the model made an incorrect prediction (misclassification) with high confidence. This can be done by sorting the test set predictions based on their confidence scores and selecting the instances with the highest scores that were misclassified.
Output : print of the instance index, predicted label , true label and a confidence score for each of the 20 worst errors. I put a condition so that only the misclassified instances will be printed.

```
# compute test set predictions and confidence scores
test_scores = test_X @ w + b
test_preds = inference_nb(test_X, w, b)
test_confidence = np.abs(test_scores)

# find misclassified instances with highest confidence scores
worst_errors = np.argsort(test_confidence[test_preds != test_Y])[-20:]

# print worst errors
for i in worst_errors:
    if test_preds[i] != test_Y[i]:
        print("Instance:", i)
        print("Prediction:", test_preds[i])
        print("Actual Label:", test_Y[i])
        print("Confidence:", test_confidence[i])
        print("Words:", [words[j] for j in range(len(test_X[i])) if test_X[i,j] != 0])
        print()
```

2.1 Vocabulary size

- Repeat the main steps by changing the size of the vocabulary. Does it influence the results? How much?

By changing the size of the vocabulary to:
- Less words (100) : the accuracy of the model decreases to 69.016%
- More words (10000) : the accuracy of the model increases to 87.208%

2.1 Classifier

SVM classifier

Classifier trained with :
- lambda = 0
- learning rate = 0.035
- steps = 10000

	SVM classifier with variants	SVM classifier without variants	NB classifier with variants	NB classifier without variants
train accuracy	87.68%	88.6%	82.94%	82.13%
test accuracy	84%	84.79%	82.12%	81.85%

I obtained bigger accuracy values with the SVM classifier (with the specifications mentioned).

"I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher."