Module 8 Summary: NLP & Text Analysis

Data Science with Python – BSc Course

What We Covered

From Raw Text to Actionable Signals
Module 8 taught you to process unstructured text data and extract quantitative insights. You learned the complete NLP pipeline: cleaning messy text, representing documents as numbers, capturing semantic meaning, and measuring sentiment.

Four Lessons:

- L37: Text Preprocessing – tokenization, stopwords, stemming, lemmatization
- L38: Bag-of-Words & TF-IDF – document-term matrices, term weighting
- L39: Word Embeddings – Word2Vec, GloVe, semantic similarity
- L40: Sentiment Analysis – VADER, FinBERT, opinion mining

You can now convert any text corpus into structured features for ML models

L37 Recap: Text Preprocessing

Problem Solved: Clean and standardize raw text for analysis

Key Takeaways:

- Tokenization splits text into meaningful units (words, sentences)
- Remove stopwords to focus on content, but keep negation words
- Stemming is fast but crude; lemmatization is accurate but slower
- Adapt preprocessing strategy to document type and task

The Pipeline: Raw text → Lowercase → Tokenize → Remove stopwords → Lemmatize → Clean tokens

Order matters: tokenize first, then remove stopwords, then stem/lemmatize

L38 Recap: Bag-of-Words & TF-IDF

Problem Solved: Convert text documents into numerical feature vectors

Key Takeaways:

- Bag of Words counts word occurrences (simple but effective)
- TF-IDF weights by distinctiveness (common words get low scores)
- N-grams capture phrases and negations ("not good", "interest rate")
- Cosine similarity measures document relatedness

Tools: `CountVectorizer` for BoW, `TfidfVectorizer` for TF-IDF (sklearn)

TF-IDF + Logistic Regression is a strong baseline for text classification

L39 Recap: Word Embeddings

Problem Solved: Represent words as dense vectors that capture semantic meaning

Key Takeaways:

- Word embeddings capture meaning (similar words $\rightarrow$ similar vectors)
- Word2Vec learns from context: "you know a word by its neighbors"
- Cosine similarity quantifies semantic relatedness between words
- Pre-trained models (Word2Vec, GloVe, FastText) are ready to use

Key Insight: king $-$ man $+$ woman $\approx$ queen (vector arithmetic captures relationships)

Always use pre-trained embeddings for finance – trained on billions of words

L40 Recap: Sentiment Analysis

Problem Solved: Automatically measure sentiment and use it as market signals

Key Takeaways:

- VADER is fast, rule-based, good for social media (no training needed)
- FinBERT is slower but more accurate for financial text
- Sentiment signals can inform trading, but edge decays quickly
- Always validate with domain expertise and proper backtesting

Use Cases: News sentiment for trading signals, earnings call tone analysis, ESG report screening

Compound score (VADER) ranges -1 to $+1$; FinBERT outputs positive/negative/neutral labels

Key Concepts Mastered

The Complete NLP Toolkit

- **Preprocessing:** Tokenization, stopword removal, stemming, lemmatization
- **Vectorization:** Bag-of-Words, TF-IDF, document-term matrices
- **Semantic Representation:** Word2Vec, GloVe embeddings, cosine similarity
- **Opinion Mining:** VADER (rule-based), FinBERT (transformer-based)

Critical Insight: The quality of your preprocessing determines the quality of your model. Garbage in, garbage out applies more to text than any other data type.

Master text preprocessing and feature extraction to unlock unstructured data insights

Finance Applications

Where NLP Creates Value in Finance

- **Earnings Call Analysis:** Sentiment predicts stock price reaction hours before markets react
- **News Sentiment Trading:** Bloomberg and Thomson Reuters offer NLP-driven signals
- **SEC Filings:** Extract risk disclosures and management tone from 10-Ks and 10-Qs
- **Social Media Monitoring:** Twitter/X sentiment for crypto and meme stock detection

Real Example: Two Sigma and Renaissance Technologies process millions of text documents daily to generate alpha from alternative data.

Financial text is unstructured alpha waiting to be extracted

Tools & Code Patterns

Essential Libraries and Workflows

Preprocessing:

- `nltk.word_tokenize()`, `nltk.corpus.stopwords`, `PorterStemmer()`, `WordNetLemmatizer()`

Vectorization:

- `from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer`
- `vec = TfidfVectorizer(max_features=5000, ngram_range=(1,2))`

Embeddings:

- `gensim.downloader.load('word2vec-google-news-300')` or `gensim.models.Word2Vec`

Sentiment:

- `vaderSentiment.SentimentIntensityAnalyzer`, `transformers.pipeline('sentiment-analysis', model='ProsusAI/finbert')`

These tools form the standard NLP stack in finance and data science

What's Next: Module 9 – Deployment

From Prototype to Production
You can now build models that analyze text, detect patterns, and generate predictions. But a model in a Jupyter notebook creates zero business value.

Module 9 covers:

- Model Serialization (L41) – save and load trained models
- REST APIs with FastAPI (L42) – expose models as web services
- Interactive Dashboards with Streamlit (L43) – build user interfaces
- Cloud Deployment (L44) – deploy to AWS/Azure/GCP for real users

You'll take your NLP models from local scripts to production-ready APIs serving thousands of requests per second.

Deployment separates hobbyists from professionals – Module 9 bridges the gap

Module 8 Complete!

You can now process financial text at scale

Next up: Module 9 – Deployment

Questions?