Machine Learning Challenge

Overview

The focus of this exercise is on a field within machine learning called Natural Language Processing. We can think of this field as the intersection between language, and machine learning. Tasks in this field include automatic translation (Google translate), intelligent personal assistants (Siri), information extraction, and speech recognition for example.

NLP uses many of the same techniques as traditional data science, but also features a number of specialised skills and approaches. There is no expectation that you have any experience with NLP, however, to complete the challenge it will be useful to have the following skills:

- understanding of the python programming language
- understanding of basic machine learning concepts, i.e. supervised learning

Instructions

1. Download this notebook!
2. Answer each of the provided questions, including your source code as cells in this notebook.
3. Share the results with us, e.g. a Github repo.

Task description

You will be performing a task known as sentiment analysis. Here, the goal is to predict sentiment -- the emotional intent behind a statement -- from text. For example, the sentence: "*This movie was terrible!*" has a negative sentiment, whereas "*loved this cinematic masterpiece*" has a positive sentiment.

To simplify the task, we consider sentiment binary: labels of `1` indicate a sentence has a positive sentiment, and labels of `0` indicate that the sentence has a negative sentiment.

Dataset

The dataset is split across three files, representing three different sources -- Amazon, Yelp and IMDB. Your task is to build a sentiment analysis model using both the Yelp and IMDB data as your training-set, and test the performance of your model on the Amazon data.

Each file can be found in the `input` directory, and contains 1000 rows of data. Each row contains a sentence, a `tab` character and then a label -- `0` or `1`.

Notes

- Feel free to use existing machine learning libraries as components in you solution!
- Suggested libraries: `sklearn` (for machine learning), `pandas` (for loading/processing data), `spacy` (for text processing).
- As mentioned, you are not expected to have previous experience with this exact task. You are free to refer to external tutorials/resources to assist you. However, you will be asked to justfify the choices you have made -- so make you understand the approach you have taken.

In [1]:
```
import os
print(os.listdir("./input"))
```

```
['amazon_cells_labelled.txt', 'yelp_labelled.txt', 'imdb_labelled.txt']
```

In [2]:
```
!head "./input/amazon_cells_labelled.txt"
```

```
So there is no way for me to plug it in here in the US unless I go by a converter.	0
Good case, Excellent value.	1
Great for the jawbone.	1
Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!	0
The mic is great.	1
I have to jiggle the plug to get it to line up right to get decent volume.	0
If you have several dozen or several hundred contacts, then imagine the fun of sending eac
h of them one by one.	0
If you are Razr owner...you must have this!	1
Needless to say, I wasted my money.	0
What a waste of money and time!.	0
```

Tasks

1. Read and concatenate data into test and train sets.

2. Prepare the data for input into your model.

install libraries

```
!pip install spacy
!python -m spacy download en_core_web_sm
!pip install nltk
!pip install sklearn
```

In [3]:
```
import spacy
from spacy.tokens.token import Token
from nltk import ngrams
from dataclasses import dataclass
from typing import List
```

In [4]:
```
nlp = spacy.load("en_core_web_sm")
```

In [5]:
```
@dataclass
class Sentence:
    sentence: str
    label: int
    tokens: List[Token]
```

In [6]:
```
def parse_line(line: str):
    sentence, label = line.strip().split('\t')
    label = int(label)
    tokens = nlp(sentence)
    return Sentence(sentence, label, tokens)


def load_sentiment_data(*files: str):
    data = (
        parse_line(line)
        for f in files
```

```
        for line in open(f, 'r')
    )
    return data
```

Load train set and test set, for only 1000 records in each file, just load to memory

In [7]:

```
train_set = list(load_sentiment_data('input/imdb_labelled.txt', 'input/yelp_labelled.txt')
test_set = list(load_sentiment_data('input/amazon_cells_labelled.txt'))
```

To get all tag of spacy

for label in nlp.get_pipe("tagger").labels: print(label, " -- ", spacy.explain(label))

To get all dep of spacy

for label in nlp.get_pipe("parser").labels: print(label, " -- ", spacy.explain(label))

2a: Find the ten most frequent words in the training set.

Wordcount by tag and dep, and then show top 20 for each of the dep/tag

In [8]:

```
tag_wordcount = {}
dep_wordcount = {}

for row in train_set:
    for token in row.tokens:
        dep = token.dep_
        tag = token.tag_
        lemma = token.lemma_
        tag_wordcount.setdefault(tag, {})[lemma] = tag_wordcount.get(tag, {}).get(lemma, (
        dep_wordcount.setdefault(dep, {})[lemma] = tag_wordcount.get(dep, {}).get(lemma, (
```

Print wordcount for each of the tag and dep

```
print('Top frequent words for each tag_:')
for tag, counts in tag_wordcount.items():
    print(tag, '>>', sorted(counts.items(), key=lambda x: -x[1])[:40])

print('Top frequent words for each dep_:')
for dep, counts in dep_wordcount.items():
    print(dep, '>>', sorted(counts.items(), key=lambda x: -x[1])[:40])
```

In [9]:

```
# reiew the wordcount for each of the tegs, and got those tags and white list

# better to review the partial dep as well when there is spare time

all_tags = {
    'RB', 'JJ', 'RBR', 'JJS', 'WRB', 'RBS', 'JJR', 'UH'
}
partial_tags = {
    'VBD': {'love', 'find', 'enjoy', 'think', 'like', 'feel', 'see', 'say', 'suck', 'wait'
    'NN': {'everything', 'nothing', 'anyone', 'quality', 'lot', 'bit', 'part'},
    'VBG': {'wait', 'consider', 'feel', 'check', 'make', 'end', 'think', 'waste', 'lose',
    'DT': {'all', 'no', 'some', 'any', 'every', 'another', 'those', 'these', 'both', 'each
    'VBN': {'recommend', 'waste', 'disappoint', 'leave', 'involve', 'lose', 'rate', 'tell'
    'CC': {'but', 'or', 'both', 'plus', 'yet', 'so', 'either', '+', 'nor'},
    'IN': {'without', 'throughout', 'although', 'whether', 'under', 'until', 'towards', 'u
    'VB': {'think', 'recommend', 'avoid', 'love', 'like', 'waste', 'feel', 'find', 'wait',
    'VBZ': {'make', 'suck', 'lack', 'give', 'fail', 'deserve', 'seem', 'rank', 'leave', 'l
    'WDT': {'whatever', 'what'},
    'VBP': {'think', 'love', 'want', 'like', 'recommend', 'guess', 'make', 'look', 'give',
```

```
}
all_deps = {'advmod', 'amod', 'acomp', 'intj', 'preconj', 'predet', 'oprd'}
```

stop words (the, a, this, it, etc) are eliminated as we have a whitelist

3. Train your model and justify your choices.

3a: Generate features, use single word and n-gram in the above `all_tags` / `partial_tags` list

In [10]:
```
n_gram = 5
```

In [11]:
```
def as_feature(tokens: List[Token]):
    feature = []
    for tok in tokens:
        if tok.tag_ in all_tags or tok.tag_ in partial_tags and tok.lemma in partial_tags
            feature.append(tok.text.lower())
    if feature:
        return ';;'.join(sorted(feature))
    return None


def generate_features(tokens: List[Token]):
    tokens = list(filter(lambda x: x.dep_ != 'punct', tokens))
    features = []
    for i in range(1, n_gram):
        for words in ngrams(tokens, i):
            feature = as_feature(words)
            if feature is not None:
                features.append(feature)
    return features
```

In [12]:
```
feature_vector = {}
for row in train_set:
    for feature in generate_features(row.tokens):
        feature_vector[feature] = feature_vector.get(feature, 0) + 1
```

a view of the features

In [13]:
```
feature_vectore_desc = sorted(feature_vector.items(), key=lambda x: -x[1])
```

In [14]:
```
print('Top 20:', feature_vectore_desc[:20], '\nMid:', feature_vectore_desc[2200:2240], '\r
```

```
Top 20: [("n't", 1415), ('not', 1043), ('good', 749), ('great', 671), ('just', 566), ('s
o', 447), ('bad', 426), ('ever', 400), ('very', 399), ('here', 389), ('really', 378), ('on
ly', 330), ('when', 325), ('back', 316), ('best', 315), ('even', 293), ('all', 290), ('oth
er', 289), ('never', 287), ('more', 275)]
Mid: [('special;;whatsoever', 3), ('probably;;worst', 3), ('fresh;;succulent', 3), ('equal
ly;;special', 3), ('enthusiastic;;real', 3), ('all;;small', 3), ('elegantly;;tiny', 3),
('eggplant;;usual', 3), ('at;;mediocre', 3), ('not;;now', 3), ('as;;friendly', 3), ('eve
n;;hi', 3), ('incredible;;nay', 3), ('nay;;transcendant', 3), ('incredible;;nay;;transcend
ant', 3), ('full;;petty', 3), ('happy;;hungry', 3), ('sore;;still', 3), ('friendly;;profes
sional', 3), ('furthermore', 3), ('delicious;;so', 3), ('just;;spicy', 3), ('gooodd;;so',
3), ('insulted;;so', 3), ('enough;;fresh', 3), ('creamy;;smooth', 3), ('quite;;really',
3), ('nice;;quite;;really', 3), ('first;;only', 3), ('just;;rather', 3), ('just;;much;;rat
her', 3), ("n't;;small", 3), ('good;;rarely', 3), ('also;;really', 3), ('also;;good;;reall
y', 3), ('-;;multi', 3), ('-;;grain', 3), ('-;;grain;;multi', 3), ('friendly;;so', 3), ('i
mpressed;;very', 3)]
```

```
Last 20: [('not;;really;;sweet', 1), ('not;;really;;spicy', 1), ('not;;really;;sweet;;to
o', 1), ('not;;really;;spicy;;sweet', 1), ('enough;;not;;really;;spicy', 1), ('horrible;;o
verpriced', 1), ('just;;maybe', 1), ('at;;busy', 1), ("all;;at;;busy;;n't", 1), ('all;;a
t;;now', 1), ('also;;dirty;;outside', 1), ('always;;friendly;;helpful', 1), ('douchey;;mor
e', 1), ('back;;then', 1), ('not;;rude', 1), ('not;;rude;;very', 1), ('even;;not;;rude',
1), ("fresh;;n't;;obviously", 1), ('not;;overall', 1), ('underwhelming;;whole', 1)]
```

In [15]:

```
print(f'Feature dimension with n-gram (n is [1, {n_gram-1}]):', len(feature_vector))
```

```
Feature dimension with n-gram (n is [1, 4]): 3694
```

Generate n-dimension traning vector

In [16]:

```
from sklearn.linear_model import LogisticRegression
```

In [17]:

```
num_features = len(feature_vector)
feature2index = dict(zip(feature_vector.keys(), range(num_features)))
```

In [18]:

```
def fillin_features(num_features, tokens: List[Token], feature2index):
    row_array = [0 for _ in range(num_features)]
    for feature in generate_features(tokens):
        if feature in feature2index:
            row_array[feature2index[feature]] = 1
    return row_array
```

In [19]:

```
X_train = [
    fillin_features(num_features, row.tokens, feature2index)
    for row in train_set
]
y_train = [row.label for row in train_set]
```

In [20]:

```
clf_model = LogisticRegression(solver='lbfgs').fit(X_train, y_train)
```

In [21]:

```
clf_model.score(X_train, y_train)
```

Out[21]:

```
0.915
```

4. Evaluate your model using metric(s) you see fit and justify your choices.

In [22]:

```
X_test = [
    fillin_features(num_features, row.tokens, feature2index)
    for row in test_set
]
y_test = [row.label for row in test_set]
```

In [23]:

```
clf_model.score(X_test, y_test)
```

Out[23]:

```
0.747
```

In [ ]: