

Counting Feeling: Affect Theory and Sentiment Analysis in TextBlob

Jeffrey Moro <jmoro_at_umd_dot_edu>, University of Maryland  <https://orcid.org/0000-0002-0998-7660>

DOI: <https://doi.org/10.63744/7gengas6hkc>

Abstract

Sentiment analysis encompasses a range of computational techniques for detecting and quantifying the presence of affect and emotion in written texts. In this essay, I use the techniques of critical code studies to offer a case study of one such sentiment analysis tool called TextBlob, about which I advance two intertwined claims. First, I map TextBlob's reliance on a web of programmatic and textual dependencies, and how in turn, TextBlob effaces these dependencies' formal specificity in the service of computational processing. This effacement supports my second claim: that TextBlob, and sentiment analysis more generally, models affect as programmatically latent in the smallest particles of language, which it then seeks to make available for computational processing. I explore the consequences of this computational model of affect for digital humanists seeking to integrate sentiment analysis and similar tools (including AI/LLMs) into their work.

Sentiment analysis encompasses a range of computational techniques for detecting and quantifying the presence of affect and emotion in written texts [Liu 2015]. Already, in this first sentence, we find sites for definitional, methodological, and even moral contestation. Across fields such as psychology, philosophy, literary studies, computer science, to say nothing of their countless interdisciplinary sub-specializations, the questions “what is emotion?”, “how is emotion inhered within or otherwise expressed through writing?”, and “how might computers extract or otherwise represent such emotion in informatic forms?” would seem to have no easy answers; for some fields, even attempting to answer them at all is a category error. For precisely these reasons, I want to offer sentiment analysis as a provocation to the digital humanities (DH). DH has long defined itself by the application of computational technologies to humanistic inquiry, and, to a lesser extent vice versa, the application of humanities questions to the study of technology itself. ^[1] Sentiment analysis, by appealing simultaneously to the humanistic idea of emotion and the computational idea of its quantification — all mediated through the messy, muddled, miraculous middle of text — stages some of the contradictions inherent to DH as a field while offering generative ground from which to project its possible futures, particularly in the current artificial intelligence moment.

1

A subfield of natural language processing (NLP), itself a field of computer science research interested in the machine address of human languages, sentiment analysis uses statistical models to attempt to measure such quantities as a text's “polarity” (where it falls on a basic scale of positivity and negativity) or even more complex blends of emotional qualities such as anger, joy, or sarcasm. Computer science has engaged sentiment analysis since the field's inception, with its key research questions latent in early artificial intelligence research such as those of [Turing 1950] or [Stone 1966]. Sentiment analysis was mainly a research curiosity for much of the twentieth century. It was only with the emergence of the internet and concomitant increases in computing power from the 1990s onwards that sentiment analysis became a practical undertaking. In the 2010s, sentiment analysis became more attractive to businesses: for example, through such tasks as analyzing consumers' internet posts about a new product to determine its reception. Research towards sentiment analysis furthermore undergirds, at least in part, the large language model (LLM) boom of

2

the early 2020s, which suggests larger political and philosophical concerns around the automated recognition of emotion in digital texts *en masse*.

Today, sentiment analysis undergirds consumer-facing technologies such as chatbots and algorithmic recommender systems, as well as behind-the-scenes applications such as data mining and marketing analytics. It has also found purchase in DH research. A search of this very publication surfaces roughly three dozen applications of sentiment analysis to such DH programs as the automatic recognition of rhetorical features in texts; sentiment analysis also undergirds such tools as DH scholar Matthew Jockers' *syuzhet* program, a code library written in the language R for "the extraction of sentiment and sentiment-based plot arcs from text" [Jockers 2014]. As such, a critical study of sentiment analysis as a technology has the potential to speak to many aspects of contemporary algorithmic culture, alongside more theoretical and methodological questions facing those scholars interested in the computational address of texts. In this essay, I use the techniques of critical code studies to offer a case study of one such sentiment analysis tool called TextBlob. TextBlob, developed from 2013 to the present day by software engineer Steven Loria, is a Python library that provides simple, off-the-shelf tools for NLP tasks, including sentiment analysis. It has two chief virtues for this kind of critical study: first, it is sufficiently small in scope, such that scholars (myself included) might be able to practice meaningfully the kind of close-code reading that critical code studies demands; and second, TextBlob is commonly used in classroom instruction and preliminary DH research, making it an especially effective tool through which to articulate some of the implications for sentiment analysis research to DH specifically.

I'll restrict my focus to one sub-program within TextBlob more generally: `en/sentiments.py`, a 97-line file that wraps and compiles functions for breaking down textual input into composite parts, assigns those parts polarity scores, and computes overall scores. [2] `en/sentiments.py` also contains the core implementation for two distinct sentiment analysis functions, named `PatternAnalyzer` and `NaiveBayesAnalyzer`, which affords the opportunity in this essay for their comparative study. Through this case study, I advance two intertwined claims about TextBlob. First, I map TextBlob's reliance on a web of programmatic and textual dependencies, and how in turn, TextBlob effaces these dependencies' formal specificities in the service of computational processing. From a technical perspective, `en/sentiments.py` does little computational lifting of its own. Instead, it draws together prior work from a range of other Python tools, in particular the libraries `Pattern` and `NLTK`, alongside textual corpora pre-packaged with these tools. [3] While such dependencies are commonplace in computer code, TextBlob is notable for how it instrumentalizes a range of culturally and materially disparate sources in the service of providing its putatively "objective" polarity calculations.

Mapping these dependencies supports this essay's second claim: that TextBlob models sentiment as programmatically latent in the smallest particles of language, which it then seeks to make available for extraction and computation. TextBlob envisions sentiment as discrete and encoded within individual words, in short, as *data*. These data, in turn, are building blocks upon which sentiment analysis researchers develop more sophisticated models of *affect*, or the primary biological impulses which give rise to emotions and moods. [4] Drawing on work on affect and information by Silvan Tompkins, Eve Kosofsky Sedgwick, and N. Katherine Hayles, among others, I argue that TextBlob's model of the relationship between sentiment, affect, and word undergirds a textual laundering inherent to much contemporary sentiment analysis, whereby culturally situated judgments (scraped, say, from corpora of book or movie reviews, both of which find their way into the TextBlob code base) come to stand for objective fact. Emotion, in turn, becomes available for computational extraction and instrumentalization. Whether or not TextBlob is empirically successful at identifying affect — if indeed affect is computationally identifiable at all — is secondary to the conceptual work of *inventing* affect through this textual laundering. Sentiment analysis does not measure sentiment: it creates it, and the circumstances of sentiment's creation have significant ramifications both for the critical work of the digital humanities and our contemporary algorithmic cultures more generally.

The program of linking affect to data is not unique to computer science. Early in the first volume of his magnum opus *Affect, Imagery, Consciousness*, published in 1962, psychologist and foundational affect theorist Silvan Tompkins

articulates the study of affect in relation to a then-unusual parallel field: the study of artificial intelligence. Unlike many of his contemporaries in the computer sciences (who he scorns as “temperamentally unsuited to create and nurture mechanisms” capable of true judgment [Tomkins 2008]), Tomkins argues that the capacity for computational intelligence as such depends on the creation of an “affect system.” An affect system is a theoretically discrete neurobiological system that mediates external stimuli into physical and emotional experience. Tomkins’ classic example is that of asphyxiation: the stimulus of constricted breath activates the specific affective channel of “fear,” which an individual then mediates, based on their own lived experience, into a range of emotional reaction. In this model, affect is a “system” organized around specific “programs,” which activate “rewarding and punishing characteristics,” in essence, a feedback mechanism [Tomkins 2008].

In their 1995 essay introducing Tomkins’ affect theory to literary studies, “Shame in the Cybernetic Fold: Reading Silvan Tomkins”, Eve Kosofsky Sedgwick and Adam Frank position his work within the cybernetic milieu of the post-’45 United States. Affect theory emerges, they argue, within intellectual frameworks of systems theory, “fold[ing]” across the technological and biological, the digital and the analog [Sedgwick and Frank 1995]. For Sedgwick and Frank, Tomkins’ work reorients literary theory toward richer considerations of interdisciplinarity and “the dynamics of consensus formulation” across fields [Sedgwick and Frank 1995]. This is a particularly rich vein of consideration for the study of sentiment analysis, given that the technology is inherently polysemous, with charged terms such as “affect” and “emotion” signifying quite differently across its related fields. But more to the point, Sedgwick and Frank point to affect’s explicitly computational intellectual history as a term. Addressing affect as information or data that circulates within and fine-tunes cognitive systems, whether human or machine, is a therefore a lineage that begins not with sentiment analysis as a technology, but rather affect’s emergence as a concept more generally. [5]

Loria describes TextBlob as a “library for processing textual data” [Loria 2020]. Before it can compute relationships *across* data, it must first mediate texts *into* data. Many core NLP tasks concern the reduction of complex textual inputs into smaller lexical units, which computers can more easily process and compare statistically. One such technique deployed in `en/sentiments.py` is called “tokenization.” In NLP, a token refers to an arbitrary unit of lexical information. A program may tokenize a paragraph into sentences, a sentence into words, a word into syllables, or a syllable into letters, depending on the researchers’ questions and interests [Manning et al 2008]. Programs can further reduce related words (for instance, different inflections of the same verb) through computational processes such as stemming (removing affixes and suffixes) and lemmatization (removing inflectional endings, e.g., “-ly” for adverbs). TextBlob’s operative level of tokenization is that of individual words. Line 11 in `en/sentiments.py` imports a tokenization function from elsewhere in the code base, itself lightly adapted from and wrapping a related function in NLTK. Line 85 demonstrates this function in action, applying `word_tokenize` to the input text, making it subsequently available for further filtering and feature extraction.

The decision to tokenize at the level of words has two consequences for TextBlob’s sentiment analysis functions. First, it renders word order irrelevant. For example, `PatternAnalyzer`, TextBlob’s primary sentiment analysis function (recall that it has two built in) assigns the example sentence “I love those who hate me” a -0.15 polarity score, indicating slight negativity. Rearranging the sentence to “I hate those who love me,” theoretically reversing the sentence’s meaning, returns the same score. From this follows the second consequence: namely, that TextBlob inheres affective meaning *within* single words, stripped bare of syntactical context and morphological derivation. A programming choice perhaps designed to lower the computational load thus produces a cognitive and conceptual model. [6] Furthermore, TextBlob imagines sentiment within these words as mathematically fixed, capable of adding to or subtracting from the sentiments of other words, but not changing based on other linguistic characteristics.

Due to the nested dependencies and imported functions in `en/sentiments.py`, it takes sleuthing to see where and how TextBlob defines these eternal constants. For `PatternAnalyzer`, the process of assigning and computing polarity scores happens with the terse invocation of `pattern_sentiment(text)`, a function imported into, rather than defined within, this file. We can follow this function along a chain of interoperable files to `en-sentiment.xml`, a lexicon file containing 2,918 individual words with associated polarity scores. Entries follow a standardized format: [7]

```

<sentiment language="en" version="1.3" author="Tom De Smedt, Walter
Daelemans" license="PDDL">
    ...
    <word form="airheaded" cornetto_synset_id="n_a-507793" wordnet_id="a-
02120828" pos="JJ" sense="lacking seriousness" polarity="0.5"
subjectivity="1.0" intensity="1.0" confidence="0.8" />
    <word form="alarming" cornetto_synset_id="n_a-527099" wordnet_id="a-
00193015" pos="JJ" sense="frightening because of an awareness of danger"
polarity="-0.1" subjectivity="0.6" intensity="1.0" confidence="0.8" />
    <word form="alas" wordnet_id="" pos="UH" polarity="-0.4" subjectivity="1.0"
intensity="1.0" confidence="0.8" />
    ...
</sentiment>

```

Just as Loria imports the Pattern library’s algorithmic functionality, so too does he re-use De Smedt and Daelemans’ lexicon. In their 2012 paper introducing Pattern, De Smedt and Daelemans describe using the web scraping tool to produce the lexicon itself: “We mined online Dutch book reviews and extracted the 1,000 most frequent adjectives. These were manually annotated with positivity, negativity, and subjectivity scores,” a task they repeated with a number of European languages, including English [De Smedt and Daelemans 2012]. “Manual annotation” means exactly what it sounds like: De Smedt and Daelemans hand-tagged (or computationally inferred, for words with related senses and meanings) words with polarity scores. These scores came from their own critical judgment rather than any computational process. While this practice may raise eyebrows to an audience of digital humanists — it seems an excellent way to encode a whole range of unattested biases and subjectivities into the program, for one — it’s a standard approach in NLP. While programmers often document such development processes in code comments and research papers, these innate subjectivities get effaced in the movement from data to code. Here I want to emphasize that I am not accusing Loria or any NLP researchers of intentionally black-boxing the textual laundering that necessarily occurs in the production of these corpora. De Smedt and Daelemans have accounted for their methodologies in their published work, and these methodologies follow best practices within their field. Rather, my point is that the operation of *encoding* these corpora within TextBlob or any similar sentiment analysis tool is where effacement occurs. In order to make language “work” as data, TextBlob strips it clean of context, even as that context is what allowed the initial humans composing these corpora to make judgments in the first place.

11

While PatternAnalyzer uses simple averaging to compute its scores, the second sentiment analysis program mapped in `en/sentiments.py`, named NaiveBayesAnalyzer, deploys machine learning techniques. True to its name, it uses a naïve Bayes algorithm to assign probable (rather than definitive) polarity scores. Named after Reverend Thomas Bayes, an eighteenth-century English mathematician, a Bayesian algorithm is one based off Bayes’ Theorem, which offers a simple yet effective statistical model for predicting the likelihood of a given event. [8] A Bayesian algorithm is “naïve” when it presumes that all units of information are discrete and have no other statistically meaningful relationships: for example, when one computes a sentence’s polarity scores based on individual words alone, regardless of syntactical context. Even given NaiveBayesAnalyzer’s relative sophistication when compared to PatternAnalyzer, they share many of the same conceptual underpinnings. Both decompose inputs into individual words, in doing so obviating finer points of lexical and syntactical relation. I make this observation, once again, not to dismiss the projects of either; sentiment analysis is an extraordinarily taxing operation, both at the level of conceptual development and machine operations. The task demands simplification, compression, and abstraction. What I am suggesting is that programs such as TextBlob could do more, from a user perspective, to flag the necessarily subjective models that underpin their analysis — particularly in the LLM moment, when the validity of automated computer-generated textual analysis is increasingly taken as a given by the public at large. Doing so would necessarily undercut sentiment analysis’s claims to objective utility. But by contrast, it would more accurately articulate what the technology

12

actually does: namely, develop (imperfect, partial, but potentially useful) models of critical judgment.

Given that TextBlob provides two distinct sentiment analysis implementations, it provides us the opportunity to see how such models diverge. The following code runs the first sentence of this essay through both `PatternAnalyzer` and `NaiveBayesAnalyzer`:

13

```
# Runs the example sentence through `PatternAnalyzer`
>>> from textblob import TextBlob
>>> blob = TextBlob("Sentiment analysis encompasses a range of
computational techniques for detecting and quantifying the presence of
affect and emotion in written texts.")
>>> blob.sentiment.polarity
0.0
# `PatternAnalyzer` returns a score of zero, indicating no positivity or
negativity. In reality, this means that none of the sentence's words were
present in `en-sentiment.xml`.
# These next commands call `NaiveBayesAnalyzer` specifically.
>>> blob = TextBlob("Sentiment analysis encompasses a range of
computational techniques for detecting and quantifying the presence of
affect and emotion in written texts.", analyzer=NaiveBayesAnalyzer())
>>> blob.sentiment
Sentiment(classification='pos', p_pos=0.9916363422665231,
p_neg=0.008363657733478571)
# `NaiveBayesAnalyzer` returns a sharply positive score, with a hint of
negative characteristics.
```

Here, the model's limitations become clear. While `PatternAnalyzer` appears to compute the sentence as perfectly neutral, in fact the score of 0.0 indicates that none of its constituent words appeared in De Smedt and Daelemans' lexicon. `NaiveBayesAnalyzer` fares a tad better, given that it returns a score at all. However, as a human reader (to say nothing of the sentence's author), I will admit that its sharply positive assessment gives me pause. Where do we locate such affirmative affect within an admittedly and self-consciously dry academic sentence? The gulf between human and machine interpretation becomes evident. N. Katherine Hayles argues that machine reading is principally distinct from human reading in its focus on interior, statistical connections between textual data rather than expansive, mediated contexts [Hayles 2018]. This is not to privilege one form of reading over another, although certainly from a human perspective, we may find the machine's ability to "correctly" interpret these texts lacking. On the one hand, from my position as a critic trained in literary and media studies, I am skeptical that TextBlob's sentiment analysis functions can do what they claim. I root this skepticism not in any particular valorization of human judgment (although my ethical and political convictions in the LLM moment admittedly encourage me to do so), but rather in the machine capacity to model human judgment through any statistical system, whether one as simple as TextBlob's or as potentially complex as ChatGPT's. On the other hand, I am intrigued by sentiment analysis's capacities to model different forms of critical judgment, and in doing so remove them from an exclusively human context. When `NaiveBayesAnalyzer` returns a sharply positive score for my own prose, I take that as a moment less to disagree with the machine's interpretation and more to ask after the conditions that gave rise to such an interpretation: what the machine is *doing* with and to my text. This may not tell me anything new about the sentence itself, or at least nothing that I could not have already said as a human reader, but it does have the capacity to tell me more about the underlying models that power the program's interpretation. Texts, it turns out, are not quieted so easily.

14

`PatternAnalyzer` and `NaiveBayesAnalyzer` differ not only in how they compute sentiment statistically, but also in the root sources of their judgments. `PatternAnalyzer`, we have seen, draws its claims from the hand-tagged Pattern

15

lexicon, in which each individual word arrives imputed with a discrete sentiment score, manually tagged by human readers. `NaiveBayesAnalyzer`, conversely, derives root scores programmatically by training a rudimentary machine learning model on a corpus of movie reviews assembled by NLP researchers Bo Pang and Lillian Lee [Pang and Lee 2005]. Pang and Lee's work is influential in NLP and sentiment analysis research for addressing various computational problems related to the relationships between ratings systems and language: is there, for instance, a textual pattern, discernable at the level of tokenization, that distinguishes a two-star review from a four-star review? The various corpora they have assembled over the years as part of this research program have subsequently become popular readymades for programmers like Loria, who need well-formatted textual-numerical data upon which to derive sentiment analysis models like `NaiveBayesAnalyzer`. In `TextBlob`, this training happens in lines 72–81:

```

72      """Train the Naïve Bayes classifier on the movie
review
corpus."""
73      super(NaiveBayesAnalyzer, self).train()
74      neg_ids = nltk.corpus.movie_reviews.fileids('neg')
75      pos_ids = nltk.corpus.movie_reviews.fileids('pos')
76      neg_feats = [(self.feature_extractor(
77          nltk.corpus.movie_reviews.words(fileids=[f])),
'neg') for f
in neg_ids]
78      pos_feats = [(self.feature_extractor(
79          nltk.corpus.movie_reviews.words(fileids=[f])), 'pos')
for f in pos_ids]
80      train_data = neg_feats + pos_feats
81      self._classifier =
nltk.classify.NaiveBayesClassifier.train(train_data)

```

In plain English, these lines call various submodules from NLTK (another dependency moment) to define both positive and negative “features,” or numerical representations of sentiment scores, derived from the Pang and Lee movie review corpus. These features are then assembled in line 80 into training data, which are used in line 81 to train the classifier itself.

We are now looking the black box in its vacant eye. What we see is both opaque and prosaic: opaque in that the fundamental features upon which `NaiveBayesAnalyzer` construes its judgments are hidden from us; yet prosaic in that we see, as plainly as one and one make two, the roots of the training data in line 80. That these data are derived from movie reviews is not — cannot be — incidental to `NaiveBayesAnalyzer`'s judgment, for the same reason that no human judgment is arbitrary (even if it can be capricious): there are always priors, and those priors ramify for the simple reason that if they were different — if the training data were otherwise — the judgments would be too. Here, I want to once again stay cautious about the applicability of generalizing familiar models of human judgment onto machines. We can speculate on the specific formal consequences of the Pang and Lee corpus, derived as it is from movie reviews, being so popular as a readymade in NLP implementations. How one speaks of a movie is not the same as how one speaks of a device, or a student, or a loved one, or the weather, or any of the countless topics upon which we are called to issue judgments in the contemporary world. But given the classifier's opacity, even with access to the features themselves it would be challenging to claim with any certainty the existence of causal relationships between the logic of movie reviews and subsequent algorithms. (It might be, for instance, just as likely that any determinative logic from the movie reviews get thoroughly decomposed in the classification process, a regression-to-the-mean familiar to any user of an LLM, in which the particularities of underlying corpora evaporate into a vague grey goo of text.)

What we *can* say, however, is that the Pang and Lee corpus of movie reviews, alongside similar corpora of product

reviews or social media posts, emerge from a shared rhetorical situation: namely, one in which participants in the modern internet are invited, repeatedly and at length, to offer their opinion about things and to encode those opinions within numerical systems. The endless invitation to grade objects, experiences, and our fellow humans on a five-star scale is one of the underappreciated burdens of modern life. It is a computational-capitalist logic that mediates the actually existing world into datasets that are subsequently available for precisely the kinds of implementations we find in TextBlob: pieces of software that emulate human judgment. Or rather, that emulate humans who have been prompted to behave as machines — not judging but rating, our messy, textured opinions reduced to a clean, legible numerical scale. We have come full circle, it seems, from Tomkins finding in the machine a model for human emotion and cognition. Now we have machines that replicate humans behaving like machines.

`en/sentiments.py` provides a brief if telling glimpse into the conceptual assumptions and encoded infrastructures underpinning sentiment analysis: that language is fundamentally reducible to mathematical information; that statistical techniques can construe meaningful relationships across this information; and that these mathematical relationships can *mean* things to the register of language. In addressing affect specifically, `en/sentiments.py` participates in what Patricia Ticineto Clough and her collaborators have called the “datalogical turn,” or “how the algorithms that parse big data are an intensification of . . . [an] unconscious drive to empiricism, positivism, and scienticism” [Matviyenko 2025]. To these we could also add work by scholars such as Safiya Noble and Ruha Benjamin on algorithmic bias, how “automatic” judgments are never thus, but rather encode the presumptions of those humans who create, use, and maintain judging technologies [Noble 2018], [Benjamin 2019].

19

I have argued in this essay that there is a fundamental slippage between what sentiment analysis claims to do and what it actually does. Rather than identify and quantify empirically existing affect within language (a contestable claim in the best of circumstances), sentiment analysis as implemented in TextBlob *creates* affect as a schema from a heterogeneous array of pre-existing judgments, in turn flattening these judgments into a computational voice of God, discerning from at once everywhere and nowhere. While NLP research understandably focuses on improving the accuracy of such underlying models — “accuracy” here serving synecdochally for concepts such as neutrality, objectivity, and factuality — my interest as a digital humanist in technologies such as TextBlob is not with their always deferred capacity to determine a text’s affect, but rather with affect’s participation in a theoretical (and indeed, quite material) project of equating the human spirit to that of the machine. ^[9] As digital humanists, we have an obligation to surface the cultural work that technologies perform before adopting them into our enterprise. In the case of sentiment analysis, this entails acknowledging how the technology’s epistemic operations rest on the elimination (or at best instrumentalization) of context — perhaps the fundamental unit of humanistic inquiry.

20

I am not dissuading scholars from using these tools; I myself have found them generative in my work chiefly for the strangeness of their readings. (I am not alone, I suspect, in finding the whole AI enterprise far more interesting when its outputs resembled humans less and machines more.) I hope in this case study to have suggested terrain for further experimentation with sentiment analysis, if for nothing else than as models for ways to work with the technologies that are different from the tech industry’s naked denigration of human judgment. These are qualities that a critical code studies reading can surface, and that I argue are integral to reshaping such technologies along more creative, expressive, and ethical lines.

21

Annotations

- File: `en/sentiments.py`
- Programming language: Python
- Developed: 2013–Present day
- Principal author: Steven Loria
- Platform: Cross-platform (Windows, macOS, Linux)
- Libraries used: NLTK
- Source file: <https://github.com/slوريا/TextBlob/blob/dev/textblob/en/sentiments.py>

- Interoperating files: base.py, sentiments.py, text.py, en/init.py, en/en-sentiment.xml

Annotation continued

```
1      # -*- coding: utf-8 -*-
2      """Sentiment analysis implementations.
3
4      .. versionadded:: 0.5.0
5      """
6      from __future__ import absolute_import
7      from collections import namedtuple
8
9      import nltk
10
11      from textblob.en import sentiment as pattern_sentiment
12      from textblob.tokenizers import word_tokenize
13      from textblob.decorators import requires_nltk_corpus
14      from textblob.base import BaseSentimentAnalyzer, DISCRETE, CONTINUOUS
15
16
17      class PatternAnalyzer(BaseSentimentAnalyzer):
18          """Sentiment analyzer that uses the same implementation as the
19          pattern library. Returns results as a named tuple of the form:
20
21          ``Sentiment(polarity, subjectivity, [assessments])``
22
23          where [assessments] is a list of the assessed tokens and their
24          polarity and subjectivity scores
25          """
26          kind = CONTINUOUS
27          # This is only here for backwards-compatibility.
28          # The return type is actually determined upon calling analyze()
29          RETURN_TYPE = namedtuple('Sentiment', ['polarity', 'subjectivity'])
30
31          def analyze(self, text, keep_assessments=False):
32              """Return the sentiment as a named tuple of the form:
33              ``Sentiment(polarity, subjectivity, [assessments])``.
34              """
35              #: Return type declaration
36              if keep_assessments:
37                  Sentiment = namedtuple('Sentiment', ['polarity',
38                  'subjectivity', 'assessments'])
39                  assessments = pattern_sentiment(text).assessments
40                  polarity, subjectivity = pattern_sentiment(text)
41                  return Sentiment(polarity, subjectivity, assessments)
42              else:
43                  Sentiment = namedtuple('Sentiment', ['polarity',
44                  'subjectivity'])
45                  return Sentiment(*pattern_sentiment(text))
46
47          def defaultfeatureextractor(words):
48              """Default feature extractor for the NaiveBayesAnalyzer."""
49              return dict(((word, True) for word in words))
50
51
52      class NaiveBayesAnalyzer(BaseSentimentAnalyzer):
53          """Naive Bayes analyzer that is trained on a dataset of movie
54          reviews.
55          Returns results as a named tuple of the form:
56          ``Sentiment(classification, ppos, pneg)``
```

```

57         :param callable featureextractor: Function that returns a dictionary
of
58         features, given a list of words.
59         """
60
61         kind = DISCRETE
62         #: Return type declaration
63         RETURN_TYPE = namedtuple('Sentiment', ['classification', 'p_pos',
64         'p_neg'])
65
66         def __init__(self, feature_extractor=default_feature_extractor):
67             super(NaiveBayesAnalyzer, self).__init__()
68             self._classifier = None
69             self.feature_extractor = feature_extractor
70
71         @requires_nltk_corpus
72         def train(self):
73             """Train the Naïve Bayes classifier on the movie review
74             corpus."""
75             super(NaiveBayesAnalyzer, self).train()
76             neg_ids = nltk.corpus.movie_reviews.fileids('neg')
77             pos_ids = nltk.corpus.movie_reviews.fileids('pos')
78             neg_feats = [(self.feature_extractor(
79                 nltk.corpus.movie_reviews.words(fileids=[f])), 'neg') for f
80                 in neg_ids]
81             pos_feats = [(self.feature_extractor(
82                 nltk.corpus.movie_reviews.words(fileids=[f])), 'pos') for f
83                 in pos_ids]
84             train_data = neg_feats + pos_feats
85             self._classifier =
86             nltk.classify.NaiveBayesClassifier.train(train_data)
87
88         def analyze(self, text):
89             """Return the sentiment as a named tuple of the form:
90             ``Sentiment(classification, p_pos, p_neg)``
91             """
92             # Lazily train the classifier
93             super(NaiveBayesAnalyzer, self).analyze(text)
94             tokens = word_tokenize(text, include_punc=False)
95             filtered = (t.lower() for t in tokens if len(t) >= 3)
96             feats = self.feature_extractor(filtered)
97             prob_dist = self._classifier.prob_classify(feats)
98             return self.RETURN_TYPE(
99                 classification=prob_dist.max(),
100                 p_pos=prob_dist.prob('pos'),
101                 p_neg=prob_dist.prob("neg")
102             )

```

1–4: TextBlob’s release in 2013 entered into a fruitful space for sentiment analysis. The rising wave of Web 2.0 and social media more generally encouraged internet users to enter unprecedented amounts of self-authored text into their machines, in turn finally providing data sets at the scale required for meaningful analysis. Development on TextBlob was at first fast and dense, although the pace of its development slowed over time. Loria added sentiment analysis to TextBlob in version 0.5.0, published on 10 August 2013. According to the changelog on GitHub, where TextBlob’s code is hosted, `en/sentiments.py` was most recently updated on 2 December 2017.

22

5–13: Python programs customarily include dependencies, or external programs which files require in order to operate, as “imports” at the beginning of code. Notably, line 8 imports NLTK; line 10 imports a function called `sentiment` from the file `en/__init__.py`, a lightly revised version of Pattern’s sentiment analysis implementation; and line 13 imports wrapper functions that define the basic form of both `PatternAnalyzer` and `NaiveBayesAnalyzer`. As noted in the essay, this reliance on Pattern’s sentiment analysis implementation, while labor-saving on the level of programming,

23

means that TextBlob simply copies wholesale Pattern's approach, which relies on the linear calculation of sentiment from a pre-scored corpus.

16: The remainder of `en/sentiments.py` defines two Python classes, one for each implementation. Loria begins with `PatternAnalyzer`, most likely due to its both conceptual and computational simplicity. `NaiveBayesAnalyzer`, by contrast, seems relegated to a secondary or experimental role.

24

28: Loria includes an `if/else` function in `PatternAnalyzer` to handle whether the user wants optionally to return "assessments," or the major lexical criteria upon which the function determines a sentence's polarity. Obscuring the assessments by default produces a cleaner, more "objective" reading on the command line at the expense of the more verbose output that would articulate the program's reasoning. Again, a reasonable choice from a user experience perspective, albeit one that encourages further interpretation of the machine's acts as somehow "neutral."

25

36, 41: In both the `if` and `else` parts of this function, the actual work of computing polarity occurs in `pattern_sentiment(text)`, which applies Pattern's sentiment analysis function, imported in line 10, to the given input `text`. While not included directly in `en/sentiments.py`, this function averages together assigned polarity scores drawn from a lexicon file. As noted in the essay, these polarity definitions were manually generated by De Smedt and Daelemans' teams in the process of training the dataset. We can only speculate on their exact identities, but this workflow more generally, which was customary in natural language processing work at the end, reminds us how ultimately judgment is a capacity of human brains, which must be in turn extracted, modeled, and mediated by the program.

26

44: In preparation for `NaiveBayesAnalyzer`, Loria includes a brief definition of a feature extractor. In NLP, a feature extractor parses input for major "features," or statistically significant lexical data. Again, simplification and compression permit Loria to perform this computational work in such a relatively parsimonious package. In particular, Loria's approach here eliminates contextual relations between words, information that later generations of sentiment analysis have attempted to consider in more depth. (For more on this, see footnote no. 6).

27

50: `NaiveBayesAnalyzer` uses machine learning functions to train its analysis on a dataset of movie reviews included as an example corpus in the NLTK code base. This corpus comprises two hundred movie reviews, half tagged positive and half tagged negative, assembled in 2004 by computer scientists Bo Pang and Lillian Lee as a tool for sentiment analysis projects. The corpus and its associated research papers are available at <https://www.cs.cornell.edu/people/pabo/movie-review-data/>. These reviews are imported in line 12. Movie reviews, alongside product reviews, are typical source corpora for sentiment analysis implementations, particularly in the time of TextBlob's most intensive development. They have the advantage of being both strongly opinionated by definition as well as accompanied often by a numerical value. As such, they offer a sort of "readymade" corpus for the work of sentiment analysis. The catch, of course, is that, as I discuss in the body of the essay, sentiment analysis not only mediates judgment, but also the *form* of those judgments. This makes tools derivate of the Pang and Lee corpus perhaps effective at ascertaining the sentiment from textual inputs that bear more than a passing similarity to movie reviews, but less effective at sentiment encoded in other kinds of linguistic forms.

28

61: Defines a subsequent feature extractor using the form previously defined in line 44. Most users will not vary their usage beyond `PatternAnalyzer`, making `NaiveBayesAnalyzer` more of an easter egg for those willing to read the documentation. This is a shame, given that I view TextBlob's most valuable contribution to the pedagogical space around sentiment analysis precisely the ease with which users can juxtapose the outputs of these two distinct implementations, and thereby understand the contingency of the machine's judgments. One might imagine a version of TextBlob that puts notions of machinic transparency first, that foregrounds both the assessments (easily editable in line 28), as well as offers users a choice between `PatternAnalyzer` and `NaiveBayesAnalyzer`.

29

66: Indicates that this function requires the user to download the NLTK corpus to their computer in order to run. The corpus is a small file all told, but perhaps this requirement is what shielded Loria away from making `NaiveBayesAnalyzer` a more readily available option. The download requirement places a small but meaningful load on the end user to have a working internet connection, introducing yet another dependency to the program's operations

30

— in this case, external networking technologies. That this fundamental dependency on the internet’s servers and the electrical grid more generally are required to download and install TextBlob in the first place would seem to make this an unnecessary point of contention, but one can understand how Loria would shy away from design choices that would take TextBlob further away from an all-in-one approach.

67–77: `NaiveBayesAnalyzer` comprises two major functions: a training and an analysis function. This first training function identifies which reviews in the Pang and Lee corpus are tagged positive or negative; extracts key textual features from each; combines them into a single variable named `train_data`; and then uses NLTK’s built-in algorithms to train a classifier. One might reasonably ask whether such an emphasis on positive and negative polarity is a useful heuristic for these reviews; what, for example, to make of mixed reviews, or ones that are neither strongly positive nor negative? This is a moment of disjuncture between the initial goals of Pang and Lee and similar natural language processing researchers and these programs’ implementations in, say, consumer-facing technologies. Reading Pang and Lee’s original paper, one comes to understand their project less as “attempting to design a movie-review-reader program” and more “attempting to solve specific problems in the computational evaluation of language, with movie reviews providing an effective starting point.” Indeed, there are several issues with the original approach that they themselves identify; one, for instance, is how to handle reviews that “turn,” so to speak, that pile up negative language only to reveal at the end that the user loved the film. (A not-unfamiliar move to those of us who love “bad” films.) For Pang and Lee, these are computational problems worthy of further investigation. However, derivative work — work for which Pang and Lee are dependencies, to use the language we have developed in this essay — strays from this initial pure-research vision. Even TextBlob, which still lingers at the edges of academic research, contains, as we have shown, many design decisions intended to emphasize the seeming neutrality and objectivity of its judgments. Tracing these histories through an analysis of the program’s code reveals the ease with which subjectivity slips into objectivity in the case of sentiment analysis.

31

79–93: These lines define `NaiveBayesAnalyzer`’s analysis function, where the sentiment analysis work actually happens. Lines 85–87 break down the text into individual features, which are then classified using a probability distribution function in line 88. Lines 89–92 print the results to the command line using the schema defined in line 52. Line 90 is particularly interesting in that it omits words of fewer than three characters. Loria is making a design choice here. Eliminating short words reduces the computational overhead, which for Naïve Bayes-based methods can be substantial, while theoretically leaving untouched all the more “meaningful” words. However, as work in distant reading has demonstrated, it is precisely in the shortest words of the English language — the articles, the conjunctions — that meaning is often expressed. Again, this turns to the problem of context in sentiment analysis: retaining these shorter words, which are often the glue holding together syntax, could allow for a more sophisticated reading of how meaning is constructed in relation. However, given TextBlob’s stated design purpose of being a simple, off-the-shelf tool for rapid sentiment analysis work, it’s understandable that Loria would eschew more complex approaches — some of which simply were unavailable to him in 2013 when he originally designed this program — in favor of more rapid response.

32

Acknowledgments

I would like to thank Mark Marino and Jeremy Douglass for their perceptive comments that have helped shape and expand this essay over its writing, as well as the anonymous reviewers for *DHQ*. Much of this essay was originally written during a fellowship with the Maryland Institute for Technology in the Humanities in the fall of 2020; I am particularly grateful to Ed Summers for his patience and generosity in teaching me the fundamentals of Python so that I might engage this project. Thanks are due also to Kari Kraus for suggesting TextBlob as a useful object for critical research on sentiment analysis, and to Alice Bi for being a thoughtful and generative interlocutor on the question of LLMs’ relationship to sentiment analysis more generally.

33

Notes

[1] I am here paraphrasing Kathleen Fitzpatrick’s famous definition of the digital humanities from her 2010 blog post on the *Chronicle of Higher Education*’s now-defunct *ProfHacker* blog. [Fitzpatrick 2012].

[2] I include the leading `en/` to distinguish from another file in the TextBlob code base named `sentiments.py`.

[3] For the code base of Pattern, see [De Smedt and Daelemans 2012]. For NLTK, see [Bird et al 2009].

[4] “Sentiment,” “affect,” “emotion,” and “mood” are often used interchangeably in sentiment analysis research, even as some scholars such as Liu seek to disambiguate them. For the purposes of my study, I define “sentiment” as an imputed judgment by sentiment analysis software about a given input — a polarity score, for instance. “Affect,” by contrast, is a theoretical aspect of human biopsychology that gives rise to emotions and moods. One might then use calculated sentiment to develop an affective schematic model, for instance.

[5] Here I am also thinking with N. Katherine Hayles’ work on “cognitive assemblages,” which she envisions as intelligences that exceed the human, animal, or machine. See [Hayles 2017].

[6] Subsequent and more sophisticated work in sentiment analysis has attempted to resolve the challenges of this single-word model with deeper considerations of the contexts between and across words. One such technique is Bidirectional Encoder Representations from Transformers (BERT), created in 2018 by researchers at Google. BERT-based sentiment analysis techniques take into consideration the words both before and after individual words when assigning values, and as such has more potential for both sentiment analysis and language prediction. BERT and derivative technologies are also foundational to later work on LLMs, which follow from but are conceptually and materially distinct from sentiment analysis. The downside of BERT is that it is computationally much more intensive than simpler tools such as TextBlob or even NLTK. Sheer processing power is easier to come by with each passing year — although it remains an open question as of this writing whether or not the “more power = better data” approach currently being tested by major tech companies such as OpenAI and Microsoft will pay off in anything other than ecosystem devastation — but still serves as a material limiting factor. See [Devlin et al. 2019] for further consideration of BERT.

[7] This .xml file has been imported wholesale, without any changes, from Pattern. This leads to some curious inconsistencies in the TextBlob code base: a header in the .xml file, for instance, claims that the reliability score, which specifies whether a value was hand-tagged or computationally inferred, takes either the value of 1.0 (for the former) or 0.7 (for the latter). However, many entries carry the unexplained value 0.9. Subjectivity scores, which putatively name a where a word falls on a subjective/objective axis, are calculated in TextBlob but not printed to the command line.

[8] Bayes’ Theorem takes the form $P(A|B) = (P(B|A)P(A)/P(B))$. In plain English, we might state this as “the probability of event A occurring given the truth of event B is equal to the probability of event B occurring given the truth of event A, multiplied by the probability of event A occurring in itself, all divided by the probability of event B occurring in itself.” Essentially, Bayes’ Theorem provides an approach to defining the *conditional* probability of an event’s occurrence. Bayes’ Theorem serves as a basis for many NLP tasks and classifier functions given its computational simplicity relative to the accuracy of its results.

[9] Here I am thinking also with Wendy Hui Kyong Chun’s work in her 2008 monograph *Programmed Visions: Software and Memory* on the cybernetic project of establishing precisely this equivalence. For Chun, early cybernetic research on machine intelligence that took for its putative model the human brain ineluctably doubled back, such that machine became the primary operative metaphors through which cognitive science came to understand the brain. See [Chun 2008].

Works Cited

Benjamin 2019 Benjamin, R. (2019) *Race after technology: Abolitionist tools for the new jim code* Polity Press.

Bird et al 2009 Bird, S., Loper, E., and Klein, E. (2009) *Natural language processing with Python*. Cambridge: O’Reilly Media.

Chun 2008 Chun, W.H. K. (2008) *Programmed visions: Software and memory*. Cambridge: MIT Press.

De Smedt and Daelemans 2012 De Smedt, T, and Daelemans, W. (2012) “Pattern for Python”, *Journal of Machine Learning Research* 13, pp. 2063–67.

Devlin et al. 2019 Devlin, J. et al. (2019) “BERT: Pre-training of deep bidirectional transformers for language understanding”, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Volume 1 (Long and Short Papers), pp. 4171–86. Minneapolis: Association for Computational Linguistics, 2019. Available at: <https://doi.org/10.18653/v1/N19-1423>.

Fitzpatrick 2012 Fitzpatrick, K. (2012) “The humanities, done digitally”, *Debates in the Digital Humanities*, ed. Matthew K. Gold. Minneapolis: University of Minnesota Press, pp. 12–15.

Hayles 2017 Hayles, N.K. (2017) *Unthought: The power of the cognitive nonconscious*. Chicago: University of Chicago Press.

Hayles 2018 Hayles, N.K. (2018) “Human and machine cultures of reading: A cognitive-assemblage approach”, *PMLA* 133.5, pp. 1225–42.

Jockers 2014 Jockers, M. (2014) “syuzhet”. *Github.com*. <https://github.com/mjockers/syuzhet>. (Accessed 22 Oct 2025).

Liu 2015 Liu, B. (2015) *Sentiment analysis: Mining ppinions, sentiments, and emotions*. New York: Cambridge University Press.

Loria 2020 Loria 2020 Loria, Steven. “TextBlob: Simplified Text Processing TextBlob 0.16.0 Documentation.” <https://textblob.readthedocs.io/en/dev/>, 2020, accessed 1 May 2020.

Manning et al 2008 Manning, C.D, Raghavan, P. and Schütze, H. (2008) *Introduction to information retrieval*. New York: Cambridge University Press.

Matviyenko 2025 Matviyenko, S. (2025) "On governance, blackboxing, measure, body, affect, and apps: A conversation with Patricia Ticineto Clough and Alexander R. Galloway", *The Fibreculture Journal*, no. 25.

Noble 2018 Noble, S. U. (2018) *Algorithms of oppression: How search engines reinforce racism*. NYU Press.

Pang and Lee 2005 Pang, B. and Lee, L. (2005) "Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales." In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics - ACL '05*, 115–24. Ann Arbor, Michigan: Association for Computational Linguistics, 2005.

Sedgwick and Frank 1995 Sedgwick, E. K. and Frank, A (1995) "Shame in the cybernetic fold: Reading Silvan Tomkin", *Critical Inquiry* 21.2, pp. 496–522.

Stone 1966 Stone, P.J. *The general inquirer: A computer approach to content analysis*. Cambridge: MIT Press.

Tomkins 2008 Tomkins, S.S. (2008) *Affect imagery consciousness: The complete edition*. New York: Springer Publishing.

Turing 1950 Turing, A. (1950) "Computing machinery and intelligence", *Mind* 59.236, pp. 433–60.

Recommendations

DHQ is testing out three new article recommendation methods! Please explore the links below to find articles that are related in different ways to the one you just read. We are interested in how these methods work for readers—if you would like to share feedback with us, please complete our short evaluation survey. You can also visit our documentation for these recommendation methods to learn more.

SPECTER Recommendations

Below are article recommendations generated by the SPECTER model:

1. Sentiment Analysis in Literary Studies. A Critical Survey, 2023, Simone Rebora, Johannes Gutenberg University Mainz
2. Sentiment Analysis: Limits and Progress of the Syuzhet Package and Its Lexicons, 2022, Hoyeol Kim, Texas A&M University
3. Mining Embodied Emotions: A Comparative Analysis of Sentiment and Emotion in Dutch Texts, 1600-1800., 2018, Inger Leemans, Faculty of Humanities, Vrije Universiteit Amsterdam, The Netherlands; Janneke M. van der Zwaan, Netherlands eScience Center, Amsterdam, The Netherlands; Isa Maks, Faculty of Humanities, Vrije Universiteit Amsterdam, The Netherlands; Erika Kuijpers, Faculty of Humanities, Vrije Universiteit, Amsterdam, The Netherlands; Kristine Steenbergh, Faculty of Humanities, Vrije Universiteit, Amsterdam, The Netherlands
4. Introduction: Situating Critical Code Studies in the Digital Humanities, 2023, Mark C. Marino, University of Southern California; Jeremy Douglass, University of California, Santa Barbara
5. Tracing "Toxicity" Through Code: Towards a Method of Explainability and Interpretability in Software, 2023, David M. Berry, University of Sussex

DHQ Keyword Recommendations

Below are article recommendations generated by DHQ Keywords:

1. Computational art Explorations of Linguistic Possibility Spaces: comparative translingual close readings of Daniel C. Howe's *Automatypograph* and *Radical of the Vertical Heart* ↑, 2023, John Cayley, Brown University
2. Digital Oulipo: Programming Potential Literature, 2017, Natalie Berkman, Princeton University
3. Introduction: Reading Code in the Age of AI, 2026, Jeremy Douglass, University of California, Santa Barbara; Mark C. Marino, University of Southern California
4. cut to fit the toolspun course, 2013, Nick Montfort, MIT; Stephanie Strickland, Independent scholar
5. Defactoring Pace of Change, 2026, Matt Burton, University of Pittsburgh; Joris van Zundert, Huygens Institute of the Royal Netherlands Academy of Arts and Sciences

TF-IDF Recommendations

Below are article recommendations generated by the TF-IDF Model:

1. Sentiment Analysis in Literary Studies. A Critical Survey, 2023, Simone Rebora, Johannes Gutenberg University Mainz
2. Sentiment Analysis: Limits and Progress of the Syuzhet Package and Its Lexicons, 2022, Hoyeol Kim, Texas A&M University
3. Unraveling reported dreams with text analytics, 2018, Iris Hendrickx, Centre for Language Studies, Radboud University, Nijmegen, The Netherlands; Centre for Language and Speech Technology, Radboud University,

Nijmegen, The Netherlands; Louis Onrust, Centre for Language Studies, Radboud University, Nijmegen, The Netherlands; Florian Kunneman, Centre for Language Studies, Radboud University, Nijmegen, The Netherlands; Ali Hürriyetoğlu, Centre for Language Studies, Radboud University, Nijmegen, The Netherlands; Wessel Stoop, Centre for Language and Speech Technology, Radboud University, Nijmegen, The Netherlands; Antal van den Bosch, Meertens Institute, Amsterdam, The Netherlands

4. Automatic Identification of Rhetorical Elements in classical Arabic Poetry, 2023, Heyam Abd Alhadi, University of Haifa; Ali Ahmad Hussein, University of Haifa; Tsvi Kuflik, University of Haifa
5. Mining Embodied Emotions: A Comparative Analysis of Sentiment and Emotion in Dutch Texts, 1600-1800., 2018, Inger Leemans, Faculty of Humanities, Vrije Universiteit Amsterdam, The Netherlands; Janneke M. van der Zwaan, Netherlands eScience Center, Amsterdam, The Netherlands; Isa Maks, Faculty of Humanities, Vrije Universiteit Amsterdam, The Netherlands; Erika Kuijpers, Faculty of Humanities, Vrije Universiteit, Amsterdam, The Netherlands; Kristine Steenbergh, Faculty of Humanities, Vrije Universiteit, Amsterdam, The Netherlands



This work is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License.