Supervised sentiment analysis

Christopher Potts

Stanford Linguistics

CS 224U: Natural language understanding
April 15

Overview

1. Sentiment as a deep and important NLU problem
2. General practical tips for sentiment analysis
3. The Stanford Sentiment Treebank (SST)
4. sst.py
5. Methods: hyperparameters and classifier comparison
6. Feature representation
7. RNN classifiers
8. Tree-structured networks

Associated materials

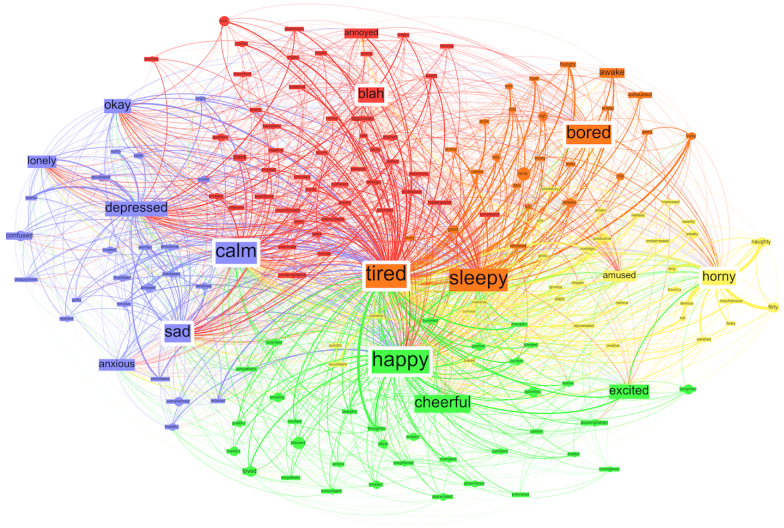
1. Code
   a. `sst.py`
   b. `sst_01_overview.ipynb`
   c. `sst_02_hand_build_features.ipynb`
   d. `sst_03_neural_networks.ipynb`
2. Core reading: Socher et al. 2013
3. Auxiliary readings: Pang and Lee 2008; Goldberg 2015

Conceptual challenges

Which of the following sentences express sentiment? What is their sentiment polarity (pos/neg), if any?

1. There was an earthquake in California.
2. The team failed to complete the physical challenge. (We win/lose!)
3. They said it would be great.
4. They said it would be great, and they were right.
5. They said it would be great, and they were wrong.
6. The party fat-cats are sipping their expensive imported wines.
7. Oh, you're terrible!
8. Here's to ya, ya bastard!
9. Of 2001, "Many consider the masterpiece bewildering, boring, slow-moving or annoying, . . . "
10. long-suffering fans, bittersweet memories, hilariously embarrassing moments, . . .

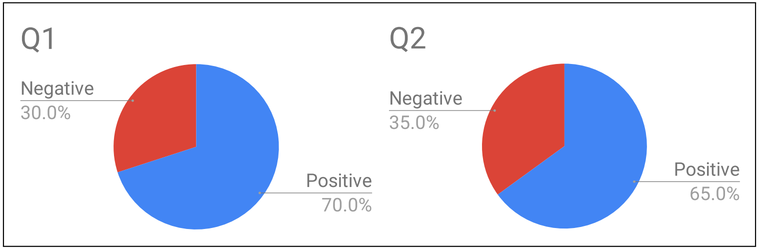
Affective dimensions, relations, and transitions



(Sudhof et al. 2014)

Lots of applications, but what’s the real goal?

Many business leaders think they want this:



When they see it, they realize that it does not help them with decision-making. The distributions (assuming they are accurately measured) are hiding the phenomena that are actually relevant.

Related tasks in affective computing

With selected papers that make excellent entry points because of their positioning and/or associated public data:

• Subjectivity	(Pang and Lee 2008)
• Bias	(Recasens et al. 2013; Pryzant et al. 2020)
• Stance	(Anand et al. 2011)
• Hate-speech	(Nobata et al. 2016)
• Microaggressions	(Breitfeller et al. 2019)
• Condescension	(Wang and Potts 2019)
• Sarcasm	(Khodak et al. 2017)
• Deception and betrayal	(Niculae et al. 2015)
• Online trolls	(Cheng et al. 2017)
• Polarization	(Gentzkow et al. 2019)
• Politeness	(Danescu-Niculescu-Mizil et al. 2013)
• Linguistic alignment	(Doyle et al. 2016)

General practical tips

Selected sentiment datasets

There are too many to try to list, so I picked some with noteworthy properties, limiting to the core task of sentiment analysis:

- IMDb movie reviews (50K) (Maas et al. 2011): `http://ai.stanford.edu/~amaas/data/sentiment/index.html`
- Datasets from Lillian Lee's group: `http://www.cs.cornell.edu/home/llee/data/`
- Datasets from Bing Liu's group: `https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html`
- RateBeer (McAuley et al. 2012; McAuley and Leskovec 2013): `http://snap.stanford.edu/data/web-RateBeer.html`
- Amazon Customer Review data: `https://s3.amazonaws.com/amazon-reviews-pds/readme.html`
- Amazon Product Data (McAuley et al. 2015; He and McAuley 2016): `http://jmcauley.ucsd.edu/data/amazon/`
- Sentiment and social networks together (West et al. 2014) `http://infolab.stanford.edu/~west1/TACL2014/`
- Stanford Sentiment Treebank (SST; Socher et al. 2013) `https://nlp.stanford.edu/sentiment/`

Lexica

- Bing Liu's Opinion Lexicon: `nltk.corpus.opinion_lexicon`
- SentiWordNet: `nltk.corpus.sentiwordnet`
- MPQA subjectivity lexicon: `http://mpqa.cs.pitt.edu`
- Harvard General Inquirer
  - Download: `http://www.wjh.harvard.edu/~inquirer/spreadsheet_guide.htm`
  - Documentation: `http://www.wjh.harvard.edu/~inquirer/homecat.htm`
- Linguistic Inquiry and Word Counts (LIWC): `https://liwc.wpengine.com`
- Hamilton et al. (2016): SocialSent `https://nlp.stanford.edu/projects/socialsent/`
- Brysbaert et al. (2014): Norms of valence, arousal, and dominance for 13,915 English lemmas

Relationships between sentiment lexica

	MPQA	Opinion Lexicon	Inquirer	SentiWordNet	LIWC
MPQA	—	33/5402 (0.6%)	49/2867 (2%)	1127/4214 (27%)	12/363 (3%)
Opinion Lexicon		—	32/2411 (1%)	1004/3994 (25%)	9/403 (2%)
Inquirer			—	520/2306 (23%)	1/204 (0.5%)
SentiWordNet				—	174/694 (25%)
LIWC					—

Table: Disagreement levels for the sentiment lexicons.

- Where a lexicon had POS tags, I removed them and selected the most sentiment-rich sense available for the resulting string.
- For SentiWordNet, I counted a word as positive if its positive score was larger than its negative score; negative if its negative score was larger than its positive score; else neutral, which means that words with equal non-0 positive and negative scores are neutral.

Tokenizing

Raw text

@NLUers: can&#39;t wait for the Jun 9 #projects! YAAAAAAY!!! &gt;:-D http://stanford.edu/class/cs224u/.

Tokenizing

Isolate mark-up, and replace HTML entities.

@NLUers: can’t wait for the Jun 9 #projects! YAAAAAAY!!!
>:-D http://stanford.edu/class/cs224u/.

Tokenizing

Isolate mark-up, and replace HTML entities.

@NLUers: can’t wait for the Jun 9 #projects! YAAAAAAY!!!
>:-D http://stanford.edu/class/cs224u/.

Whitespace tokenizer

@NLUers:
can’t
wait
for
the
Jun
9
#projects
YAAAAAAY!!!
>:-D
http://stanford.edu/class/cs224u/.

Tokenizing

Isolate mark-up, and replace HTML entities.

@NLUers: can’t wait for the Jun 9 #projects! YAAAAAAY!!!
>:-D http://stanford.edu/class/cs224u/.

Treebank tokenizer

@	!
NLUers	YAAAAAAY
:	!
ca	!
n’t	!
wait	>
for	:
the	-D
Jun	http
9	:
#	//stanford.edu/class/cs224u/
projects	.

Tokenizing

Isolate mark-up, and replace HTML entities.

@NLUers: can’t wait for the Jun 9 #projects! YAAAAAAY!!! >:-D http://stanford.edu/class/cs224u/.

Elements of a sentiment-aware tokenizer

- Isolates emoticons
- Respects Twitter and other domain-specific markup
- Uses the underlying mark-up (e.g., <strong> tags)
- Captures those #$%ing masked curses!
- Preserves capitalization where it seems meaningful
- Regularizes lengthening (e.g., *YAAAAAAY*⇒*YAAAY*)
- Captures significant multiword expressions (e.g., *out of this world*)

A good start: `nltk.tokenize.casual.TweetTokenizer`

Tokenizing

Isolate mark-up, and replace HTML entities.

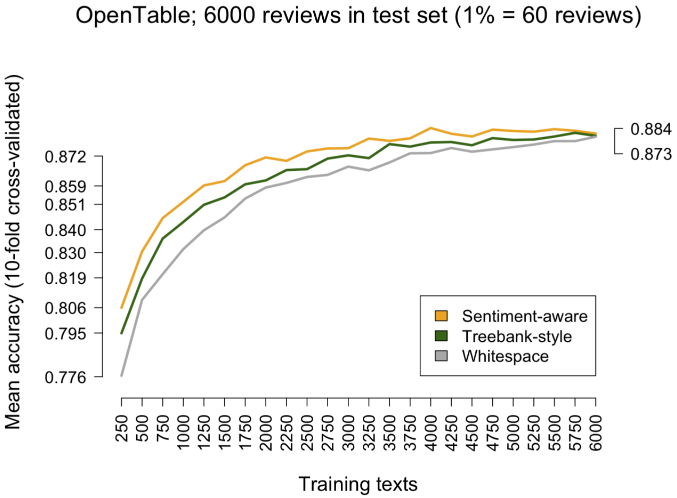
@NLUers: can’t wait for the Jun 9 #projects! YAAAAAAY!!! >:-D http://stanford.edu/class/cs224u/.

Sentiment-aware tokenizer

@nluers	!
:	YAAAY
can’t	!
wait	!
for	!
the	>:-D
Jun_9	http://stanford.edu/class/cs224u/
#projects	.

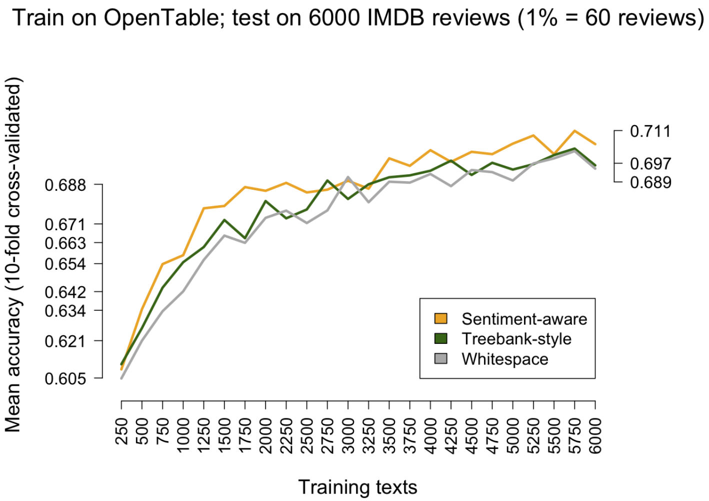
A good start: `nltk.tokenize.casual.TweetTokenizer`

The impact of sentiment-aware tokenizing



Softmax classifier. Training on 12,000 OpenTable reviews (6000 positive/4-5 stars; 6000 negative/1-2 stars).

The impact of sentiment-aware tokenizing



Softmax classifier. Training on 12,000 OpenTable reviews (6000 positive/4-5 stars; 6000 negative/1-2 stars).

The dangers of stemming

- Stemming collapses distinct word forms.
- Three common stemming algorithms in the context of sentiment:
  - the Porter stemmer
  - the Lancaster stemmer
  - the WordNet stemmer
- Porter and Lancaster destroy too many sentiment distinctions.
- The WordNet stemmer does not have this problem nearly so severely, but it generally doesn't do enough collapsing to be worth the resources necessary to run it.

The dangers of stemming

The Porter stemmer heuristically identifies word suffixes (endings) and strips them off, with some regularization of the endings.

Positiv	Negativ	Porter stemmed
defense	defensive	defens
extravagance	extravagant	extravag
affection	affectation	affect
competence	compete	compet
impetus	impetuous	impetu
objective	objection	object
temperance	temper	temper
tolerant	tolerable	toler

Table: Sample of instances in which the Porter stemmer destroys a Harvard Inquirer Positiv/Negativ distinction.

The dangers of stemming

The Lancaster stemmer uses the same strategy as the Porter stemmer.

Positiv	Negativ	Lancaster stemmed
call	callous	cal
compliment	complicate	comply
dependability	dependent	depend
famous	famished	fam
fill	filth	fil
flourish	floor	flo
notoriety	notorious	not
passionate	passe	pass
savings	savage	sav
truth	truant	tru

Table: Sample of instances in which the Lancaster stemmer destroys a Harvard Inquirer Positiv/Negativ distinction.

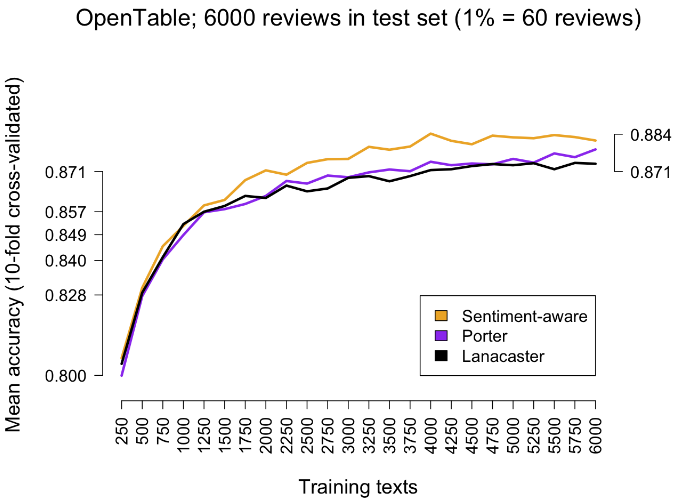
The dangers of stemming

The WordNet stemmer (NLTK) is high-precision. It requires word–POS pairs. Its only general issue for sentiment is that it removes comparative morphology.

Positiv	WordNet stemmed
(exclaims, v)	exclaim
(exclaimed, v)	exclaim
(exclaiming, v)	exclaim
(exclamation, n)	exclamation
(proved, v)	prove
(proven, v)	prove
(proven, a)	proven
(happy, a)	happy
(happier, a)	happy
(happiest, a)	happy

Table: Representative examples of what WordNet stemming does and doesn't do.

The impact of stemming



Softmax classifier. Training on 12,000 OpenTable reviews (6000 positive/4-5 stars; 6000 negative/1-2 stars).

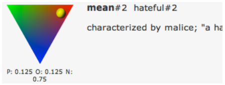
Part-of-speech (POS) tagging

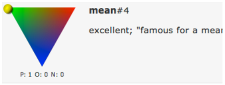
Word	Tag1	Val1	Tag2	Val2
arrest	jj	Positiv	vb	Negativ
even	jj	Positiv	vb	Negativ
even	rb	Positiv	vb	Negativ
fine	jj	Positiv	nn	Negativ
fine	jj	Positiv	vb	Negativ
fine	nn	Negativ	rb	Positiv
fine	rb	Positiv	vb	Negativ
help	jj	Positiv	vbn	Negativ
help	nn	Positiv	vbn	Negativ
help	vb	Positiv	vbn	Negativ
hit	jj	Negativ	vb	Positiv
mind	nn	Positiv	vb	Negativ
order	jj	Positiv	vb	Negativ
order	nn	Positiv	vb	Negativ
pass	nn	Negativ	vb	Positiv

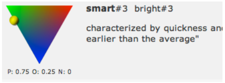
Table: Harvard Inquirer POS contrasts.

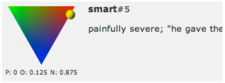
The dangers of POS tagging

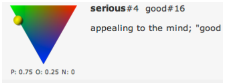
1,424 cases where a (word, tag) pair is consistent with pos. and neg. lemma-level sentiment

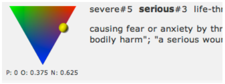

mean#2 hateful#2
characterized by malice; "a ha
P: 0.125 O: 0.125 N: 0.75

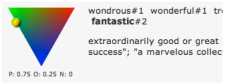

mean#4
excellent; "famous for a mear
P: 1 O: 0 N: 0

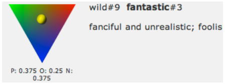

smart#3 bright#3
characterized by quickness an
earlier than the average"
P: 0.75 O: 0.25 N: 0

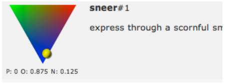

smart#5
painfully severe; "he gave the
P: 0 O: 0.125 N: 0.875

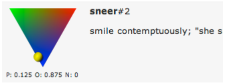

serious#4 good#16
appealing to the mind; "good
P: 0.75 O: 0.25 N: 0


severe#5 **serious**#3 life-thr
causing fear or anxiety by thr
bodily harm"; "a serious wour
P: 0 O: 0.375 N: 0.625


wondrous#1 wonderful#1 tr
fantastic#2
extraordinarily good or great
success"; "a marvelous collec
P: 0.75 O: 0.25 N: 0


wild#9 **fantastic**#3
fanciful and unrealistic; foolis
P: 0.375 O: 0.25 N: 0.375


sneer#1
express through a scornful sm
P: 0 O: 0.875 N: 0.125


sneer#2
smile contemptuously; "she s
P: 0.125 O: 0.875 N: 0

Word	Tag	ScoreDiff
mean	s	1.75
abject	s	1.625
benign	a	1.625
modest	s	1.625
positive	s	1.625
smart	s	1.625
solid	s	1.625
sweet	s	1.625
artful	a	1.5
clean	s	1.5
evil	n	1.5
firm	s	1.5
gross	s	1.5
iniquity	n	1.5
marvellous	s	1.5
marvelous	s	1.5
plain	s	1.5
rank	s	1.5
serious	s	1.5
sheer	s	1.5
sorry	s	1.5
stunning	s	1.5
wickedness	n	1.5
	[. . .]	
unexpectedly	r	0.25
velvet	s	0.25
vibration	n	0.25
weather-beaten	s	0.25
well-known	s	0.25
whine	v	0.25
wizard	n	0.25
wonderland	n	0.25
yawn	v	0.25

Simple negation marking

The phenomenon

1. I didn’t enjoy it.
2. I never enjoy it.
3. No one enjoys it.
4. I have yet to enjoy it.
5. I don’t think I will enjoy it.

Simple negation marking

The phenomenon

1. I didn’t enjoy it.
2. I never enjoy it.
3. No one enjoys it.
4. I have yet to enjoy it.
5. I don’t think I will enjoy it.

The method (Das and Chen 2001; Pang et al. 2002)

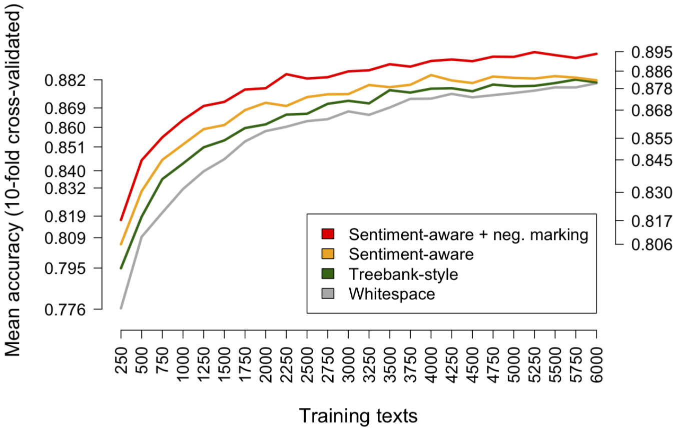
Append a _NEG suffix to every word appearing between a negation and a clause-level punctuation mark.

Simple negation marking

No one enjoys it.	no one_NEG enjoys_NEG it_NEG .
I don’t think I will enjoy it, but I might.	i don’t think_NEG i_NEG will_NEG enjoy_NEG it_NEG , but i might .

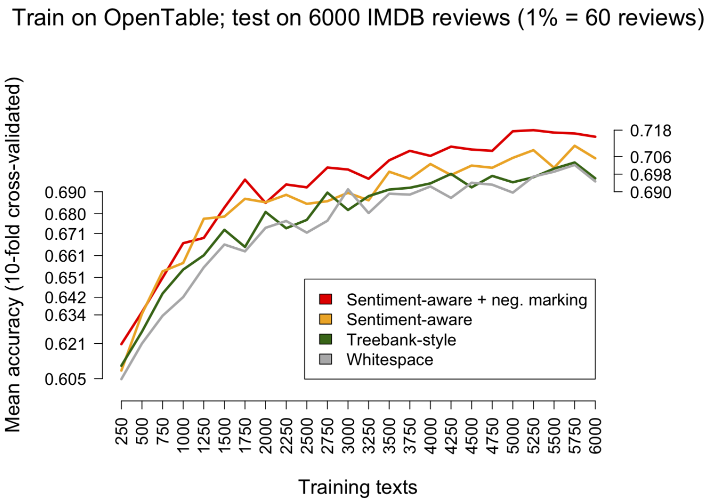
The impact of negation marking

OpenTable; 6000 reviews in test set (1% = 60 reviews)



Softmax classifier. Training on 12,000 OpenTable reviews (6000 positive/4-5 stars; 6000 negative/1-2 stars).

The impact of negation marking



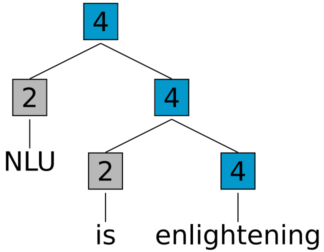
Softmax classifier. Training on 12,000 OpenTable reviews
(6000 positive/4-5 stars; 6000 negative/1-2 stars).

SST

SST project overview

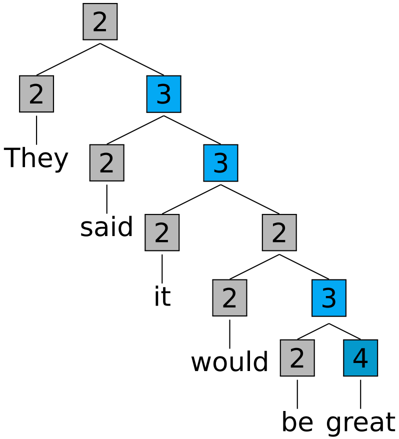
1. Socher et al. (2013)
2. Full code and data release: `https://nlp.stanford.edu/sentiment/`
3. Sentence-level corpus (10,662 sentences)
4. Original data from Rotten Tomatoes (Pang and Lee 2005)
5. Fully-labeled trees (crowdsourced labels)
6. The 5-way labels were extracted from workers’ slider responses.

Fully labeled trees



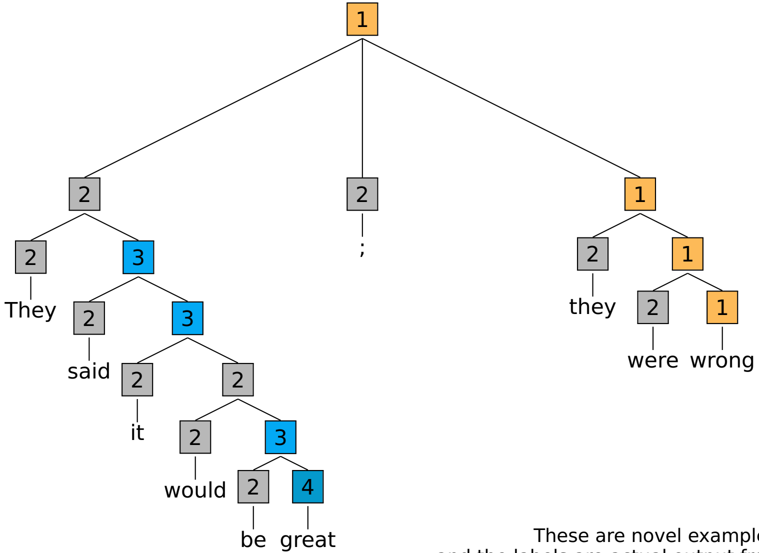
These are novel examples,
and the labels are actual output from
`https://nlp.stanford.edu/sentiment/`

Fully labeled trees



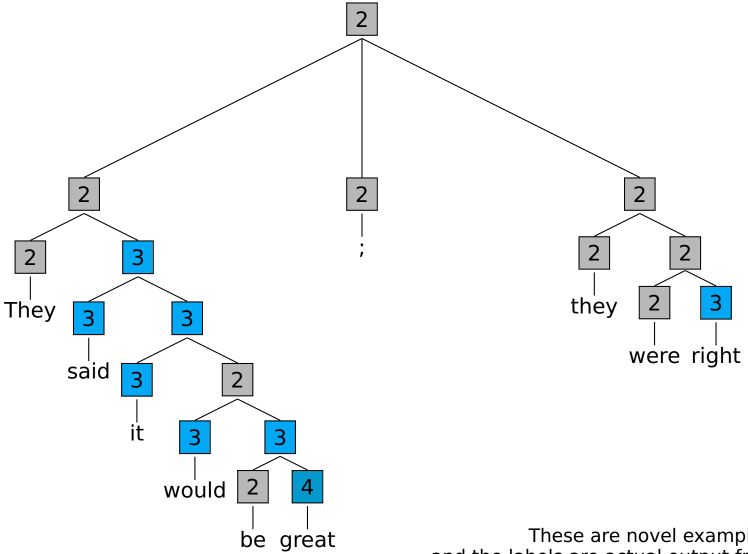
These are novel examples,
and the labels are actual output from
https://nlp.stanford.edu/sentiment/

Fully labeled trees



These are novel examples,
and the labels are actual output from
https://nlp.stanford.edu/sentiment/

Fully labeled trees



These are novel examples,
and the labels are actual output from
https://nlp.stanford.edu/sentiment/

Root-level tasks

Five-way problem

Label	Meaning	Train	Dev
0	very negative	1,092	139
1	negative	2,218	289
2	neutral	1,624	229
3	positive	2,322	279
4	very positive	1,288	165
		8,544	1,101

Note: 4 > 3 (more positive) but 0 > 1 (more negative)

Root-level tasks

Five-way problem

Label	Meaning	Train	Dev
0	very negative	1,092	139
1	negative	2,218	289
2	neutral	1,624	229
3	positive	2,322	279
4	very positive	1,288	165
		8,544	1,101

Note: 4 > 3 (more positive) but 0 > 1 (more negative)

Ternary problem

Label	Meaning	Train	Dev
0, 1	negative	3,310	428
2	neutral	1,624	229
3, 4	positive	3,610	444
		8,544	1,101

Root-level tasks

Five-way problem

Label	Meaning	Train	Dev
0	very negative	1,092	139
1	negative	2,218	289
2	neutral	1,624	229
3	positive	2,322	279
4	very positive	1,288	165
		8,544	1,101

Note: 4 > 3 (more positive) but 0 > 1 (more negative)

Binary problem (neutral data simply excluded)

Label	Meaning	Train	Dev
0, 1	negative	3,310	428
3, 4	positive	3,610	444
		6,920	872

All-nodes tasks

Five-way problem

Label	Meaning	Train	Dev
0	very negative	40,774	5,217
1	negative	82,854	10,757
2	neutral	58,398	8,227
3	positive	89,308	11,001
4	very positive	47,248	6,245
		318,582	41,447

Note: 4 > 3 (more positive) but 0 > 1 (more negative)

All-nodes tasks

Five-way problem

Label	Meaning	Train	Dev
0	very negative	40,774	5,217
1	negative	82,854	10,757
2	neutral	58,398	8,227
3	positive	89,308	11,001
4	very positive	47,248	6,245
		318,582	41,447

Note: $4 > 3$ (more positive) but $0 > 1$ (more negative)

Ternary problem

Label	Meaning	Train	Dev
0, 1	negative	123,628	15,974
2	neutral	58,398	8,227
3, 4	positive	136,556	17,246
		318,582	41,447

All-nodes tasks

Five-way problem

Label	Meaning	Train	Dev
0	very negative	40,774	5,217
1	negative	82,854	10,757
2	neutral	58,398	8,227
3	positive	89,308	11,001
4	very positive	47,248	6,245
		318,582	41,447

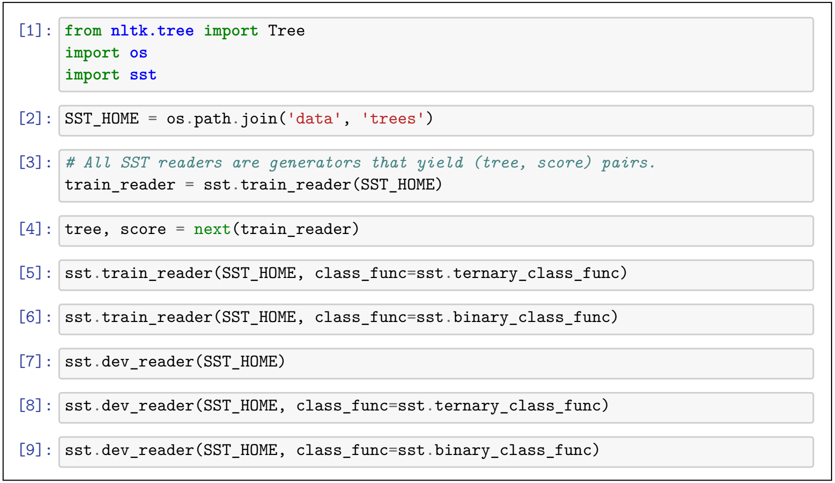
Note: 4 > 3 (more positive) but 0 > 1 (more negative)

Binary problem (neutral data simply excluded)

Label	Meaning	Train	Dev
0, 1	negative	123,628	15,974
3, 4	positive	136,556	17,246
		260,184	33,220

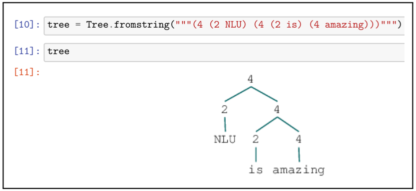
sst.py

Readers

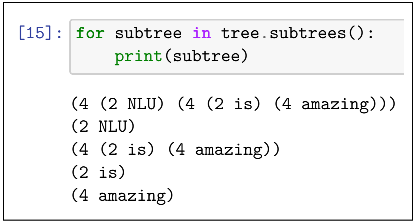


```
[1]: from nltk.tree import Tree
     import os
     import sst

[2]: SST_HOME = os.path.join('data', 'trees')

[3]: # All SST readers are generators that yield (tree, score) pairs.
     train_reader = sst.train_reader(SST_HOME)

[4]: tree, score = next(train_reader)

[5]: sst.train_reader(SST_HOME, class_func=sst.ternary_class_func)

[6]: sst.train_reader(SST_HOME, class_func=sst.binary_class_func)

[7]: sst.dev_reader(SST_HOME)

[8]: sst.dev_reader(SST_HOME, class_func=sst.ternary_class_func)

[9]: sst.dev_reader(SST_HOME, class_func=sst.binary_class_func)
```

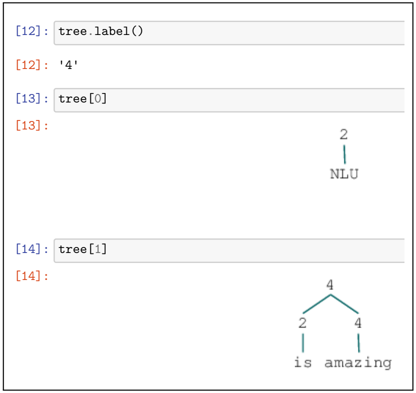
nltk.tree.Tree



```
[10]: tree = Tree.fromstring("""(4 (2 NLU) (4 (2 is) (4 amazing)))""")

[11]: tree

[11]:
```



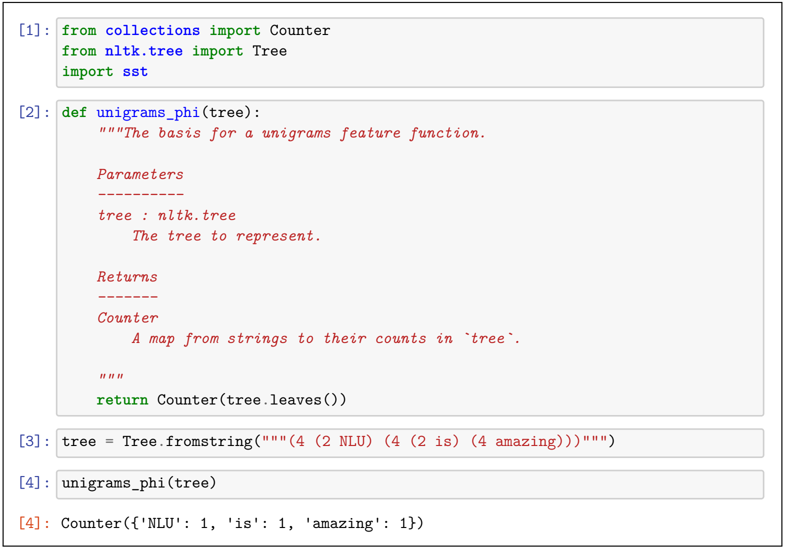
```
[15]: for subtree in tree.subtrees():
          print(subtree)

      (4 (2 NLU) (4 (2 is) (4 amazing)))
      (2 NLU)
      (4 (2 is) (4 amazing))
      (2 is)
      (4 amazing)
```



```
[12]: tree.label()

[12]: '4'

[13]: tree[0]

[13]:
```

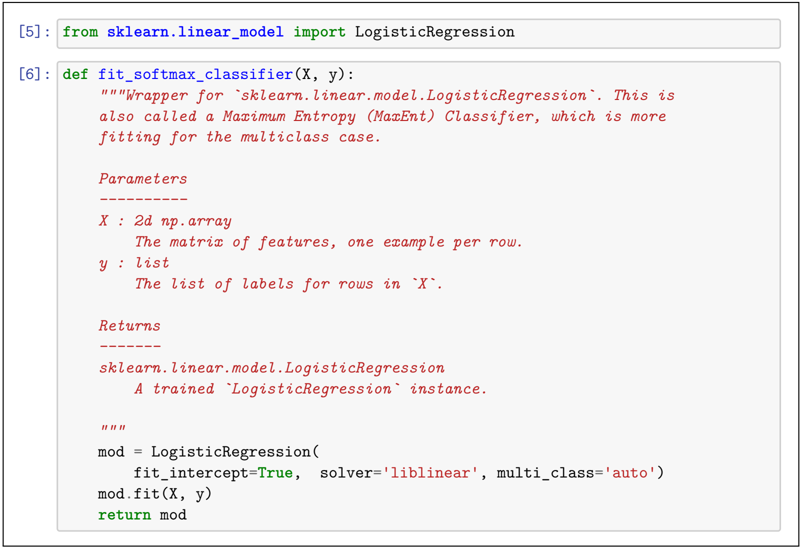
```
[14]: tree[1]

[14]:
```

Feature functions



```
[1]: from collections import Counter
     from nltk.tree import Tree
     import sst

[2]: def unigrams_phi(tree):
         """The basis for a unigrams feature function.

         Parameters
         ----------
         tree : nltk.tree
             The tree to represent.

         Returns
         -------
         Counter
             A map from strings to their counts in `tree`.

         """
         return Counter(tree.leaves())

[3]: tree = Tree.fromstring("""(4 (2 NLU) (4 (2 is) (4 amazing)))""")

[4]: unigrams_phi(tree)

[4]: Counter({'NLU': 1, 'is': 1, 'amazing': 1})
```

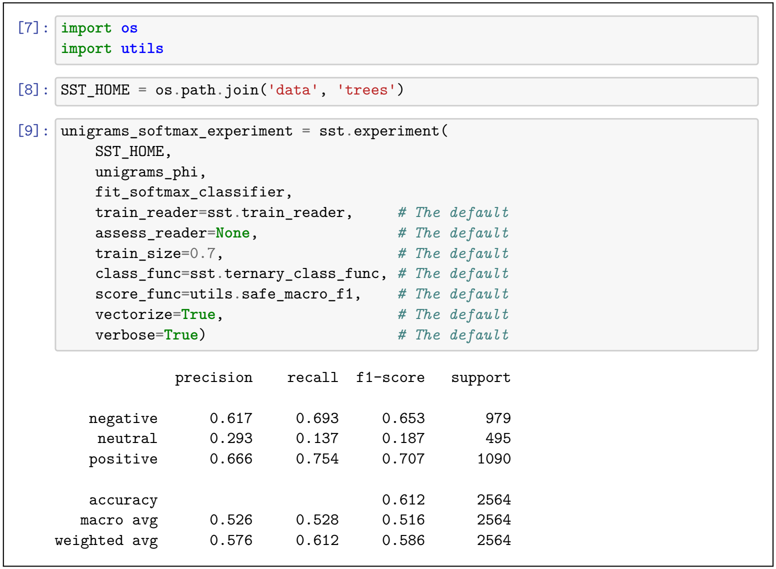
Model wrappers



```
[5]: from sklearn.linear_model import LogisticRegression
```

```
[6]: def fit_softmax_classifier(X, y):
         """Wrapper for `sklearn.linear.model.LogisticRegression`. This is
         also called a Maximum Entropy (MaxEnt) Classifier, which is more
         fitting for the multiclass case.

         Parameters
         ----------
         X : 2d np.array
             The matrix of features, one example per row.
         y : list
             The list of labels for rows in `X`.

         Returns
         -------
         sklearn.linear.model.LogisticRegression
             A trained `LogisticRegression` instance.

         """
         mod = LogisticRegression(
             fit_intercept=True,  solver='liblinear', multi_class='auto')
         mod.fit(X, y)
         return mod
```

sst.experiment



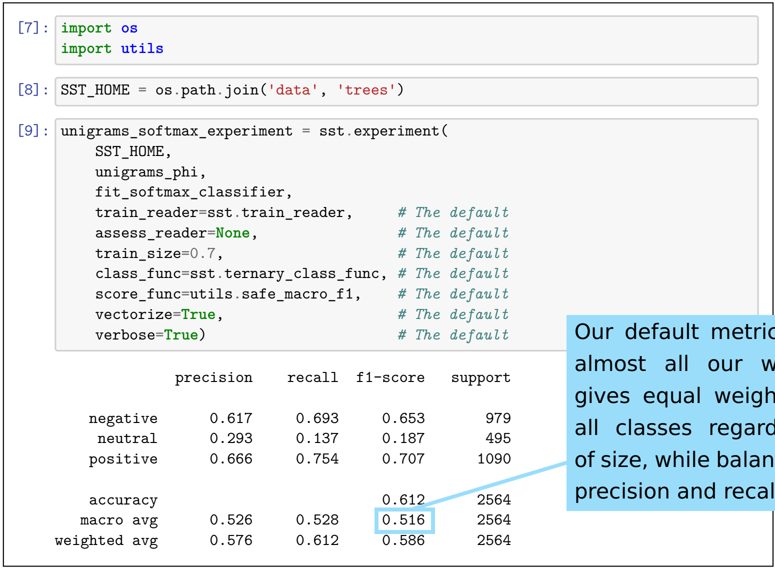
```
[7]: import os
     import utils
```

```
[8]: SST_HOME = os.path.join('data', 'trees')
```

```
[9]: unigrams_softmax_experiment = sst.experiment(
         SST_HOME,
         unigrams_phi,
         fit_softmax_classifier,
         train_reader=sst.train_reader,      # The default
         assess_reader=None,                 # The default
         train_size=0.7,                     # The default
         class_func=sst.ternary_class_func,  # The default
         score_func=utils.safe_macro_f1,     # The default
         vectorize=True,                     # The default
         verbose=True)                       # The default
```

```
              precision    recall  f1-score   support

    negative      0.617     0.693     0.653       979
     neutral      0.293     0.137     0.187       495
    positive      0.666     0.754     0.707      1090

    accuracy                          0.612      2564
   macro avg      0.526     0.528     0.516      2564
weighted avg      0.576     0.612     0.586      2564
```

sst.experiment



```
[7]: import os
     import utils
```

```
[8]: SST_HOME = os.path.join('data', 'trees')
```

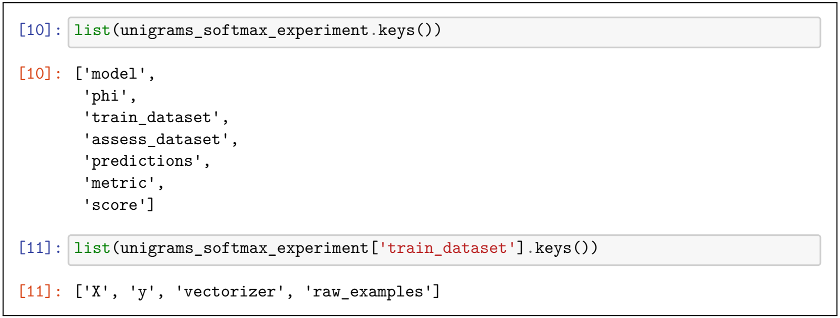
```
[9]: unigrams_softmax_experiment = sst.experiment(
         SST_HOME,
         unigrams_phi,
         fit_softmax_classifier,
         train_reader=sst.train_reader,      # The default
         assess_reader=None,                 # The default
         train_size=0.7,                     # The default
         class_func=sst.ternary_class_func, # The default
         score_func=utils.safe_macro_f1,     # The default
         vectorize=True,                     # The default
         verbose=True)                       # The default
```

```
              precision    recall  f1-score   support

    negative      0.617     0.693     0.653       979
     neutral      0.293     0.137     0.187       495
    positive      0.666     0.754     0.707      1090

    accuracy                          0.612      2564
   macro avg      0.526     0.528     0.516      2564
weighted avg      0.576     0.612     0.586      2564
```

Our default metric for almost all our work: gives equal weight to all classes regardless of size, while balancing precision and recall.

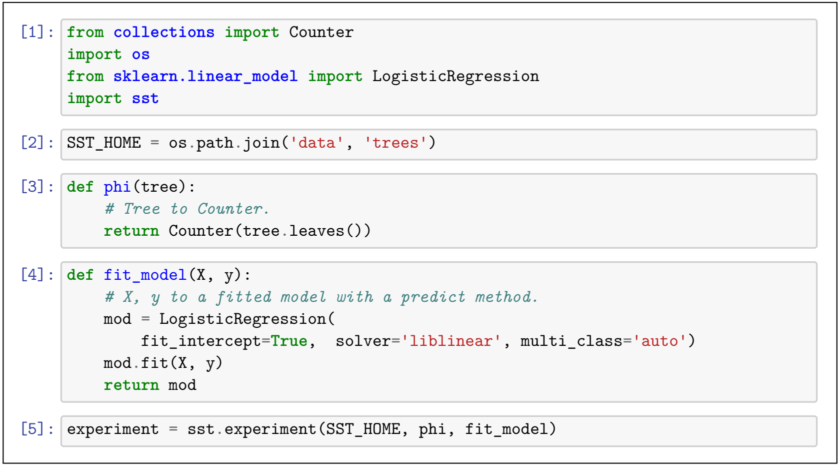
sst.experiment

The return value of `sst.experiment` is a `dict` packaging up the objects and info needed to test this model in new settings and conduct deep error analysis:



```
[10]: list(unigrams_softmax_experiment.keys())

[10]: ['model',
       'phi',
       'train_dataset',
       'assess_dataset',
       'predictions',
       'metric',
       'score']

[11]: list(unigrams_softmax_experiment['train_dataset'].keys())

[11]: ['X', 'y', 'vectorizer', 'raw_examples']
```

Bringing it all together



```
[1]: from collections import Counter
     import os
     from sklearn.linear_model import LogisticRegression
     import sst
```

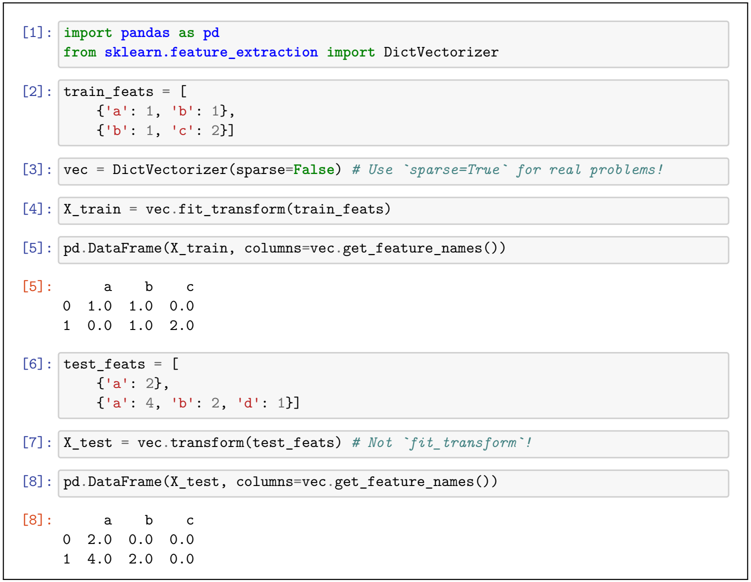
```
[2]: SST_HOME = os.path.join('data', 'trees')
```

```
[3]: def phi(tree):
         # Tree to Counter.
         return Counter(tree.leaves())
```

```
[4]: def fit_model(X, y):
         # X, y to a fitted model with a predict method.
         mod = LogisticRegression(
             fit_intercept=True,  solver='liblinear', multi_class='auto')
         mod.fit(X, y)
         return mod
```

```
[5]: experiment = sst.experiment(SST_HOME, phi, fit_model)
```

sklearn.feature_extraction.DictVectorizer



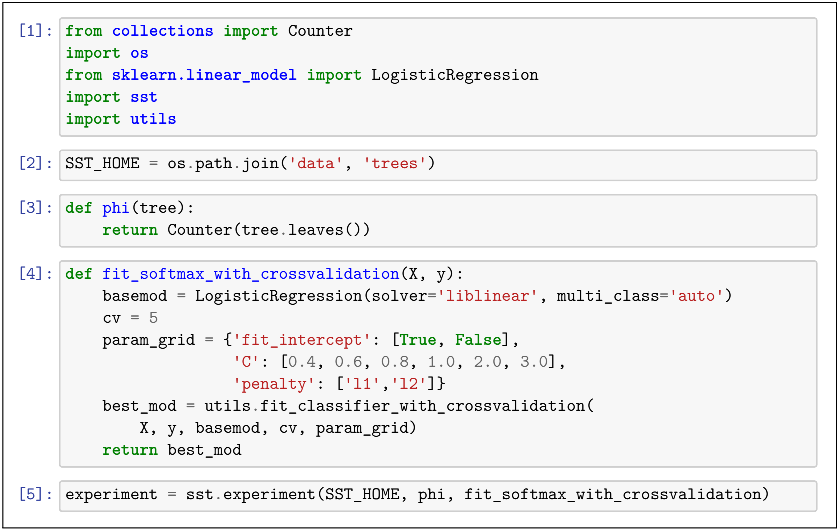
```
[1]: import pandas as pd
     from sklearn.feature_extraction import DictVectorizer

[2]: train_feats = [
         {'a': 1, 'b': 1},
         {'b': 1, 'c': 2}]

[3]: vec = DictVectorizer(sparse=False) # Use `sparse=True` for real problems!

[4]: X_train = vec.fit_transform(train_feats)

[5]: pd.DataFrame(X_train, columns=vec.get_feature_names())

[5]:      a    b    c
     0  1.0  1.0  0.0
     1  0.0  1.0  2.0

[6]: test_feats = [
         {'a': 2},
         {'a': 4, 'b': 2, 'd': 1}]

[7]: X_test = vec.transform(test_feats) # Not `fit_transform`!

[8]: pd.DataFrame(X_test, columns=vec.get_feature_names())

[8]:      a    b    c
     0  2.0  0.0  0.0
     1  4.0  2.0  0.0
```

Methods

Hyperparameter search: Rationale

1. The **parameters** of a model are those whose values are learned as part of optimizing the model itself.
2. The **hyperparameters** of a model are any settings that are set outside of this optimization. Examples:
   a. GloVe or LSA dimensionality
   b. GloVe $x_{\max}$ and α
   c. Regularization terms, hidden dimensionalities, learning rates, activation functions
   d. Optimization methods
3. Hyperparameter optimization is crucial to building a persuasive argument: every model must be put in its best light!
4. Otherwise, one could appear to have evidence that one model is better than other simply by strategically picking hyperparameters that favored the outcome.

Hyperparameter search in sst.py

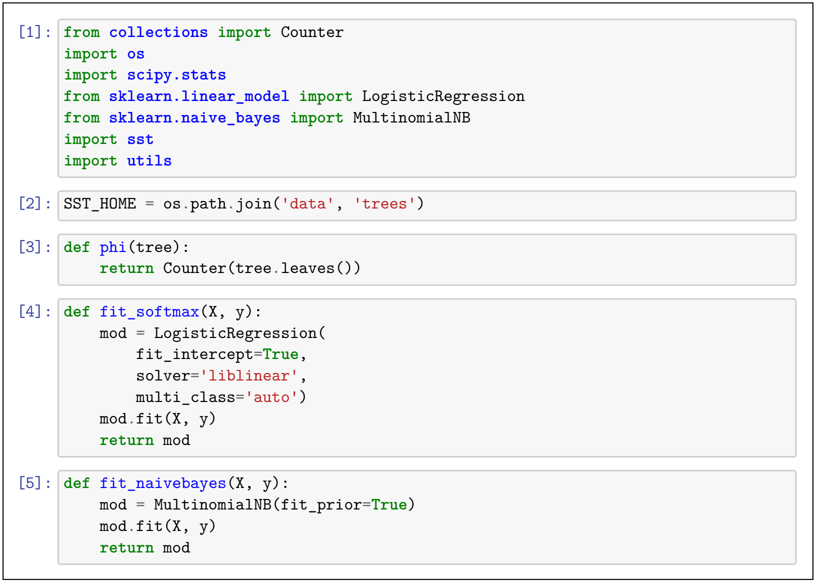


```
[1]: from collections import Counter
     import os
     from sklearn.linear_model import LogisticRegression
     import sst
     import utils

[2]: SST_HOME = os.path.join('data', 'trees')

[3]: def phi(tree):
         return Counter(tree.leaves())

[4]: def fit_softmax_with_crossvalidation(X, y):
         basemod = LogisticRegression(solver='liblinear', multi_class='auto')
         cv = 5
         param_grid = {'fit_intercept': [True, False],
                       'C': [0.4, 0.6, 0.8, 1.0, 2.0, 3.0],
                       'penalty': ['l1','l2']}
         best_mod = utils.fit_classifier_with_crossvalidation(
             X, y, basemod, cv, param_grid)
         return best_mod

[5]: experiment = sst.experiment(SST_HOME, phi, fit_softmax_with_crossvalidation)
```

Classifier comparison: Rationale

1. Suppose you've assessed a baseline model *B* and your favored model *M*, and your chosen assessment metric favors *M*. Is *M* really better?
2. If the difference between *B* and *M* is clearly of practical significance, then you might not need to do anything beyond presenting the numbers. Still, is there variation in how *B* or *M* performs?
3. Demšar (2006) advises the Wilcoxon signed-rank test for situations in which you can afford to repeatedly assess *B* and *M* on different train/test splits. We'll talk later in the term about the rationale for this.
4. For situations where you can't repeatedly assess *B* and *M*, McNemar's test is a reasonable alternative. It operates on the confusion matrices produced by the two models, testing the null hypothesis that the two models have the same error rate.

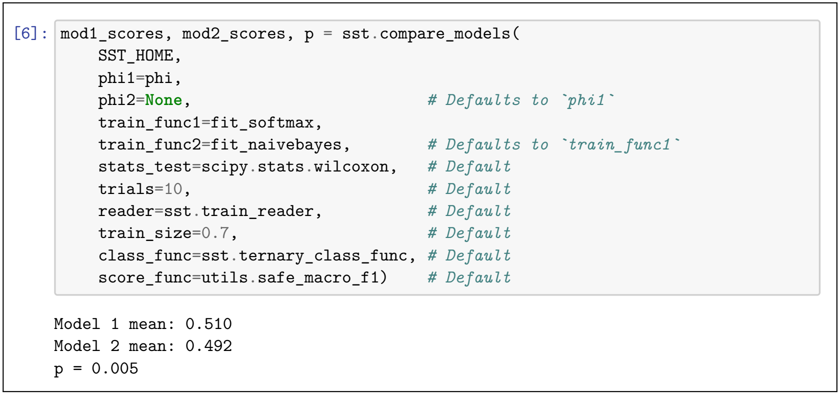
Classifier comparison in sst.py



```
[1]: from collections import Counter
     import os
     import scipy.stats
     from sklearn.linear_model import LogisticRegression
     from sklearn.naive_bayes import MultinomialNB
     import sst
     import utils

[2]: SST_HOME = os.path.join('data', 'trees')

[3]: def phi(tree):
         return Counter(tree.leaves())

[4]: def fit_softmax(X, y):
         mod = LogisticRegression(
             fit_intercept=True,
             solver='liblinear',
             multi_class='auto')
         mod.fit(X, y)
         return mod

[5]: def fit_naivebayes(X, y):
         mod = MultinomialNB(fit_prior=True)
         mod.fit(X, y)
         return mod
```

Classifier comparison in sst.py

Wilcoxon signed rank test

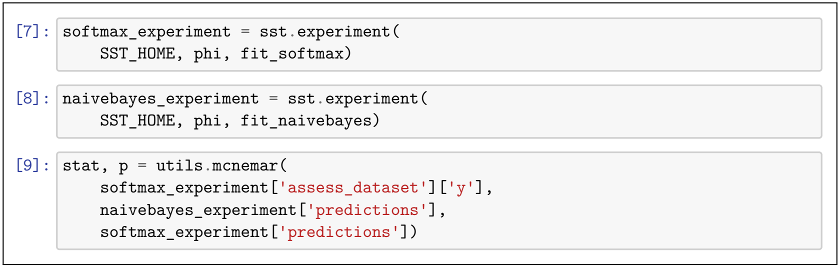


```
[6]: mod1_scores, mod2_scores, p = sst.compare_models(
         SST_HOME,
         phi1=phi,
         phi2=None,                        # Defaults to `phi1`
         train_func1=fit_softmax,
         train_func2=fit_naivebayes,       # Defaults to `train_func1`
         stats_test=scipy.stats.wilcoxon,  # Default
         trials=10,                        # Default
         reader=sst.train_reader,          # Default
         train_size=0.7,                   # Default
         class_func=sst.ternary_class_func, # Default
         score_func=utils.safe_macro_f1)   # Default
```

```
Model 1 mean: 0.510
Model 2 mean: 0.492
p = 0.005
```

Classifier comparison in sst.py

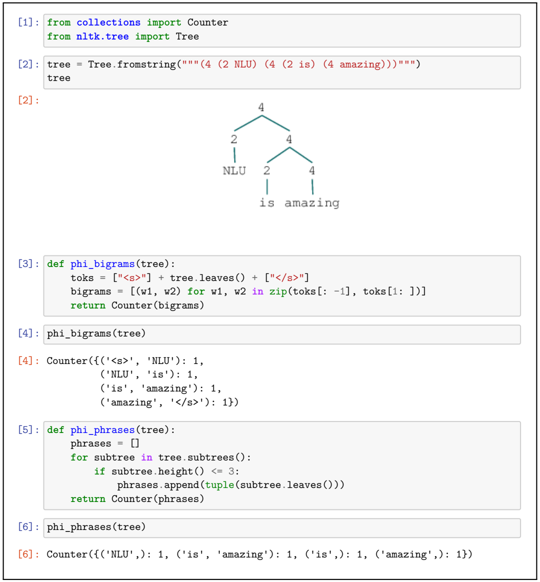
McNemar’s test



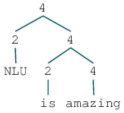
```
[7]: softmax_experiment = sst.experiment(
         SST_HOME, phi, fit_softmax)

[8]: naivebayes_experiment = sst.experiment(
         SST_HOME, phi, fit_naivebayes)

[9]: stat, p = utils.mcnemar(
         softmax_experiment['assess_dataset']['y'],
         naivebayes_experiment['predictions'],
         softmax_experiment['predictions'])
```

Feature representation

Hand-built features: Bags of subparts



```
[1]: from collections import Counter
     from nltk.tree import Tree

[2]: tree = Tree.fromstring("""(4 (2 NLU) (4 (2 is) (4 amazing)))""")
     tree

[2]:
```



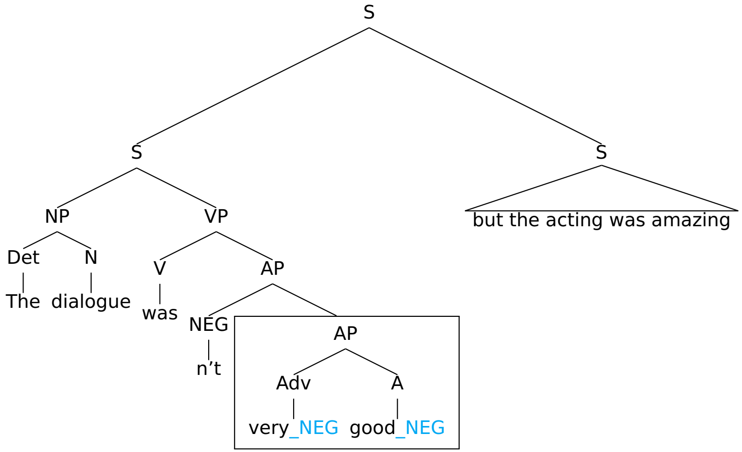
```
[3]: def phi_bigrams(tree):
         toks = ["<s>"] + tree.leaves() + ["</s>"]
         bigrams = [(w1, w2) for w1, w2 in zip(toks[: -1], toks[1: ])]
         return Counter(bigrams)

[4]: phi_bigrams(tree)

[4]: Counter({('<s>', 'NLU'): 1,
              ('NLU', 'is'): 1,
              ('is', 'amazing'): 1,
              ('amazing', '</s>'): 1})

[5]: def phi_phrases(tree):
         phrases = []
         for subtree in tree.subtrees():
             if subtree.height() <= 3:
                 phrases.append(tuple(subtree.leaves()))
         return Counter(phrases)

[6]: phi_phrases(tree)

[6]: Counter({('NLU',): 1, ('is', 'amazing'): 1, ('is',): 1, ('amazing',): 1})
```

Hand-built feature: Negation

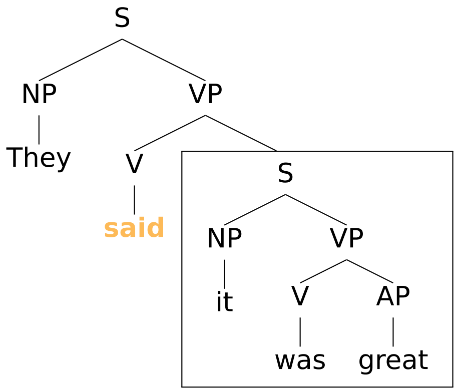
Simple negation marking

The dialogue was n’t very_NEG good_NEG but_NEG the_NEG acting_NEG was_NEG amazing_NEG ._NEG

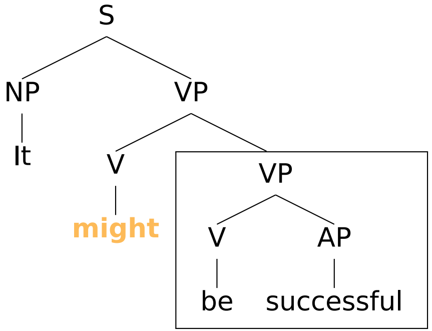
Negation marking based on structure



Extension to other kinds of scope-taking



Extension to other kinds of scope-taking



Other ideas for hand-built feature functions

- Lexicon-derived features
- Modal adverbs:
  - “It is quite possibly a masterpiece.”
  - “It is totally amazing.”
- Thwarted expectations:
  - “Many consider the movie bewildering, boring, slow-moving or annoying.”
  - “It was hailed as a brilliant, unprecedented artistic achievement worthy of multiple Oscars.”
- Non-literal language:
  - “Not exactly a masterpiece.”
  - “Like 50 hours long.”
  - “The best movie in the history of the universe.”

Assessing individual feature functions

1. `sklearn.feature_selection` offers functions to assess how much information your feature functions contain with respect to your labels.
2. Take care when assessing feature functions individually; correlations betwen them will make these assessments hard to interpret:

X_1	X_2	X_3	y
1	1	0	T
1	0	1	T
1	0	0	T
0	1	1	T
0	1	0	F
0	0	1	F
0	0	1	F
0	0	1	F

$\text{chi2}(X_1, y) = 3$

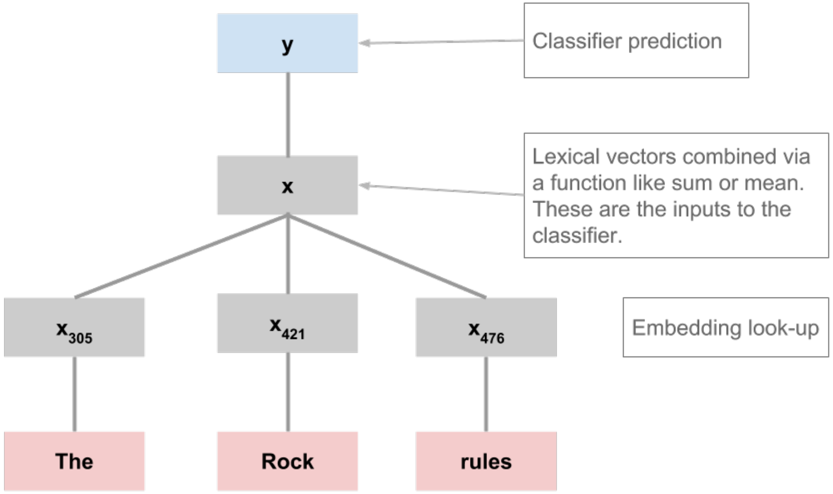
$\text{chi2}(X_2, y) = 0.33$

$\text{chi2}(X_3, y) = 0.2$

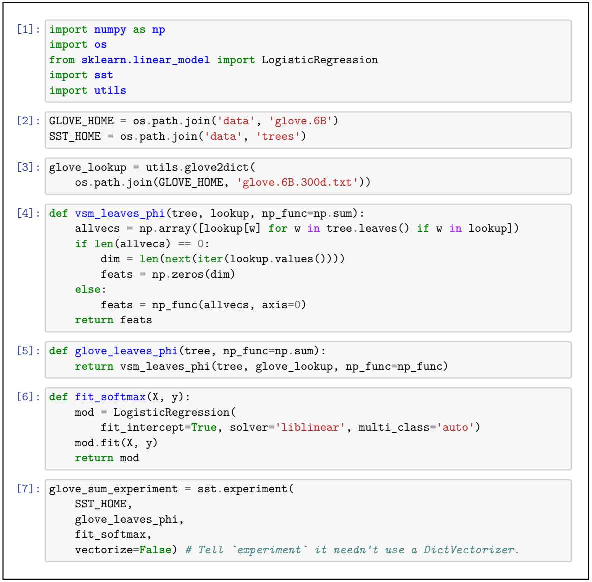
What do the scores tell us about the best model? In truth, a linear model performs best with just X_1, and including X_2 hurts.

3. Consider more holistic assessment methods: systematically removing or disrupting features in the context of a full model and comparing performance before and after.

Distributed representations as features


y
Classifier prediction
x
Lexical vectors combined via a function like sum or mean. These are the inputs to the classifier.
x_{305}
x_{421}
x_{476}
Embedding look-up
The
Rock
rules

Distributed representations as features



```
[1]: import numpy as np
     import os
     from sklearn.linear_model import LogisticRegression
     import sst
     import utils
```

```
[2]: GLOVE_HOME = os.path.join('data', 'glove.6B')
     SST_HOME = os.path.join('data', 'trees')
```

```
[3]: glove_lookup = utils.glove2dict(
         os.path.join(GLOVE_HOME, 'glove.6B.300d.txt'))
```

```
[4]: def vsm_leaves_phi(tree, lookup, np_func=np.sum):
         allvecs = np.array([lookup[w] for w in tree.leaves() if w in lookup])
         if len(allvecs) == 0:
             dim = len(next(iter(lookup.values())))
             feats = np.zeros(dim)
         else:
             feats = np_func(allvecs, axis=0)
         return feats
```

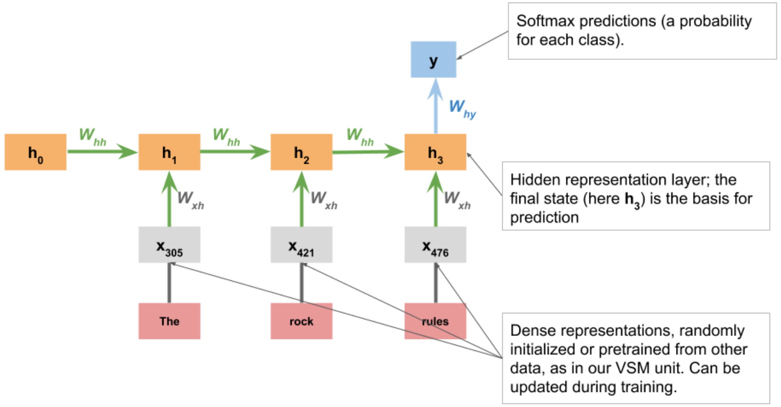
```
[5]: def glove_leaves_phi(tree, np_func=np.sum):
         return vsm_leaves_phi(tree, glove_lookup, np_func=np_func)
```

```
[6]: def fit_softmax(X, y):
         mod = LogisticRegression(
             fit_intercept=True, solver='liblinear', multi_class='auto')
         mod.fit(X, y)
         return mod
```

```
[7]: glove_sum_experiment = sst.experiment(
         SST_HOME,
         glove_leaves_phi,
         fit_softmax,
         vectorize=False) # Tell `experiment` it needn't use a DictVectorizer.
```

RNN classifiers

Model overview



Softmax predictions (a probability for each class).

Hidden representation layer; the final state (here $\mathbf{h}_3$) is the basis for prediction

Dense representations, randomly initialized or pretrained from other data, as in our VSM unit. Can be updated during training.

For complete details, see the reference implementation `np_rnn_classifier.py`

Standard RNN dataset preparation

	Embedding		
1	−0.42	0.10	0.12
2	−0.16	−0.21	0.29
3	−0.26	0.31	0.37

Examples [a, b, a]
[b, c]

$\Downarrow$

Indices [1, 2, 1]
[2, 3]

$\Downarrow$

Vectors

$$\Big[[-0.42\ 0.10\ 0.12], [-0.16\ -0.21\ 0.29], [-0.42\ 0.10\ 0.12]\Big]$$

$$\Big[[-0.16\ -0.21\ 0.29], [-0.26\ 0.31\ 0.37]\Big]$$

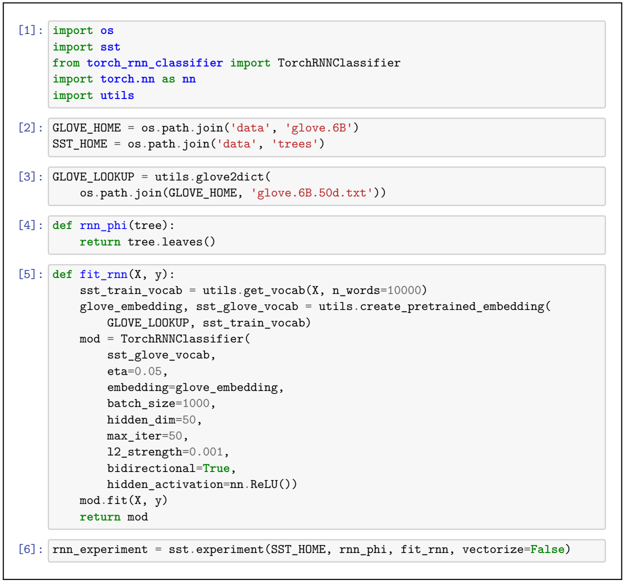
A note on LSTMs

1. Plain RNNs tend to perform poorly with very long sequences; as information flows back through the network, it is lost or distorted.

2. LSTM cells are a prominent response to this problem: they introduce mechanisms that control the flow of information.

3. We won't review all the mechanism for this here. I instead recommend these excellent blog posts, which include intuitive diagrams and discuss the motivations for the various pieces in detail:
   - Towards Data Science: Illustrated Guide to LSTM's and GRU's: A step by step explanation
   - colah's blog: Understanding LSTM networks

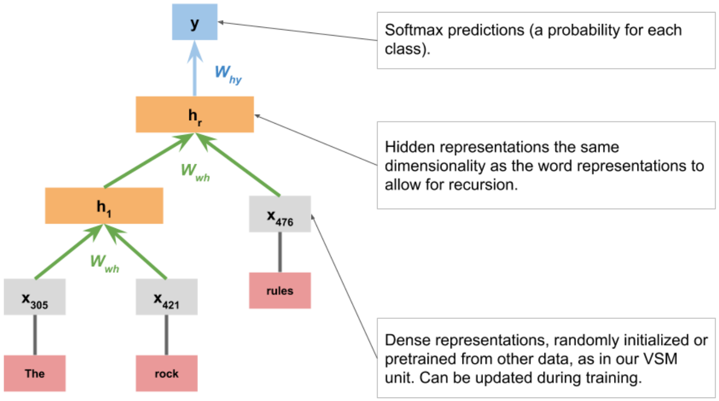
Code snippets



```
[1]: import os
     import sst
     from torch_rnn_classifier import TorchRNNClassifier
     import torch.nn as nn
     import utils

[2]: GLOVE_HOME = os.path.join('data', 'glove.6B')
     SST_HOME = os.path.join('data', 'trees')

[3]: GLOVE_LOOKUP = utils.glove2dict(
         os.path.join(GLOVE_HOME, 'glove.6B.50d.txt'))

[4]: def rnn_phi(tree):
         return tree.leaves()

[5]: def fit_rnn(X, y):
         sst_train_vocab = utils.get_vocab(X, n_words=10000)
         glove_embedding, sst_glove_vocab = utils.create_pretrained_embedding(
             GLOVE_LOOKUP, sst_train_vocab)
         mod = TorchRNNClassifier(
             sst_glove_vocab,
             eta=0.05,
             embedding=glove_embedding,
             batch_size=1000,
             hidden_dim=50,
             max_iter=50,
             l2_strength=0.001,
             bidirectional=True,
             hidden_activation=nn.ReLU())
         mod.fit(X, y)
         return mod

[6]: rnn_experiment = sst.experiment(SST_HOME, rnn_phi, fit_rnn, vectorize=False)
```

Tree-structured networks

Model overview

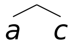

y
W_{hy}
h_r
W_{wh}
h_1
x_{476}
W_{wh}
x_{305}
x_{421}
rules
The
rock
Softmax predictions (a probability for each class).
Hidden representations the same dimensionality as the word representations to allow for recursion.
Dense representations, randomly initialized or pretrained from other data, as in our VSM unit. Can be updated during training.

For complete details, see the reference implementation `np_tree_nn.py`

Some alternative composition functions

Basic, as in the previous diagram (Pollack 1990)

$$h = f\left([a; c]W + b\right)$$



Matrix–Vector (Socher et al. 2012)

All nodes are represented by both vectors and matries, and the combination function creates a lot of multiplicative interactions between them.

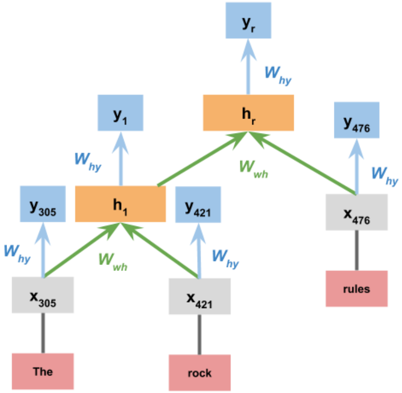
Tensor (Socher et al. 2013)

An extension of our basic model with a 3d tensor that allows for multiplicative interactions between the child vectors.

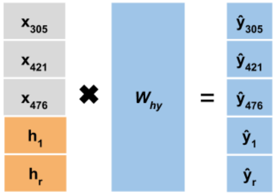
LSTM (Tai et al. 2015)

Each parent node combines separately-gated memory and hidden states of its children.

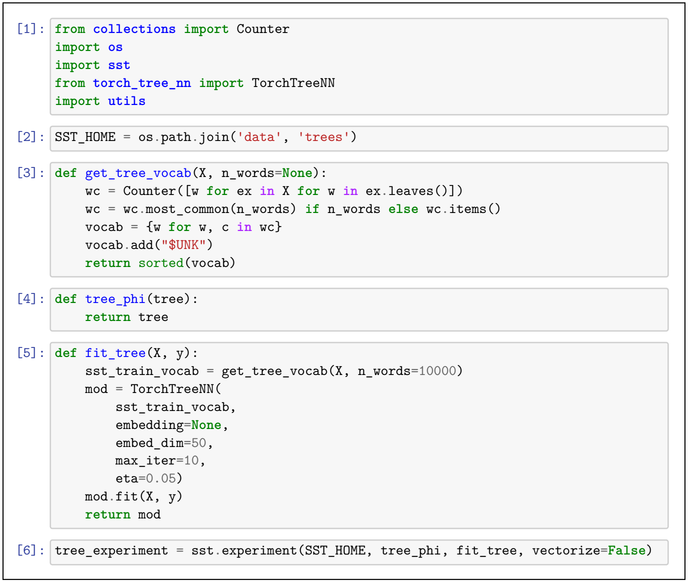
Subtree supervision



The total classifier error for the tree is the sum of the classifier errors of all its nodes, so:



Code snippets



```
[1]: from collections import Counter
     import os
     import sst
     from torch_tree_nn import TorchTreeNN
     import utils
```

```
[2]: SST_HOME = os.path.join('data', 'trees')
```

```
[3]: def get_tree_vocab(X, n_words=None):
         wc = Counter([w for ex in X for w in ex.leaves()])
         wc = wc.most_common(n_words) if n_words else wc.items()
         vocab = {w for w, c in wc}
         vocab.add("$UNK")
         return sorted(vocab)
```

```
[4]: def tree_phi(tree):
         return tree
```

```
[5]: def fit_tree(X, y):
         sst_train_vocab = get_tree_vocab(X, n_words=10000)
         mod = TorchTreeNN(
             sst_train_vocab,
             embedding=None,
             embed_dim=50,
             max_iter=10,
             eta=0.05)
         mod.fit(X, y)
         return mod
```

```
[6]: tree_experiment = sst.experiment(SST_HOME, tree_phi, fit_tree, vectorize=False)
```

References I

Pranav Anand, Marilyn Walker, Rob Abbott, Jean E. Fox Tree, Robeson Bowmani, and Michael Minor. 2011. Cats rule and dogs drool!: Classifying stance in online debate. In *Proceedings of the 2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis*, pages 1–9, Portland, Oregon. Association for Computational Linguistics.

Luke Breitfeller, Emily Ahn, Aldrian Obaja Muis, David Jurgens, and Yulia Tsvetkov. 2019. Finding microaggressions in the wild: A case for locating elusive phenomena in social media posts. In *Proceedings of 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics.

Marc Brysbaert, Amy Beth Warriner, and Victor Kuperman. 2014. Concreteness ratings for 40 thousand generally known English word lemmas. *Behavior Research Methods*, 46(3):904–911.

Justin Cheng, Michael Bernstein, Cristian Danescu-Niculescu-Mizil, and Jure Leskovec. 2017. Anyone can become a troll: Causes of trolling behavior in online discussions. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW ’17, pages 1217–1230, New York, NY, USA. ACM.

Cristian Danescu-Niculescu-Mizil, Moritz Sudhof, Dan Jurafsky, Jure Leskovec, and Christopher Potts. 2013. A computational approach to politeness with application to social factors. In *Proceedings of the 2013 Annual Conference of the Association for Computational Linguistics*, pages 250–259, Stroudsburg, PA. Association for Computational Linguistics.

Sanjiv Das and Mike Chen. 2001. Yahoo! for Amazon: Extracting market sentiment from stock message boards. In *Proceedings of the 8th Asia Pacific Finance Association Annual Conference*.

Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Gabriel Doyle, Dan Yurovsky, and Michael C. Frank. 2016. A robust framework for estimating linguistic alignment in twitter conversations. In *Proceedings of the 25th International World Wide Web Conference*, WWW ’16, pages 637–648, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Matthew Gentzkow, Jesse M. Shapiro, and Matt Taddy. 2019. Measuring group differences in high-dimensional choices: Method and application to congressional speech. Ms, Stanford University, Brown Universitym and Amazon.

Yoav Goldberg. 2015. A primer on neural network models for natural language processing. Ms., Bar Ilan University.

William L. Hamilton, Kevin Clark, Jure Leskovec, and Dan Jurafsky. 2016. Inducing domain-specific sentiment lexicons from unlabeled corpora. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 595–605, Austin, Texas. Association for Computational Linguistics.

Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*, WWW ’16, pages 507–517, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. 2017. A large self-annotated corpus for sarcasm. *arXiv preprint arXiv:1704.05579*.

References II

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 142–150, Portland, Oregon. Association for Computational Linguistics.

Julian McAuley and Jure Leskovec. 2013. From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd International World Wide Web Conference*, pages 897–907, New York. ACM.

Julian McAuley, Jure Leskovec, and Dan Jurafsky. 2012. Learning attitudes and attributes from multi-aspect reviews. In *12th International Conference on Data Mining*, pages 1020–1025, Washington, D.C. IEEE Computer Society.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’15, pages 43–52, New York, NY, USA. ACM.

Vlad Niculae, Srijan Kumar, Jordan Boyd-Graber, and Cristian Danescu-Niculescu-Mizil. 2015. Linguistic harbingers of betrayal: A case study on an online strategy game. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1650–1659, Beijing, China. Association for Computational Linguistics.

Chikashi Nobata, Joel Tetreault, Achint Thomas, Yashar Mehdad, and Yi Chang. 2016. Abusive language detection in online user content. In *Proceedings of the 25th International Conference on World Wide Web*, WWW ’16, pages 145–153, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 115–124, Ann Arbor, MI. Association for Computational Linguistics.

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1):1–135.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 79–86, Philadelphia. Association for Computational Linguistics.

Jordan B. Pollack. 1990. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105.

Reid Pryzant, Richard Diehl Martinez, Nathan Dass, Sadao Kurohashi, Dan Jurafsky, and Diyi Yang. 2020. Automatically neutralizing subjective bias in text. In *Proceedings of AAAI*.

Marta Recasens, Cristian Danescu-Niculescu-Mizil, and Dan Jurafsky. 2013. Linguistic models for analyzing and detecting biased language. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1650–1659, Sofia, Bulgaria. Association for Computational Linguistics.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211, Stroudsburg, PA.

References III

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Stroudsburg, PA. Association for Computational Linguistics.

Moritz Sudhof, Andrés Gómez Emilsson, Andrew L. Maas, and Christopher Potts. 2014. Sentiment expression conditioned by affective transitions and social forces. In *Proceedings of 20th Conference on Knowledge Discovery and Data Mining*, pages 1136–1145, New York. ACM.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured Long Short-Term Memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Zijian Wang and Christopher Potts. 2019. TalkDown: A corpus for condescension detection in context. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3702–3710, Stroudsburg, PA. Association for Computational Linguistics.

Robert West, Hristo S. Paskov, Jure Leskovec, and Christopher Potts. 2014. Exploiting social network structure for person-to-person sentiment analysis. *Transactions of the Association for Computational Linguistics*, 2(2):297–310.