

NLP – Unit 3 (Language Modelling) – END-SEM PYQ Answers

May-June 2023

Q1(a) — Markov Model for Predicting Next Word

[6 Marks]

- **Markov Model Concept:** Predicts the next word based only on the previous word(s) — called the Markov Assumption.
- **Key Idea:** $P(w_n | w_1, w_2, \dots, w_{n-1}) \approx P(w_n | w_{n-1})$ [Bigram / 1st order Markov]
- **Steps to Build a Markov Model:**
 - Step 1 — Tokenize corpus into words
 - Step 2 — Count bigram frequencies: $C(w_{i-1}, w_i)$
 - Step 3 — Count unigram frequencies: $C(w_{i-1})$
 - Step 4 — Compute transition probabilities: $P(w_i | w_{i-1}) = C(w_{i-1}, w_i) / C(w_{i-1})$
 - Step 5 — Represent as a Transition Matrix or State Diagram
- **Example:** Corpus: 'the cat sat on the mat'
 - Bigrams: (the,cat), (cat,sat), (sat,on), (on,the), (the,mat)
 - $P(\text{cat} | \text{the}) = C(\text{the,cat})/C(\text{the}) = 1/2 = 0.5$
 - $P(\text{mat} | \text{the}) = 1/2 = 0.5$
 - $P(\text{sat} | \text{cat}) = 1/1 = 1.0$

Note: A Markov model is memory-less — the prediction depends only on the current state (last word), not the entire history.

Q1(b) — Add-k Smoothing: Bigram Probability

[8 Marks]

- **Given:** Corpus = 10,000 words | Vocabulary = 5,000 | $k = 0.5$
 - $C(\text{the, cat}) = 50$ | $C(\text{the}) = 1000$ | $C(\text{cat}) = 100$

- **Add-k Smoothing Formula:**

$$P_k(w_i | w_{i-1}) = [C(w_{i-1}, w_i) + k] / [C(w_{i-1}) + k \times V]$$

- **Calculate $P(\text{cat} | \text{the})$:**

$$P(\text{cat} | \text{the}) = (50 + 0.5) / (1000 + 0.5 \times 5000) = 50.5 / 3500 = 0.01443$$

- **Sentence:** 'the cat sat on the mat'
 - $P(\text{the}) = \text{assumed} = 1/10000$ (start probability)
 - $P(\text{cat}|\text{the}) = 0.01443$

- $P(\text{sat}|\text{cat}) = (0+0.5)/(100+2500) = 0.5/2600 = 0.000192$
- $P(\text{on}|\text{sat}) = 0.5/(C(\text{sat})+2500) \approx \text{very small}$
- $P(\text{the}|\text{on}) = 0.5/(C(\text{on})+2500) \approx \text{very small}$
- $P(\text{mat}|\text{the}) = 0.5/3500 \approx 0.000143$
- Final probability = product of all above individual bigram probabilities

Note: Add-k smoothing prevents zero probabilities for unseen bigrams. $k=1$ is called Laplace smoothing. $k=0.5$ is a softer version.

Q1(c) — Latent Semantic Analysis (LSA)

[4 Marks]

- **Definition:** LSA is a technique to find hidden (latent) relationships between words and documents using linear algebra.
- **Core Idea:** Words with similar meanings appear in similar contexts.
 - Steps:
 1. Build a Term-Document Matrix (rows = terms, columns = docs)
 2. Apply TF-IDF weighting to the matrix
 3. Apply Singular Value Decomposition (SVD) to reduce dimensions
 4. Use reduced matrix to find semantic similarity
- **SVD:** $A = U \times \Sigma \times V^T$ — decomposes matrix into latent semantic space
- **Application:** Document similarity, query expansion, topic discovery

Note: LSA captures synonymy (same meaning, different words) and polysemy (same word, different meanings) to some extent.

Q2(a) — Generative vs Discriminative Models

[4 Marks]

- Generative language models learn the statistical patterns of text from large corpora, then use that learned distribution to produce new sequences of words token by token. At their core, they answer a single question repeatedly: given everything written so far, what comes next?

Feature	Generative Models	Discriminative Models
Goal	Model joint probability $P(X,Y)$	Model conditional $P(Y X)$
Learns	How data is generated	Decision boundary only
Examples	Naive Bayes, HMM, LDA	Logistic Regression, SVM, CRF
Can generate new data?	Yes	No
Data needed	Less	More

- **Generative Model Example:** N-gram LM — models $P(\text{word sequence})$ and can generate new sentences.
- **Discriminative Example:** Text classifier — models $P(\text{class} | \text{text})$ directly.

Q2(b) — TF-IDF Calculation**[6 Marks]**

- **Given Document-Term Matrix:**

	Doc 1	Doc 2	Doc 3
Term 1	10	5	0
Term 2	2	0	8
Term 3	1	3	6

- **Find:** TF-IDF of Term 1 in Document 1
- **Step 1 — TF (Term Frequency):** $TF = \text{count of term in doc} / \text{total terms in doc}$

Total terms in Doc1 = $10+2+1 = 13$ $TF(\text{Term1}, \text{Doc1}) = 10/13 = 0.769$

- **Step 2 — IDF (Inverse Document Frequency):**

$IDF = \log(N / df)$ where $N = \text{total docs}$, $df = \text{docs containing term}$ $N = 3$, $df(\text{Term1}) = 2$ (Doc1 and Doc2 have Term1) $IDF(\text{Term1}) = \log(3/2) = \log(1.5) = 0.405$

- **Step 3 — TF-IDF:**

$TF-IDF(\text{Term1}, \text{Doc1}) = 0.769 \times 0.405 = 0.3114$

Note: IDF reduces weight of common terms (appearing in many docs) and boosts rare but important terms.

Q2(c) — Latent Dirichlet Allocation (LDA) for Topic Modeling**[8 Marks]**

- **Definition:** LDA is a probabilistic generative model that discovers abstract topics in a collection of documents.
- **Assumptions:**
 - Each document is a mixture of multiple topics
 - Each topic is a probability distribution over words
- **Key Components:**
 - Documents — collections of words
 - Topics — hidden variables (latent), represented as word distributions
 - Word Distributions — each topic has a multinomial distribution over vocab
 - α (alpha) — controls document-topic distribution (Dirichlet prior)
 - β (beta) — controls topic-word distribution (Dirichlet prior)
- **LDA Generative Process (for each document):**
 - 1. Choose topic mixture $\theta \sim \text{Dirichlet}(\alpha)$
 - 2. For each word position: choose topic $z \sim \text{Multinomial}(\theta)$
 - 3. Choose word $w \sim \text{Multinomial}(\phi_n)$ based on topic z

- **Learning:** Inference via Variational Bayes or Gibbs Sampling to find best topic assignments.
- **Output:** K topics, each represented as top N words with probabilities.
- **Example Output (2 topics from news corpus):**
 - Topic 1: politics, election, government, vote, party → 'Politics'
 - Topic 2: match, goal, team, player, score → 'Sports'

Note: LDA is unsupervised — number of topics K must be specified in advance. Unlike LSA, LDA gives probabilistic interpretations.

November-December 2023

Q1(a) — Generative Models of Language

[4 Marks]

[REPEATED] — Refer to: May-June 2023 → Q2(a) [Generative vs Discriminative Models]

- **Additional: One model in detail — N-gram Language Model:**
 - Models the probability of a word sequence
 - Bigram: $P(w_1, w_2, \dots, w_n) = \prod P(w_i | w_{i-1})$
 - Training: count bigrams from corpus, normalize
 - Generation: start with <s>, sample next word using probabilities, end at </s>

Q1(b) — Bigram Probability Calculation

[8 Marks]

- **Given Corpus:**
 - <s> I am from Pune </s>
 - <s> I am a teacher </s>
 - <s> students are good and are from various cities </s>
 - <s> students from Pune do engineering </s>
- **Test Sentence:** <s> students are from Pune </s>
- **Step 1 — Count Bigrams from Training Corpus:**

Bigram	Count
<s> → I	2
I → am	2
am → from	1
am → a	1
from → Pune	2
Pune → </s>	1
a → teacher	1

<s> → students	2
students → are	1
are → good	1
students → from	1
from → various	1
are → from	1

• **Step 2 — Compute Bigram Probabilities for test sentence:**

$P(< s > \rightarrow \text{students}) = C(< s >, \text{students}) / C(< s >) = 2/4 = 0.5$
 $P(\text{students} \rightarrow \text{are}) = C(\text{students}, \text{are}) / C(\text{students}) = 1/2 = 0.5$
 $P(\text{are} \rightarrow \text{from}) = C(\text{are}, \text{from}) / C(\text{are}) = 1/2 = 0.5$
 $P(\text{from} \rightarrow \text{Pune}) = C(\text{from}, \text{Pune}) / C(\text{from}) = 2/4 = 0.5$
 $P(\text{Pune} \rightarrow < / s >) = C(\text{Pune}, < / s >) / C(\text{Pune}) = 1/2 = 0.5$

• **Step 3 — Final Probability:**

$P(\text{sentence}) = 0.5 \times 0.5 \times 0.5 \times 0.5 \times 0.5 = 0.03125$

Note: Note: $C(< s >) = 4$ (4 sentences), $C(\text{students}) = 2$ (appears as start in 2 sentences), $C(\text{from}) = 4$ total occurrences, $C(\text{are}) = 2$

Q1(c) — Latent Semantic Analysis for Topic Modelling

[6 Marks]

[REPEATED] — Refer to: May-June 2023 → Q1(c) [LSA Short Note]

- **Additional Detail — SVD breakdown:**
 - U matrix: document-topic relationships
 - Σ matrix: singular values (importance of each latent dimension)
 - V^T matrix: term-topic relationships
 - Truncate to top k singular values → reduced representation
- **LSA vs LDA:** LSA is deterministic (algebraic), LDA is probabilistic (generative).

Q2(a) — Short Note on BERT

[4 Marks]

- **BERT:** Bidirectional Encoder Representations from Transformers
- **Key Features:**
 - Bidirectional — reads text left-to-right AND right-to-left simultaneously
 - Contextualized embeddings — same word gets different vectors based on context
 - Pre-trained on: Masked Language Modeling (MLM) + Next Sentence Prediction (NSP)
- **MLM:** 15% of words masked randomly, model predicts masked words
- **NSP:** Given 2 sentences, predict if sentence B follows sentence A
- **Architecture:** Transformer Encoder — 12 layers (BERT-Base), 768 hidden dims, 12 attention heads

- **Fine-tuning:** Add task-specific layer → fine-tune on labeled data for classification, NER, QA etc.

Note: BERT revolutionized NLP by providing context-aware word representations unlike static embeddings (Word2Vec, GloVe).

Q2(b) — TF-IDF Calculation (Same Matrix)

[6 Marks]

[REPEATED] — Refer to: May-June 2023 → Q2(b)

Q2(c) — LDA for Topic Modeling

[8 Marks]

[REPEATED] — Refer to: May-June 2023 → Q2(c) [LDA algorithm and topic modeling]

May-June 2024

Q1(a) — Generative vs Discriminative Models with Example

[9 Marks]

[REPEATED] — Refer to: May-June 2023 → Q2(a) [Basic comparison]

- **Additional — Generative Model Example in NLP (N-gram):**
 - Learns joint distribution $P(\text{sentence}) = P(w_1) \times P(w_2|w_1) \times \dots \times P(w_n|w_{n-1})$
 - Can generate new text by sampling from learned distribution
 - Used in speech recognition, machine translation
- **Additional — How Generative Models Work:**
 1. Estimate $P(X|Y)$ — likelihood of features given class
 2. Estimate $P(Y)$ — prior probability of each class
 3. Use Bayes' theorem: $P(Y|X) = P(X|Y) \times P(Y) / P(X)$

Note: Generative models model the world; discriminative models model the decision boundary. Both are useful in different NLP tasks.

Q1(b) — LDA: Components and Topic Modeling

[9 Marks]

[REPEATED] — Refer to: May-June 2023 → Q2(c) [LDA detailed explanation]

- **Additional — LDA Key Components Summary:**

Component	Description	Example
Topics (K)	Latent themes discovered	K=3: politics, sports, tech
Documents	Mixture of topics	Article = 60% tech + 40% science
Word Distribution	$P(\text{word} \text{topic})$ for each topic	Topic=sports: 'ball' 0.1, 'goal' 0.08...
α (alpha)	Document-topic prior	Small α → focused topic per doc
β (beta)	Topic-word prior	Small β → few words per topic

Q2(a) — Contextualized Representations (BERT)**[10 Marks]****[REPEATED] — Refer to: Nov-Dec 2023 → Q2(a) [BERT short note]**

- **Additional — Advantages of Contextualized Representations:**
 - Handles polysemy: 'bank' in 'river bank' vs 'money bank' gets different vectors
 - Better performance on downstream tasks (NER, QA, classification)
 - Pre-training on large unlabeled corpus reduces need for labeled data
 - Transfer learning — one pre-trained model for many tasks
- **Disadvantages:**
 - Computationally expensive (large model size, slow inference)
 - Requires significant GPU memory
 - Black box — difficult to interpret
 - Fixed max input length (512 tokens for BERT-Base)

Q2(b) — Add-k Smoothing: Bigram Probability**[8 Marks]****[REPEATED] — Refer to: May-June 2023 → Q1(b) [Same corpus, same values, same calculation]****May-June 2025****Q1(a) — Log-Linear Models in NLP****[6 Marks]**

- **Definition:** Log-linear models compute probability as a weighted sum of features passed through a softmax (log-linear) function.
- **Formula:**

$$P(y | x) = \exp(w \cdot f(x, y)) / \sum_{y'} \exp(w \cdot f(x, y'))$$
 where: w = weight vector $f(x, y)$ = feature function Σ = normalization over all classes

- **Why 'log-linear'?**
 - Taking log: $\log P(y|x) = w \cdot f(x, y) - \log Z(x)$ [linear in features]
 - Linear in the log space
- **Key Properties:**
 - Can incorporate arbitrary features (words, POS tags, context, etc.)
 - Well-calibrated probabilities
 - Discriminative model — directly models $P(y|x)$
- **Applications in NLP:**
 - Text classification (spam detection, sentiment analysis)
 - POS tagging and Named Entity Recognition
 - Machine translation reranking
 - Language modeling (MaxEnt LM)
- **Training:** Maximum likelihood estimation using gradient descent / L-BFGS

Note: Logistic Regression is the simplest log-linear model. MaxEnt (Maximum Entropy) classifiers are also log-linear models.

Q1(b) — doc2vec vs word2vec**[8 Marks]**

- **word2vec:** Produces fixed-size vector for each word (word-level embedding).
- **doc2vec:** Extends word2vec to produce fixed-size vector for entire documents/paragraphs.

Feature	word2vec	doc2vec
Unit	Word	Document/Paragraph
Output	Word embeddings	Document embeddings
Methods	CBOW, Skip-gram	PV-DM, PV-DBOW
Context	Local word window	Global document context
Use case	Word similarity, analogy	Document classification, retrieval
Memory	One vector per word	One vector per doc + word vectors

- **doc2vec Methods:**
 - PV-DM (Distributed Memory): Paragraph vector acts as memory — combined with word context to predict next word
 - PV-DBOW (Distributed Bag of Words): Paragraph vector predicts random words from document

Note: doc2vec can represent documents of different lengths as fixed-size vectors, making them usable for ML models.

Q1(c) — Non-Negative Matrix Factorization (NMF) for Topic Modeling**[4 Marks]**

- **Definition:** NMF decomposes a document-term matrix V into two non-negative matrices W and H .

$V \approx W \times H$ V ($m \times n$): Document-term matrix (m docs, n terms) W ($m \times k$): Document-topic matrix H ($k \times n$): Topic-term matrix k = number of topics

- **Why Non-Negative?**
 - All values $\geq 0 \rightarrow$ additive parts-based representation
 - More interpretable than LSA (no negative values)
- **Algorithm:** Iterative multiplicative updates to minimize reconstruction error $\|V - WH\|^2$
- **Use Case:** Each column of H = one topic (top words with weights), each row of W = document's topic mixture

Note: NMF is similar to LDA in output but deterministic (algebraic) rather than probabilistic. Better for short documents.

Q2(a) — Markov Model for Language Generation**[6 Marks]****[REPEATED] — Refer to: May-June 2023 → Q1(a) [Building a Markov model]**

- **Additional — Language Generation Process:**
 - 1. Start with <start> token
 - 2. Look up row in transition matrix for current state
 - 3. Sample next word proportional to probabilities
 - 4. Repeat until <end> token generated
- **Higher-order Markov:** Trigram model: $P(w_i | w_{i-2}, w_{i-1})$ — more context but data sparsity issue

Q2(b) — Latent Semantic Analysis (LSA)**[8 Marks]****[REPEATED] — Refer to: May-June 2023 → Q1(c) [LSA short note]**

- **Additional — Detailed LSA Steps:**
 - 1. Build Term-Document Matrix X (rows=terms, cols=docs)
 - 2. Apply TF-IDF weighting to normalize term importance
 - 3. Apply SVD: $X = U \Sigma V^T$
 - 4. Truncate to top k dimensions: $X_k = U_k \Sigma_k V_k^T$
 - 5. Use X_k for downstream tasks
- **Similarity Computation:** Cosine similarity between document vectors in reduced space
- **Query Processing:** Map query into latent space using V matrix, then compute similarity
- **Limitations:**
 - No probabilistic interpretation
 - SVD is computationally expensive for large matrices
 - Cannot handle new words without recomputation

Q2(c) — TF-IDF Short Note**[4 Marks]**

- **TF-IDF:** Term Frequency-Inverse Document Frequency — a numerical statistic for word importance.
- **TF (Term Frequency):** How often word appears in a document

$$TF(t,d) = (\text{count of } t \text{ in } d) / (\text{total words in } d)$$

- **IDF (Inverse Document Frequency):** How rare/unique the word is across all documents

$$IDF(t) = \log(N / df(t)) \text{ where } N = \text{total docs, } df(t) = \text{docs containing } t$$

- **TF-IDF:**

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

- **Intuition:** Common words (the, is, a) → low IDF → low TF-IDF. Rare important words → high TF-IDF.
- **Uses:** Document ranking in IR, feature extraction for ML, keyword extraction

November-December 2025

Q1(a) — Bigram Model with Add-1 Smoothing

[9 Marks]

- **Given Corpus:**
 - S1: 'language models learn patterns'
 - S2: 'models learn from data'
 - S3: 'data helps improve language models'
- **Target Sentence:** 'language models learn from data'
- **Step 1 — Vocabulary:**
 - {language, models, learn, patterns, from, data, helps, improve} = 8 unique words
- **Step 2 — Unigram Counts:**

Word	Count
language	2
models	3
learn	2
patterns	1
from	1
data	2
helps	1
improve	1

- **Step 3 — Key Bigram Counts (for target sentence):**

Bigram	Count
language → models	2
models → learn	1
learn → from	0
from → data	1

- **Step 4 — Add-1 Smoothed Probabilities:**

$$P_{\text{smooth}}(w_2|w_1) = [C(w_1, w_2) + 1] / [C(w_1) + V] \text{ where } V = 8 \text{ (vocabulary size)}$$

$$P(\text{models}|\text{language}) = (2+1)/(2+8) = 3/10 = 0.3 \quad P(\text{learn}|\text{models}) = (1+1)/(3+8) = 2/11 = 0.182 \quad P(\text{from}|\text{learn}) = (0+1)/(2+8) = 1/10 = 0.1 \quad P(\text{data}|\text{from}) = (1+1)/(1+8) = 2/9 = 0.222$$

- **Step 5 — Final Sentence Probability:**

$$P(\text{sentence}) = P(\text{models}|\text{language}) \times P(\text{learn}|\text{models}) \times P(\text{from}|\text{learn}) \times P(\text{data}|\text{from}) = 0.3 \times 0.182 \times 0.1 \times 0.222 = 0.001212 \text{ (approximately)}$$

Note: Add-1 (Laplace) smoothing adds 1 to all bigram counts and V to denominator to ensure no zero probabilities for unseen bigrams.

Q1(b) — Probabilistic Language Modeling: Trigram Model with MLE**[9 Marks]**

- **Probabilistic Language Modeling:**
 - Goal: Assign probability to a sequence of words $P(w_1, w_2, \dots, w_n)$
 - Markov Assumption: Future word depends only on limited history (n-1 previous words)
- **Markov Assumptions by Order:**

Model	Assumption	Formula
Unigram	Each word independent	$P(w_1 \dots w_n) = \prod P(w_i)$
Bigram	Depends on 1 prev word	$P(w_i w_{i-1})$
Trigram	Depends on 2 prev words	$P(w_i w_{i-2}, w_{i-1})$
N-gram	Depends on n-1 prev words	$P(w_i w_{i-n+1}, \dots, w_{i-1})$

- **Trigram Model — MLE:**

$$P_{\text{MLE}}(w_i | w_{i-2}, w_{i-1}) = C(w_{i-2}, w_{i-1}, w_i) / C(w_{i-2}, w_{i-1})$$

- **Example:** Corpus: 'I like to play football. I like to play cricket.'

$$P(\text{football} | \text{like}, \text{to}) = C(\text{like}, \text{to}, \text{football}) / C(\text{like}, \text{to}) = 1/2 = 0.5$$

$$P(\text{cricket} | \text{like}, \text{to}) = C(\text{like}, \text{to}, \text{cricket}) / C(\text{like}, \text{to}) = 1/2 = 0.5$$

- **Computing sentence probability using MLE:**

$$P(\text{'I like to play'}) = P(I) \times P(\text{like} | <s>, I) \times P(\text{to} | I, \text{like}) \times P(\text{play} | \text{like}, \text{to})$$

Note: Trigram models capture more context than bigrams but suffer from data sparsity — most trigrams are never seen in training. Smoothing is essential.

Q2(a) — NMF vs LDA for Topic Modeling**[9 Marks]**

Feature	NMF	LDA
Type	Matrix factorization (algebraic)	Probabilistic graphical model
Basis	$V \approx W \times H$	Bayesian generative model
Output	Document-topic and topic-term matrices	Topic distributions per doc
Interpretability	High (non-negative)	High (probabilistic)
Sparsity	Can enforce sparsity	Built-in sparsity via Dirichlet
Speed	Faster convergence	Slower (sampling/variational inference)
Best For	Short docs, deterministic output	Long docs, uncertainty estimation

- **NMF Matrix Factorization Example:**

- Input V (3 docs \times 4 terms):

$$V = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 2 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 \end{bmatrix} \begin{matrix} (\text{doc1}) \\ (\text{doc2}) \\ (\text{doc3}) \end{matrix}$$

- After NMF with $k=2$ topics:

$$W (\text{doc-topic}) = \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \\ 0.1 & 0.9 \end{bmatrix} \begin{matrix} \rightarrow \text{doc1 mostly topic1} \\ \rightarrow \text{doc2 mixed} \\ \rightarrow \text{doc3 mostly topic2} \end{matrix} \quad H (\text{topic-term}) = \begin{bmatrix} \text{topic1 weights on terms} \\ \text{topic2 weights on terms} \end{bmatrix}$$

Q2(b) — TF-IDF Calculation on given documents

[9 Marks]

- **Documents:**

- D1: 'neural networks are powerful'
- D2: 'deep learning powers neural models'
- D3: 'networks and models are important'

- **Find:** TF-IDF of 'neural' in D1, D2, D3

- **Step 1 — Term Frequency:**

$$TF(\text{neural}, D1) = 1/4 = 0.25 \quad TF(\text{neural}, D2) = 1/5 = 0.20 \quad TF(\text{neural}, D3) = 0/5 = 0.00$$

- **Step 2 — Document Frequency (DF) and IDF:**

$$DF(\text{neural}) = 2 \quad (\text{appears in D1 and D2}) \quad IDF(\text{neural}) = \log(3/2) = \log(1.5) \approx 0.405$$

- **Step 3 — TF-IDF:**

$$TF-IDF(\text{neural}, D1) = 0.25 \times 0.405 = 0.101 \quad TF-IDF(\text{neural}, D2) = 0.20 \times 0.405 = 0.081 \quad TF-IDF(\text{neural}, D3) = 0.00 \times 0.405 = 0.000$$

- **Interpretation:** 'neural' is most significant in D1 (highest TF-IDF = 0.101). Not present in D3 at all.

Topic Frequency Analysis — Unit 3

Topics ranked by how often they appear across all exam sessions (2023–2025):

Rank	Topic	Frequency	Sessions Asked
1	LDA (Latent Dirichlet Allocation)	4×	MJ23, ND23, MJ24, ND25 (indirectly)
2	Bigram/N-gram Probability Calculations	4×	MJ23, ND23, MJ25(Nov), MJ25(May)
3	TF-IDF Calculation	3×	MJ23, ND23, ND25
4	LSA (Latent Semantic Analysis)	3×	MJ23, ND23, MJ25
5	Generative vs Discriminative Models	3×	MJ23, ND23, MJ24
6	BERT / Contextualized Representations	2×	ND23, MJ24
7	Markov Models	2×	MJ23, MJ25
8	NMF (Non-Negative Matrix Factorization)	2×	MJ25, ND25
9	doc2vec vs word2vec	1×	MJ25
10	Log-linear Models	1×	MJ25

Note: HIGH PRIORITY for exam: LDA, Bigram calculations (with smoothing), TF-IDF, LSA, BERT. These appear in almost every session. Master the calculation steps especially for bigram smoothing and TF-IDF.