Project Task 3 & 4

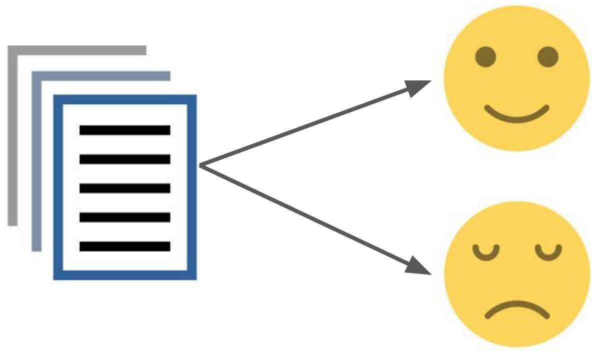
10-315: Intro to Machine Learning

Previously in Tasks 1 & 2 ...

Problem statement: given movie review, predict whether it is positive or negative.

Task - 1: test reviews were from the same dataset and train reviews

Task - 2: different set of test reviews not from the same distribution



Task 3

Task 2 continued … more information about modified set
“**original**” set: the train set in Task 1, 2
“**modified**” set: the test set in Task 2.

- “modified set”
  - Take a review from “original” set -- **change a few words so that the label flips**

	Example review	Label
Original	I have spent the last week watching John Cassavetes films - starting with 'a woman under the influence' and ending on 'opening night'. I am completely and utterly blown away, in particular by these two films. from the first minute to the last in 'opening night' i was completely and utterly absorbed. i've only experienced it on a few occasions, but the feeling that this film was perfect lasted from about two thirds in, right through till the credits came up.	Positive
Converted	I have spent the last week watching John Cassavetes films - starting with 'a woman under the influence' and ending on 'opening night'. I am completely frustrated, in particular by these two films. from the first minute to the last in 'opening night' i was completely and utterly disappointed. i've only experienced it on a few occasions, but the feeling that this film was a disaster lasted from about two thirds in, right through till the credits came up.	Negative

Task 3

- "modified set"
  - Take a review from "original" set -- **change a few words so that the label flips**

	Example review	Label
Original	I have spent the last week watching John Cassavetes films - starting with 'a woman under the influence' and ending on 'opening night'. I am completely and utterly blown away, in particular by these two films. from the first minute to the last in 'opening night' i was completely and utterly absorbed. i've only experienced it on a few occasions, but the feeling that this film was perfect lasted from about two thirds in, right through till the credits came up.	Positive
Converted	I have spent the last week watching John Cassavetes films - starting with 'a woman under the influence' and ending on 'opening night'. I am completely frustrated, in particular by these two films. from the first minute to the last in 'opening night' i was completely and utterly disappointed. i've only experienced it on a few occasions, but the feeling that this film was a disaster lasted from about two thirds in, right through till the credits came up.	Negative

  - Insert or replace words, use qualifiers, add sarcasm etc.
  - Modified review looks very similar to original, and has many words in common, but the label is completely opposite!
  - A good algorithm must pay attention to the few important words that predict sentiment. Hard Task -- "dumb" classifiers will when trained only on original reviews will perform poorly on modified ones.

Task 3

```
./
├── test
└── train
    ├── modified [size: 1707]
    │   ├── pos
    │   └── neg
    ├── original [size: 22553]
    │   ├── pos
    │   └── neg
    └── original_extra [size: 1707]
        ├── pos
        └── neg
```

Train	Test
original	test

Test Set

- Mix of “modified” reviews and corresponding “original” reviews.
- Must to well simultaneously on “modified” and “original” reviews to get accuracy above 50%.

Submit to Gradescope Leaderboard

Task 4a

```
./
├─ test
└─ train
   ├─ modified [size: 1707]
   │  ├─ pos
   │  └─ neg
   ├─ original [size: 22553]
   │  ├─ pos
   │  └─ neg
   └─ original_extra [size: 1707]
      ├─ pos
      └─ neg
```

Train	Test
original + modified	test

Task 4a

- What happens if you have a (small) set of modified reviews as well?
- Train on combined “original” + “modified” reviews.
- Report improvement in performance due to using modified reviews.

Submit to Gradescope Leaderboard

Task 4b

```
./
├── test
└── train
    ├── modified [size: 1707]
    │   ├── pos
    │   └── neg
    ├── original [size: 22553]
    │   ├── pos
    │   └── neg
    └── original_extra [size: 1707]
        ├── pos
        └── neg
```

Train	Test
original + original_extra	test

Task 4b: Ablation for Task 4a

- How much of the improvement from Task 3 to Task 4a is due to just using additional data, vs using better quality data?
- Control experiment for this by using the same amount of extra *"original"* data.
- Use `original_extra` instead of `modified` -- they both have the same number of examples.
- Leaderboard will not be evaluated.

Task 4c -- Interpreting Learned Models

Task 4c: Interpreting ML models

- Train a linear model using Bag-of-words on Task 4a, 4b
  - Logistic Regression
  - Linear Regression
  - SVM etc.

- Find 10 most important words for +ve classification
  10 most important words for -ve classification

- How? Look at the weights learned - one for each word. Pick 10 words with largest positive and largest negative weights.

No Gradescope submission -- only include results in project report

NLP Preprocessing Tips

- Can implement from scratch, or use **NLTK + SKLearn** libraries
- SKLearn CountVectorizer -- gives you counts of words
    - binary=True argument gives you Bag-Of-Words model

sklearn.feature_extraction.text.CountVectorizer

Examples

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> print(vectorizer.get_feature_names())
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
>>> print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

NLP Preprocessing Tips

- Can implement from scratch, or use **NLTK + SKLearn** libraries
- SKLearn TF-IDF
  - gives you TF-IDF scores, more meaningful measure of information content.

sklearn.feature_extraction.text.TfidfVectorizer

Examples

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = TfidfVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> print(vectorizer.get_feature_names())
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
>>> print(X.shape)
(4, 9)
```

NLP Preprocessing Tips

- NLTK stopwords
  - words that don’t contribute much meaning and can be removed

```
>>> from nltk.corpus import stopwords
>>> stopwords.words('english')
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours
', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',
 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'th
ese', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did
', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with'
, 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'do
wn', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why
', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own',
 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd',
'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'had
n', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn
', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "w
ouldn't"]
```

NLP Preprocessing Tips

- NLTK Word Tokenizer -- divides sentences into list of words.
    - Handles punctuation, spaces, tabs, newlines etc.

```
>>> from nltk.tokenize import word_tokenize
>>> s = '''Good muffins cost $3.88\nin New York.  Please buy me
... two of them.\n\nThanks.'''
>>> word_tokenize(s)
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.',
'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

NLP Preprocessing Tips

- Putting stopwords, tokenizer, and counting words/BoW together at once ...

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
```

```
word_counter = CountVectorizer('filename',
                               tokenizer=word_tokenize,
                               stop_words=set(stopwords.words('english')))
```

```
X_train = word_counter.fit_transform(files_train)
```

```
X_test = word_counter.transform(files_test)
```

Models

- Logistic Regression -- SKLearn
- BERT features -- good at generalizing

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

```
# Train logistic regression classifier.
clf = LogisticRegression()
clf.fit(X_train, labels_train)
```

```
Y_test = clf.predict(X_test)
```