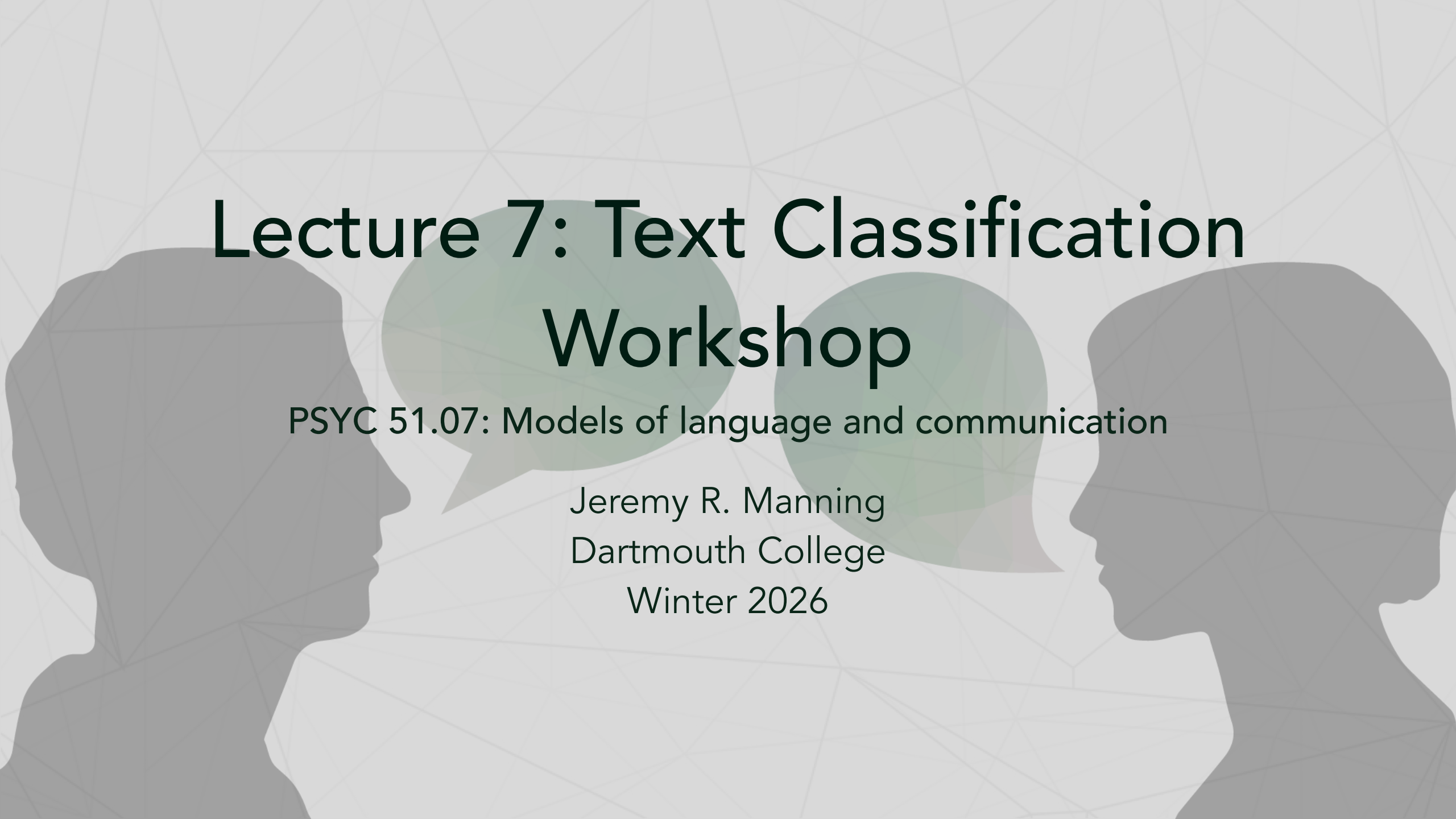


Lecture 7: Text Classification Workshop

PSYC 51.07: Models of language and communication

Jeremy R. Manning
Dartmouth College
Winter 2026

Learning Objectives

By the end of this session, you will

1. Build text classifiers from scratch using scikit-learn
2. Understand different text representation methods (BoW, TF-IDF, embeddings)
3. Compare Naive Bayes, Logistic Regression, and Neural approaches
4. Evaluate classifier performance using appropriate metrics
5. Debug common issues in text classification pipelines

Workshop format

Hands-on coding with the 20 Newsgroups dataset

Workshop Overview

Today's agenda

1. **Part 1:** Loading and exploring real data
2. **Part 2:** Feature engineering for text (BoW, TF-IDF)
3. **Part 3:** Building classifiers (Naive Bayes, Logistic Regression, Neural Networks)
4. **Part 4:** Model comparison and analysis
5. **Part 5:** Error analysis and improvements
6. **Part 6:** Real-world considerations (class imbalance)

Companion notebook

`xhour_classification_demo.ipynb`

Part 1: The 20 Newsgroups Dataset

A classic text classification benchmark

- Posts from 20 different newsgroups
- ~20,000 documents total
- Good for learning classification fundamentals

Today's subset (4 categories)

- `sci.space` — Science discussions about space
- `rec.sport.hockey` — Sports discussions about hockey
- `talk.politics.misc` — Political discussions
- `comp.graphics` — Computer graphics

Why these?

Relatively distinct topics for easier learning.

Loading the Data

```
from sklearn.datasets import fetch_20newsgroups

categories = [
    'sci.space',
    'rec.sport.hockey',
    'talk.politics.misc',
    'comp.graphics'
]

train_data = fetch_20newsgroups(
    subset='train',
    categories=categories,
    shuffle=True,
    random_state=42,
```

Always explore your data before building models

Key questions to ask

1. How many documents per category?
2. What do the documents look like?
3. What words/phrases might be good indicators?
4. Are there categories that might be hard to distinguish?

Exploring the Data: Concrete Example

```python
import pandas as pd
from collections import Counter

# Check class distribution
print("Documents per category:")
for i, name in enumerate(train_data.target_names):
    count = (train_data.target == i).sum()
    print(f"  {name}: {count}")
# sci.space: 593, rec.sport.hockey: 600, talk.politics.misc: 465, comp.graphics: 584

# Look at a sample document
idx = [i for i, t in enumerate(train_data.target) if t == 0][0]
print(train_data.data[idx][:500])
```

Notice

Classes are roughly balanced (good!), but `talk.politics.misc` has fewer examples.

Convert text to numbers for machine learning

Three approaches today

1. **Bag of Words (BoW):** Count word frequencies
2. **TF-IDF:** Weight by document frequency
3. **Dense embeddings:** (Preview for future lectures)

Bag of Words: count how many times each word appears

```python
from sklearn.feature_extraction.text import CountVectorizer

bow_vectorizer = CountVectorizer(
    max_features=5000,    # Keep only top 5000 words
    min_df=2,             # Word must appear in at least 2 docs
    max_df=0.8,           # Word must appear in <80% of docs
    stop_words='english' # Remove common words
)

X_train_bow = bow_vectorizer.fit_transform(train_data.data)
```

Result: Sparse matrix of word counts

Bag of Words: Limitations

What BoW captures

- Word presence/frequency
- Vocabulary overlap between documents

What BoW ignores

- Word order ("not good" vs "good not")
- Semantics ("great" vs "excellent")
- Context

Key insight

Common words dominate but are often uninformative!

BoW vectors are sparse: mostly zeros

Example

```python
from sklearn.feature_extraction.text import CountVectorizer

docs = ["NASA launches rocket to Mars", "Hockey game ends in overtime",
        "NASA discovers water on Mars"]

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(docs)

print("Vocabulary:", vectorizer.vocabulary_)
# {'nasa': 5, 'launches': 4, 'rocket': 7, 'to': 8, 'mars': 6, ...}

print("\nDocument 1:", X[0].toarray())
# [0 0 0 0 1 1 1 1 1 0 0 0 0] ← counts for each word
```

Observation

Most entries are 0 (sparse!). Documents share "mars" and "nasa".

Method 2: TF-IDF

Term Frequency-Inverse Document Frequency

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

where:

- $\text{TF}(t, d)$ = frequency of term t in document d
- $\text{IDF}(t) = \log \frac{N}{\text{df}(t)}$ = inverse document frequency

Intuition

Downweight common words, upweight rare informative words!

TF-IDF in Practice

```python
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(
    max_features=5000,
    min_df=2,
    max_df=0.8,
    stop_words='english',
    use_idf=True,
    sublinear_tf=True  # Use log scaling for term frequency
)

X_train_tfidf = tfidf_vectorizer.fit_transform(train_data.data)
```

Result: Sparse matrix of TF-IDF scores

BoW vs TF-IDF Comparison

Same document, different representations

Word	BoW Count	TF-IDF Score
"the"	15	0.02 (low — common everywhere)
"nasa"	3	0.45 (high — rare, informative)
"space"	5	0.38 (moderate — distinctive)

Key insight

TF-IDF identifies the truly distinctive terms!

Three classifier approaches to compare

Today's candidates

1. **Naive Bayes:** Fast, probabilistic, good baseline
2. **Logistic Regression:** Linear, interpretable, often best
3. **Neural Network:** Flexible, can learn complex patterns

Classifier 1: Naive Bayes

Bayes' theorem with independence assumption

$$P(y|x) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

- **Why "naive"?** Assumes features are independent (they're not!)
- **Why does it work?** Despite the wrong assumption, it often performs well for text.

```
1 from sklearn.naive_bayes import MultinomialNB
2
3 nb = MultinomialNB()
```

Classifier 2: Logistic Regression

Learns weights for each feature

$$P(y = k|x) = \frac{e^{w_k^T x}}{\sum_j e^{w_j^T x}}$$

Advantages

- Interpretable weights (which words matter?)
- Often outperforms Naive Bayes
- Fast training and prediction

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(max_iter=1000, C=1.0)
lr.fit(X_train_tfidf, train_data.target)
```

Logistic regression weights reveal which words matter

Top positive features per category

Category	Top Positive Features
sci.space	nasa, orbit, shuttle, moon, launch
rec.sport.hockey	hockey, nhl, team, game, play
talk.politics.misc	government, president, tax, policy
comp.graphics	image, graphics, 3d, rendering

Key insight

The model learns what we'd expect! Interpretability matters.

Classifier 3: Simple Neural Network

Feedforward architecture

```python
import torch.nn as nn

class TextClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim // 2)
        self.fc3 = nn.Linear(hidden_dim // 2, output_dim)
        self.dropout = nn.Dropout(0.3)
        self.relu = nn.ReLU()
```

Linear models are competitive with BoW features

Results on 20 Newsgroups

Model	Accuracy
Naive Bayes (BoW)	~85%
Naive Bayes (TF-IDF)	~87%
Logistic Regression	~90%
Neural Network	~89%

Key insight

Neural networks shine with richer representations (embeddings), not BoW.

Accuracy alone is not enough

Better metrics

- **Precision:** Of predicted positives, how many are truly positive?
- **Recall:** Of actual positives, how many did we catch?
- **F1-Score:** Harmonic mean: $F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

Use precision/recall/F1 when...

Datasets are imbalanced or different errors have different costs.

Confusion Matrix

Visual representation of classifier errors

	Predicted A	Predicted B	Predicted C	Predicted D
Actual A	85	2	3	0
Actual B	1	92	2	5
Actual C	4	3	88	5
Actual D	0	8	2	90

Interpretation

Diagonal = correct predictions. Off-diagonal = errors.

Error analysis: critical but often skipped

The process

1. Find misclassified examples
2. Look for patterns
3. Understand why the model failed
4. Use insights to improve

Common culprits

Mixed topics, short documents, unusual vocabulary

Error Analysis: Concrete Example

```
# Find misclassified examples
y_pred = lr.predict(X_test_tfidf)
errors = np.where(y_pred ≠ test_data.
idx = errors[0]

print(f"True:
{test_data.target_names[test_data.targ
print(f"Pred:
{test_data.target_names[y_pred[idx]]}"
print(test_data.data[idx][:300])
```

Example output

True: sci.space **Predicted:** comp.graphics

"I'm working on a 3D visualization of the solar system for my graphics project..."

Insight

Document mentions both graphics AND space. Model reasonably confused!

Common Error Patterns

Why do classifiers fail?

1. **Ambiguous content:** Document mentions multiple topics
2. **Limited context:** Very short documents
3. **Domain shift:** Test data differs from training
4. **Rare vocabulary:** Important words not in training

Solution ideas

- Better preprocessing
- More features (bigrams, trigrams)
- Domain-specific fine-tuning

Class imbalance makes models predict the majority class

Solutions

1. **Class weights:** Penalize minority errors more
2. **Oversampling:** Duplicate minority examples
3. **Undersampling:** Remove majority examples
4. **SMOTE:** Generate synthetic minority examples

```
lr_balanced = LogisticRegression(
    class_weight='balanced'  # Automatically adjust weights
)
```

Discussion Questions

Think about these

1. **BoW vs TF-IDF:** When would you prefer one over the other?
2. **Linear vs Neural:** Why didn't the neural network significantly outperform logistic regression?
3. **Feature Engineering:** How important was feature engineering compared to model choice?
4. **Scalability:** Which approach scales best to millions of documents?
5. **Interpretability:** Which models are most interpretable? Why does it matter?

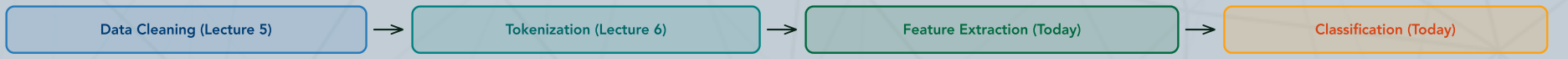
Key Takeaways

Remember

1. **Good features matter more than complex models** (for many tasks)
2. **TF-IDF usually beats raw BoW** for text classification
3. **Linear models are competitive** with neural networks on bag-of-words
4. **Always examine errors** to understand model behavior
5. **Consider class imbalance** and adjust accordingly

Connection to Course Themes

This week's pipeline



Next lecture

POS Tagging & Sentiment Analysis — How do these building blocks combine for real NLP applications?

Hands-On Exercise

Open the companion notebook

`xhour_classification_demo.ipynb`

Steps

1. Load the 20 Newsgroups dataset
2. Experiment with different vectorizers
3. Train multiple classifiers
4. Analyze errors and improve
5. Try your own text examples!

Goal

Build intuition for text classification

Additional Resources

Libraries

- scikit-learn
- PyTorch

HuggingFace

- Chapter 1: Transformer Models

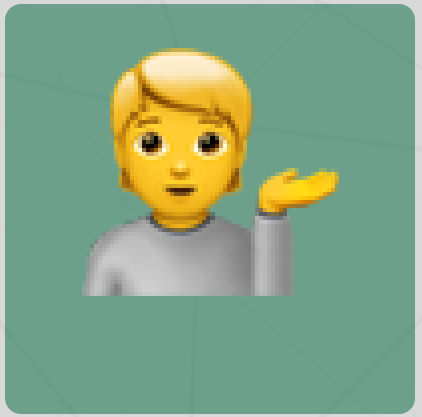
Datasets

- **20 Newsgroups:** Classic benchmark
- **IMDb Reviews:** Sentiment classification
- **AG News:** News categorization

Questions? Want to chat more?

Email me

Join our Discord



Come to office hours

Next up

Lecture 8 — POS Tagging & Sentiment Analysis