Lecture 8: POS Tagging & Sentiment Analysis

PSYC 51.07: Models of language and communication

Jeremy R. Manning
Dartmouth College
Winter 2026

Learning objectives

By the end of this lecture, you will

1. Understand part-of-speech (POS) tagging and its applications
2. Explore how neural networks learn grammatical structure
3. Apply sentiment analysis to real-world text
4. Fine-tune pre-trained models for domain-specific tasks
5. Critically evaluate whether models "understand" language

Central questions

- Can statistical patterns capture grammatical knowledge?
- What does it mean for a model to "understand" emotion?

Part-of-speech (POS) tagging

What is POS tagging?

- Assigning grammatical category to each word
- Categories: noun, verb, adjective, adverb, pronoun, preposition, etc.
- A fundamental NLP task

Example

```
The    cat    sat    on    the    mat
DET    NOUN   VERB   ADP   DET    NOUN
```

Why it matters

- Disambiguation: "book" as noun vs. verb
- Syntax parsing and understanding
- Information extraction, machine translation

POS tagsets: Universal vs. fine-grained

Universal POS (17 tags):

- ADJ, ADV, ADP, AUX
- CONJ, DET, NOUN, NUM
- PRON, PROPN, VERB, ...

Penn Treebank (45+ tags):

- NN/NNS/NNP/NNPS (nouns)
- VB/VBD/VBG/VBN/VBP/VBZ (verbs)
- Much finer distinctions!

Trade-off

Simplicity vs. linguistic detail

Context matters: Ambiguous words

Sentence	Word	POS	Explanation
"I read a **book**"	book	NOUN	Object being read
"Please **book** a table"	book	VERB	Action of reserving
"She runs **fast**"	fast	ADV	Modifies "runs"
"I will **fast** today"	fast	VERB	Action of not eating
"Please **close** the door"	close	VERB	Action
"Stay **close** to me"	close	ADV	Modifies position

Key insight

Context determines POS! Models must look at surrounding words.

spaCy resolves ambiguity using context

Code example

```python
import spacy
nlp = spacy.load("en_core_web_sm")

sentences = [
    "I need to book a flight",       # book = VERB
    "I'm reading a great book",      # book = NOUN
    "The record was broken",         # record = NOUN
    "Please record the meeting",     # record = VERB
]

for sent in sentences:
    doc = nlp(sent)
    for token in doc:
        if token.text.lower() in ["book", "record"]:
            print(f'"{sent}"')
            print(f'  "{token.text}" → {token.pos_}
```

continued...

spaCy resolves ambiguity using context

```
print()
```

The model uses surrounding words to disambiguate

`"to book"` vs `"a book"` — context is everything!

...continued

POS tagging with spaCy

```python
import spacy
nlp = spacy.load("en_core_web_sm")

sentence = "She will book the meeting room tomorrow"
doc = nlp(sentence)

print(f"{'Word':<12} {'POS':<8} {'Tag':<8} {'Explanation'}")
for token in doc:
    print(f"{token.text:<12} {token.pos_:<8} {token.tag_:<8} {spacy.explain(token.pos_)}")

# Output:
# She          PRON     PRP      pronoun, personal
# will         AUX      MD       verb, modal auxiliary
```

Notice

"book" correctly identified as VERB!

How do POS taggers work?

Traditional approaches (pre-neural):

- Rule-based: Hand-crafted grammar rules
- Hidden Markov Models (HMMs): Probabilistic sequences
- Conditional Random Fields (CRFs): Structured

Modern neural approaches:

- Recurrent Neural Networks (RNNs/LSTMs): Process sequences
- Transformers (BERT, etc.): Bidirectional context
- Fine-tune pre-trained models on POS data

Neural networks and grammar

Can neural networks learn syntactic structure?

Classic test: Subject-verb agreement (Linzen et al., 2016)

The key → to the cabinets → is/are? → ...

Challenge: Distractor nouns between subject and verb

The challenge

- Model must identify "key" (singular) as subject
- Ignore "cabinets" (plural distractor)
- Predict correct verb form "is" (not "are")

Finding

LSTMs can learn this! But they struggle with complex cases.

Reference: Linzen, Dupoux, & Goldberg (2016). *TACL*.

BLiMP: Testing linguistic knowledge

BLiMP Dataset

- 67,000 minimal pairs across 67 paradigms
- Tests syntax, semantics, morphology
- **Task:** Model assigns higher P to acceptable sentence

Results

Transformers (BERT, GPT-2) score 70-85%, but not perfect!

Acceptable	Unacceptable
"Who did you see?"	"Who did you saw?"
"I think that she left"	"I think that she leave"

Reference: Warstadt et al. (2020). *TACL.*

Token classification with HuggingFace

```python
from transformers import pipeline
pos_tagger = pipeline("token-classification",
    model="vblagoje/bert-english-uncased-finetuned-pos",
    aggregation_strategy="simple")

sentence = "Apple Inc. is looking at buying a UK startup"
results = pos_tagger(sentence)

for result in results:
    print(f"{result['word']:<15} {result['entity_group']:<8} ({result['score']:.3f})")
# Apple           PROPN    (0.998)
# Inc.            PROPN    (0.995)
```

Further reading

[HuggingFace Chapter 7.2: Token Classification]

Discussion: Do models "understand" grammar?

Perspectives to consider

1. **Chomsky's view:** Grammar requires innate, symbolic rules
   - Can statistical patterns truly capture grammatical knowledge?
2. **Emergentist view:** Grammar emerges from usage patterns
   - Maybe neural networks learn similarly to humans?
3. **Functional perspective:** If it works, does it matter?
   - Models perform well on tasks—is that "understanding"?
4. **Limitations:** Models still fail on edge cases
   - What does this tell us about their knowledge?

Your thoughts?

Is pattern matching sufficient for grammatical competence?

Sentiment analysis determines emotional tone of text

Applications:

- Social media monitoring
- Customer feedback analysis
- Market research
- Review analysis
- Political tracking

Granularity levels:

- Binary: positive/negative
- Ternary: positive/negative/neutral
- Fine-grained: 1-5 stars
- Continuous: sentiment score

Sentiment analysis challenges

1. Sarcasm and irony

- "Oh great, another meeting" (negative, despite "great")
- "This is the best movie I've ever fallen asleep to" (negative!)

2. Context-dependent sentiment

- "This movie is sick!" (positive in slang, negative literally)
- "The book was long" (neutral? negative?)

3. Mixed sentiment

- "Great food but terrible service" (both positive and negative)
- Aspect-based sentiment: food=positive, service=negative

4. Negation

- "not good" vs. "good"
- "I don't dislike it" (double negative = positive?)

5. Domain specificity

- "Explosive growth" (positive in business, negative in safety)
- Different domains have different sentiment patterns

Sentiment challenges: Code examples

Testing how models handle tricky cases

```
from transformers import pipeline
sentiment = pipeline("sentiment-analysis")

tricky_cases = [
    ("Oh great, another Monday meeting", "NEGATIVE"),        # Sarcasm
    ("This movie is not bad at all", "POSITIVE"),            # Negation
    ("I don't dislike this product", "POSITIVE"),            # Double negative
    ("Great camera but terrible battery life", "MIXED"),    # Mixed sentiment
]

for text, expected in tricky_cases:
    result = sentiment(text)[0]
    print(f"{text}")
    print(f"  Model: {result['label']} ({result['score']:.3f}) |
```

Key finding

Sentiment lexicons: words with predefined sentiment scores

Popular lexicons:

- **AFINN:** -5 to +5 ratings
- **SentiWordNet:** pos/neg/neutral
- **VADER:** Social media focused

Limitations: Ignores context

Example (AFINN):

Word	Score
great	+3
good	+3
hate	-3
terrible	-3

Lexicon-based sentiment analysis

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

texts = ["I love this product! It's amazing!", "This is the worst experience ever.",
         "It's okay, nothing special.", "Great food but terrible service!"]

for text in texts:
    scores = analyzer.polarity_scores(text)
    print(f"{text}")
    print(f"  Pos: {scores['pos']:.2f}, Neg: {scores['neg']:.2f}, Compound: {scores['compound']:.3f}\n")
# VADER handles emoji and punctuation! "Great!!!" scores higher than "Great"
```

Modern approach: Pre-trained models

Neural network advantages

- Learn context-dependent representations
- Capture word order and negation
- Handle sarcasm better (though still imperfect)
- Transfer learning: pre-train on large corpus, fine-tune for sentiment

Input Text → Pre-trained Model (BERT) → Classification Head → Sentiment Label

Typical neural sentiment analysis architecture

Training

Fine-tune on labeled sentiment data (IMDb, Amazon reviews, etc.)

VADER vs Neural: Head-to-head comparison

Testing both approaches on the same examples

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from transformers import pipeline

vader = SentimentIntensityAnalyzer()
neural = pipeline("sentiment-analysis")

test_cases = [
    "I absolutely love this product!",
    "This is not what I expected, but in a good way",
    "The movie was so bad it was actually hilarious",
]

for text in test_cases:
    v_score = vader.polarity_scores(text)['compound']
    n_result = neural(text)[0]
```

continued...

VADER vs Neural: Head-to-head comparison

```
    print(f"Text: {text}")
    print(f"  VADER: {v_score:+.3f} ({'POS' if v_score > 0 else 'NEG'})")
    print(f"  Neural: {n_result['label']} ({n_result['score']:.3f})\n")
```

Key finding

Neural models handle nuance better, but VADER is faster and interpretable.

...continued

Sentiment analysis with HuggingFace

```python
from transformers import pipeline
sentiment_analyzer = pipeline("sentiment-analysis")

texts = ["I love this product! It's amazing!", "This is the worst experience ever.",
         "It's okay, nothing special.", "I don't hate it, but I don't love it either."]

for text in texts:
    result = sentiment_analyzer(text)[0]
    print(f"{text}")
    print(f"  {result['label']}, Confidence: {result['score']:.3f}\n")
# "I love this product!" → POSITIVE (0.999)
```

Further reading

[HuggingFace Chapter 1.2: NLP Tasks]()

Domain-specific sentiment models

Problem

General models miss domain-specific language

Solution

Fine-tune on domain-specific data!

Why it works: Domain-specific vocabulary ("bullish" in finance = positive), different sentiment expressions, adapted conventions.

Domain	Model	Data
Medical	BioBERT	Patient feedback
Twitter	TwitterBERT	Social posts
Products	RoBERTa	Amazon reviews
Movies	BERT	IMDb reviews

Comparing general vs. domain-specific

```python
from transformers import pipeline
general = pipeline("sentiment-analysis")
financial = pipeline("sentiment-analysis", model="ProsusAI/finbert")

texts = ["The company's earnings exceeded expectations",
         "Revenue declined but margins improved", "Stock prices plummeted"]

for text in texts:
    gen = general(text)[0]
    fin = financial(text)[0]
    print(f"{text}")
    print(f"  General: {gen['label']} ({gen['score']:.3f}) | "
          f"Financial: {fin['label']} ({fin['score']:.3f})\n")
# Financial model often more accurate for finance text!
```

Fine-tuning for sentiment analysis

Process overview

1. **Start with pre-trained model** (BERT, RoBERTa) — already knows language
2. **Prepare labeled dataset** — text + sentiment labels (pos/neg/neutral)
3. **Add classification head** — dense layer outputting class probabilities
4. **Fine-tune on sentiment data** — much faster than training from scratch!
5. **Evaluate** — accuracy, precision, recall, F1-score

Further reading

HuggingFace Chapter 3.2: Processing Data for Fine-tuning

Fine-tuning example (simplified)

```python
from transformers import AutoModelForSequenceClassification, Trainer, TrainingArguments

# 1. Load pre-trained model
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)

# 2. Define training arguments
training_args = TrainingArguments(output_dir="./results", num_train_epochs=3,
    per_device_train_batch_size=16, evaluation_strategy="epoch")

# 3. Create Trainer and train (train_dataset, eval_dataset prepared separately)
trainer = Trainer(model=model, args=training_args,
```

Evaluation metrics for sentiment analysis

Beyond accuracy

- **Precision:** Of predicted positives, how many are truly positive?

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall:** Of actual positives, how many did we catch?

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score:** Harmonic mean of precision and recall

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Why not just accuracy?

- Imbalanced datasets (e.g., 90% positive reviews)
- Different costs for false positives vs. false negatives
- F1 gives balanced view of model performance

Confusion matrix example

	Predicted Positive	Predicted Negative
Actual Positive	TP = 90	FN = 10
Actual	[illegible]	TN =

Metrics

- **Accuracy** = (90 + 95) / 200 = 92%
- **Precision** = 90 / 95 = 95%
- **Recall** = 90 / 100 = 90%
- **F1** = 2 × (0.95 × 0.90) / (0.95 + 0.90) = 92%

Aspect-based sentiment analysis

Problem

Reviews often mention multiple aspects with different sentiments

Example

"The **food was delicious** but the **service was terrible**. The **atmosphere was okay**."

- Food: Positive
- Service: Negative
- Atmosphere: Neutral

Applications:

- Restaurant reviews: food, service, ambiance, price
- Product reviews: quality, price, shipping, customer service
- Hotel reviews: room, location, staff, cleanliness

Key insight

More nuanced than overall sentiment! Provides actionable insights.

Aspect-based sentiment: Worked example

```python
review = """The pasta was incredible - best I've had!
However, we waited 45 min which was frustrating.
Ambiance was nice but loud.
Prices reasonable."""

aspects = {
    "food": ["pasta", "incredible"],    # POSITIVE
    "service": ["waited", "frustrating"], # NEGATIVE
    "ambiance": ["nice", "loud"], # MIXED
    "price": ["reasonable"] # POSITIVE
}
```

Aspect	Sentiment
Food	POSITIVE
Service	NEGATIVE
Ambiance	MIXED
Price	POSITIVE

Actionable

Focus on improving wait times!

Real-world application: Product review analysis

Business value

- Identify product strengths and weaknesses
- Track sentiment trends over time
- Compare against competitors
- Prioritize product improvements

Collect Reviews → Extract Aspects → Analyze Sentiment → Generate Insights

Product review analysis pipeline

Example insights

- **Product A:** Stable positive sentiment
- **Product B:** Declining sentiment → investigate quality issues!

Hands-on exercise

Try this yourself

1. **Collect data:**
   - Scrape product reviews (Amazon, Yelp)
   - Or use public dataset (IMDb, Twitter)
2. **Compare approaches:**
   - Lexicon-based (VADER)
   - General pre-trained model (HuggingFace pipeline)
   - Domain-specific model (if available)
3. **Analyze results:**
   - Where do models disagree?
   - Which handles sarcasm better?
   - Which is most accurate for your domain?
4. **Bonus:** Fine-tune a model on your specific dataset!

Discussion: Understanding emotion

Philosophical questions

1. **Can models "feel" sentiment?**
    - They predict labels, but do they understand emotion?
2. **Is sentiment objective or subjective?**
    - Different annotators may disagree on sentiment
    - How do models handle ambiguity?
3. **Cultural and linguistic variation:**
    - Sentiment expressions vary across cultures
    - Can models capture these nuances?
4. **Ethical considerations:**
    - Automated sentiment analysis in hiring, lending...
    - Risks of bias and discrimination
    - Should we trust model judgments?

Week 2: Each step builds on the previous one

Data Cleaning → Tokenization → POS Tagging → Sentiment Analysis

The NLP pipeline

- **Data cleaning:** Remove noise
- **Tokenization:** Break into units
- **POS tagging:** Grammar structure
- **Sentiment:** Emotional meaning

Assignment 2: SPAM email classifier

Your task

Build a classifier to detect spam emails

Apply this week's concepts:

- **Data cleaning:** Remove HTML tags, normalize text
- **Tokenization:** Try different tokenizers (word, subword)
- **Features:** Extract useful signals (POS patterns, sentiment?)
- **Classification:** Train a model to identify spam

Think about:

- What makes spam different from legitimate emails?
- How does preprocessing affect accuracy?
- Can you use sentiment as a feature?
- What about POS patterns? (e.g., spam has more imperatives?)

Link

Assignment 2: SPAM classifier

Key takeaways

What we learned

1. **POS tagging reveals grammatical structure** — Neural nets learn syntax, but do they "understand" it?
2. **Sentiment analysis extracts emotional meaning** — Lexicons → neural networks; domain fine-tuning helps
3. **Statistical learning powers modern NLP** — Models learn patterns without explicit rules
4. **Context is crucial** — Words are ambiguous; Transformers excel at capturing context
5. **Critical thinking matters** — Question what "understanding" means; be aware of biases

Primary references

POS tagging and syntax:

- Linzen, Dupoux, & Goldberg (2016). Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *TACL*.
- Warstadt et al. (2020). BLiMP: The Benchmark of Linguistic Minimal Pairs. *TACL*.

Sentiment analysis:

- Pang & Lee (2008). Opinion Mining and Sentiment Analysis. *Foundations and Trends in IR*.
- Hutto & Gilbert (2014). VADER: A

HuggingFace resources:

- Chapter 1.2: NLP Tasks with Pipeline
- Chapter 3.2: Processing Data for Fine-tuning
- Chapter 7.2: Token Classification

Looking ahead: Weeks 3-4

Next topics

- Dimensionality reduction (PCA, UMAP)
- Bag-of-words and TF-IDF
- Word embeddings (Word2Vec, GloVe)
- Distributional semantics

Central question

"You shall know a word by the company it keeps"

How can we represent word *meaning* computationally?

Prepare by

- Completing Assignment 2
- Thinking about: What is "meaning"? How would you define it?
- Exploring: Vector representations and semantic similarity

Additional resources

Tools and libraries:

- spaCy: https://spacy.io/
- HuggingFace Transformers: https://huggingface.co/docs/transformers/
- VADER Sentiment: https://github.com/cjhutto/vaderSentiment

Datasets:

- IMDb Movie Reviews: https://ai.stanford.edu/~amaas
- Amazon Product Reviews: https://registry.opendata.aws/a
- Stanford Sentiment Treebank: https://nlp.stanford.edu/sentim

Interactive demos

Questions? Want to chat more?

Email me

Join our Discord

Come to office hours

Congratulations!

Week 2 complete! See you in Week 3!