Exploration and a Simple Baseline

SOLUTIONS - do not distribute!

In this part of the assignment, we'll introduce the Stanford Sentiment Treebank (https://nlp.stanford.edu/sentiment/index.html) (SST) dataset, and train a Naive Bayes model as a simple baseline. The SST, introduced by (Socher et al. 2013) (http://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf) consists of approximately 10,000 sentences from movie reviews associated with fine-grained sentiment labels on a five-point scale, and is a popular benchmark for text classification.

Outline

- **Part (a):** The Stanford Sentiment Treebank
- **Part (b):** Naive Bayes
- **Part (c):** Exploring Negation

Exercises are interspersed throughout the notebook. Be sure you catch all of them! There are 4 questions for Part (a), 2 for Part (b), and 3 for Part (c).

Part (a) questions: Exploring the Data

Let's explore the data a bit, to get a sense of what we're working with. If you're not familiar with DataFrames, you may wish to review the Pandas documentation: https://pandas.pydata.org/pandas-docs/stable/dsintro.html (https://pandas.pydata.org/pandas-docs/stable/dsintro.html)

Answer the following questions in the cells below:

1. Looking at only the root examples in the training set (*Hint: use `ds.train[ds.train.is_root]`*), what is the fraction of positive labels? Is the classification task balanced, or close to it? If we used most-common-class as a baseline classifier, what would the accuracy be?
2. What are the five most common words in the dataset? (*Hint: there are several ways to get at this - you might use `collections.Counter`, or poke around in the `ds.vocab` object.*)
3. Use the `plot_length_histogram` function (defined above) to plot a histogram of the sentence lengths. What is the 95% percentile length? (i.e. 95% of examples in the training set should be shorter than this)
4. Repeat 3., but this time including all subphrases. Notice the difference in distributions. Could this be problematic, if we include subphrases in our training data?

Answers for Part (a)

SOLUTIONS - do not distribute!

1. The fraction of positive labels = 0.52. The task is approximately balanced.
   Using this as a baseline, the accuracy would be 0.52.

2. The most common tokens are: ['.', 'the', ',', 'a', 'and'] .
   Eliminating punctuation, the 5 most common tokens are: ['the', 'a', 'and', 'of', 'to']

3. The 95%-ile length is: 36 words.

4. Including all sub-trees, the 95%-ile length is: 24 words. This could be problematic, since if our model trains on mainly shorter phrases it would conceivably focus less on accuracy of complete longer phrases.

Use the cells below for your code solutions. Each part shouldn't require more than a couple lines, but you're welcome to explore more!

No solutions for coding part (a)

We won't be releasing code solutions, as this makes it difficult to re-use the assignment in future iterations of the class. If you have specific questions on your code, please come to office hours or post a private question on Piazza.

In [30]:
```
#### YOUR CODE HERE ####
# Code for Part (a).1

#### END(YOUR CODE) ####
```

```
Index(['tokens', 'ids', 'label', 'is_root', 'root_id'], dtype='object')
```

Out[30]:
```
0.5216763005780347
```

In [33]:
```
#### YOUR CODE HERE ####
# Code for Part (a).2

#### END(YOUR CODE) ####
```

Out[33]:
```
['.', 'the', ',', 'a', 'and']
```

In [34]:
```
#### YOUR CODE HERE ####
# Code for Part (a).3

#### END(YOUR CODE) ####
```

```
95th-percentile index: 6574 with length: 36
```

In [35]:
```
#### YOUR CODE HERE ####
# Code for Part (a).4


#### END(YOUR CODE) ####
```

```
95th-percentile index: 93854 with length: 24
```

Part (b): Naive Bayes

In this section, we'll build and explore a Naive Bayes model as a baseline classifier for our dataset.

Naive Bayes is perhaps the simplest possible classification algorithm, but it's one that still surprisingly effective for many text classification problems. Recall from your Machine Learning course:

$$P(y=k) = \hat{\theta}_k = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}[y_i = k]$$

$$P(x_j|y=1) = \hat{\theta}_{k,j} = \frac{\sum_{i=1}^{N}\sum_{j'=1}^{n_i}\mathbf{1}[y_i = 1 \wedge x_{j'} = j]}{\sum_{i=1}^{N}\mathbf{1}[y_i = 1]\cdot n_i}$$

where N is the size of the dataset, and n_i is the length (number of tokens of the i^{th} example. Prediction is done by computing the score:

$$\text{score}(x) = \log\left(\frac{P(y=1)\prod_{j=1}^{n}P(x_j|y=1)}{P(y=0)\prod_{j=1}^{n}P(x_j|y=0)}\right)$$

We'll just use the implementation from scikit-learn (http://scikit-learn.org/stable/modules/naive_bayes.html). Like other scikit-learn classifiers, this expects the input as a `scipy.sparse` matrix. Run the cell below:

Part (b) Questions

Answer the following questions in the answer and code cells below.

Question 1.) Implement Naive Bayes using `sklearn.naive_bayes.MultinomialNB`. Train on the training set and evaluate accuracy on the test set using `.predict(...)`.

Your model should train almost instantly, and score 82.21% - not bad! On SST, this is actually a very strong baseline.

Recall that Naive Bayes can be interpreted as a linear model, where score is given by:

$$\text{score}(x) = \log\left(\frac{P(y=1)\prod_{j=1}^{n} P(x_j|y=1)}{P(y=0)\prod_{j=1}^{n} P(x_j|y=0)}\right) = \left(\log\hat{\theta}_1 - \log\hat{\theta}_0\right) + \sum_{j=1}^{n}\left(\log\hat{\theta}_{1,j} - \log\hat{\theta}_{0,j}\right)$$

You can access the values $\log\hat{\theta}_{k,j}$ from the trained model using `nb.feature_log_prob_[k,j]`.

Question 2.) In the cell below, compute the weights $w_j = \left(\log\hat{\theta}_{1,j} - \log\hat{\theta}_{0,j}\right)$ of the linear model, and find the top 10 most negative and most positive weights. *(Hint: use `np.argsort` to get the indices of the most extreme elements.)* Do the features you found make sense for this domain?

Answer for Part (b).2

SOLUTIONS - do not distribute!

2.) See positive and negative words below along with weights. In some cases you may get one or more different words in your list with the same linear weights, depending on how you do indexing.

No coding solutions for part (b)

We won't be releasing code solutions, as this makes it difficult to re-use the assignment in future iterations of the class. If you have specific questions on your code, please come to office hours or post a private question on Piazza.

In [37]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
#### YOUR CODE HERE ####


#### END(YOUR CODE) ####
acc = accuracy_score(test_y, y_pred)
print("Accuracy on test set: {:.02%}".format(acc))
```

```
Accuracy on test set: 82.21%
```

In [38]:
```
#### YOUR CODE HERE ####
# Code for part (b).2





#### END(YOUR CODE) ####

print("Most negative features:")
for idx in top_negative_features:
    print("  {:s} ({:.02f})".format(ds.vocab.id_to_word[idx],
                                linear_weights[idx]))
print("")
print("Most positive features:")
for idx in top_positive_features:
    print("  {:s} ({:.02f})".format(ds.vocab.id_to_word[idx],
                                linear_weights[idx]))
```

```
[-3.91008605 -3.70127696 -3.43161339 -3.3492226  -3.24671617]
[8 6 5 4 3]
['of', 'a', ',', 'the', '.']
[-3.63051769 -3.58993984 -3.31372977 -3.26689224 -3.22825445]
[7 6 4 5 3]
[1437  356  578 1091  524]
Most negative features:
  stupid (-3.17)
  suffers (-3.08)
  unfunny (-2.97)
  mess (-2.79)
  pointless (-2.79)
  flat (-2.75)
  poorly (-2.72)
  car (-2.72)
  tiresome (-2.64)
  disguise (-2.56)

Most positive features:
  heartwarming (2.41)
  portrait (2.41)
  wonderful (2.49)
  riveting (2.49)
  touching (2.56)
  refreshing (2.63)
  inventive (2.69)
  perfectly (2.81)
  solid (3.45)
  powerful (3.53)
```

Part (c): Examining Negation

While Naive Bayes performs well in the aggregate, as a linear model it's still limited in its ability to model complex phenomena in the data. Each feature - in this case, each word - contributes a weight to the total, and if the sum is ≥ 0 we predict the example is positive. But what happens when we have an example with both positive and negative words? For instance:

```
[Brando 's performance fell short of the high standards set by his earlier work .]
[A thoughtful look at a painful incident that made headlines in 1995 .]
```

Run the cell below to evaluate your model on these examples. It should predict both as positive.

Part (c).1

SOLUTIONS - do not distribute!

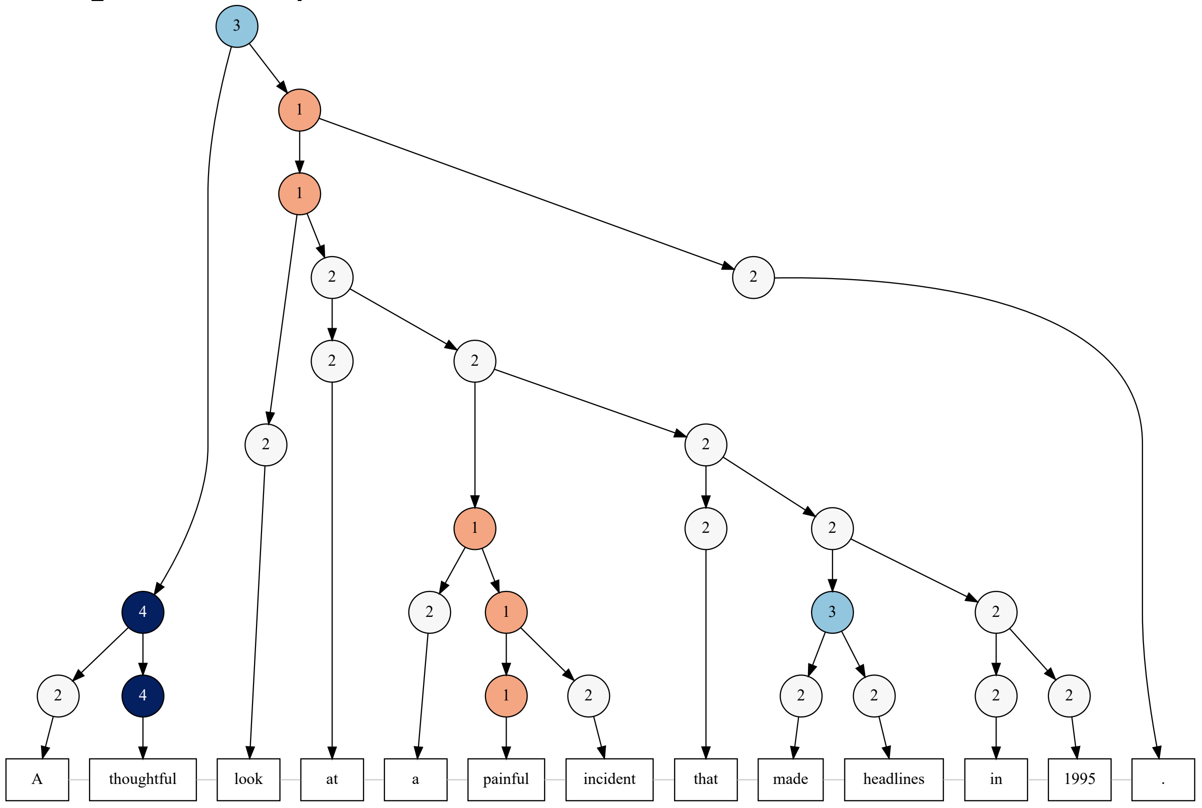
Question 1.) Why does the model get the first example wrong? Why does it get the second one correct, despite the presence of the words "painful" and "incident"?

1.) In general, the words in each example are not "strong" enough to individually determine over-all sentence sentiment. In the first case, positive-weighted words out-weigh negative-weighted words to infer a sentiment value = '1' which ends up being wrong. The same with the second case. We see that the predictions are a statistical artifact of the weights being stronger for certain words than others. Because the model doesn't make use of context it's impossible for it to use context to decide overall sentiment.

```
In [50]: ds.test_trees[210]
```

Out[50]:

A thoughtful look at a painful incident that made headlines in 1995 .



Part (c) continued

Answer the following in the cells below.

Question 2.) Examine a few of the "interesting" trees. What kinds of patterns do you see? What is the relation of the polarity of the sub-phrases to that of the whole sentence? Is this well-captured by a linear model?

Question 3.) Evaluate your model on `test_x_interesting` and compute accuracy. Does your model do better or worse on this slice of the data than on the rest of the test set? What is the *relative* change in the error rate? *(For example, if you go from 90% accuracy to 85%, that's a 50% increase in error!)*

Answers for Part (c).2 and 3

SOLUTIONS - do not distribute!

2.) In general *interesting* examples have many sub-phrases with competing sentiments; depending on how they are conjoined together with words such as 'but', 'however', 'despite', or simple negation, a positive-sentiment phrase or word can be rendered negative and vice-versa. This highlights a limitation of the linear bag-of-words naive Bayes model: the only features are individual words which are weighted strictly on their occurrence in positive or negative sentiment sentences. There is no consideration of interactions of words in sub-phrases.

3.) By only testing accuracy on 'interesting' samples, the error rate has increased from 17.8% to 26.74% which is an increase of:

$$\frac{\Delta\text{error-rate}}{\text{error-rate}_{initial}} = \frac{8.96\%}{17.8\%} \approx 50\%$$

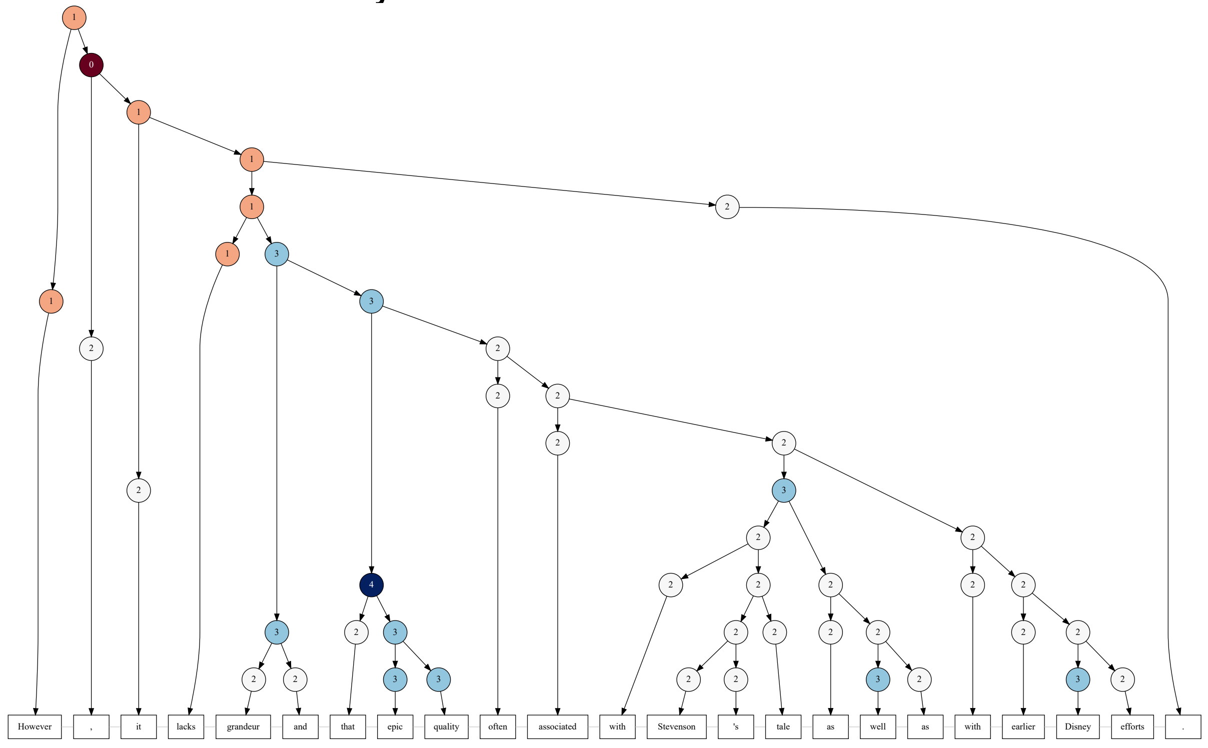
No coding solutions for part (c)

We won't be releasing code solutions, as this makes it difficult to re-use the assignment in future iterations of the class. If you have specific questions on your code, please come to office hours or post a private question on Piazza.

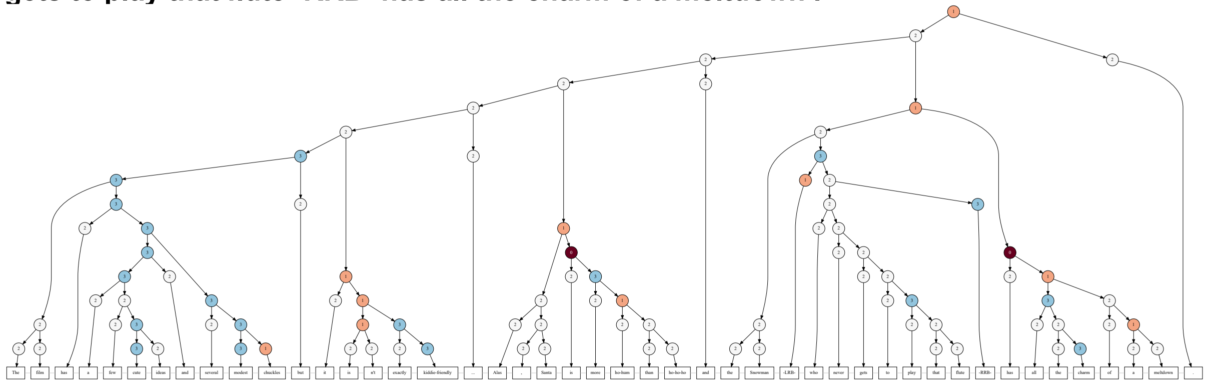
In [53]:
```
#### YOUR CODE HERE ####
# Code for part (c).2


#### END(YOUR CODE) ####
```

However , it lacks grandeur and that epic quality often associated with Stevenson 's tale as well as with earlier Disney efforts .



The film has a few cute ideas and several modest chuckles but it is n't exactly kiddie-friendly ... Alas , Santa is more ho-hum than ho-ho-ho and the Snowman -LRB- who never gets to play that flute -RRB- has all the charm of a meltdown .



In [54]:
```
#### YOUR CODE HERE ####
# Code for part (c).3



#### END(YOUR CODE) ####
print("Accuracy on selected examples: {:.02f}%".format(100*acc))
```

```
Accuracy on selected examples: 73.26%
```