WEEK - 1

1. Supervised ML & Sentiment Analysis

- **Definition**: Training a model on labeled data — features XXX map to labels YYY — to make predictions on unseen data.
- **Objective**: Learn parameters θ\thetaθ that minimize a cost (or maximize likelihood), aligning Y^ to true Y quizlet.com+14aman.ai+14jiaqifang.medium.com+14.
- **Sentiment analysis framing**: Here, tweets labeled *positive* (1) or *negative* (0) are modeled as a **binary classification** problem using logistic regression.

2. Preprocessing & Feature Extraction

1. **Text preprocessing**
   - Tasks: lowercasing, tokenization, stop-words removal, stemming/lemmatization, punctuation/emoji handling.

2. **Vocabulary building**
   - Create a dictionary VVV of all unique tokens from training tweets jiaqifang.medium.com+2aman.ai+2arxiv.org+2.

3. **Sparse encoding**
   - Represent each tweet as a vector of counts (or binary indicators/Tf): xi=[1, count+, count−,…]
   - In the Week 1 example, features were limited to three: bias term, *sum of positive-seed words*, *sum of negative-seed words* medium.com+15cocalc.com+15pangruitao.com+15.

Example:
Tweet: “I love learning NLP!” →
– Preprocessed → tokens: ["i","love","learning","nlp"]
– If "love" is in the positive seed list, positive_sum = 1; negative_sum = 0; bias = 1.
→ Feature vector = [1,1,0][1,1,0][1,1,0].

3. Logistic Regression Overview & Model

1. **Hypothesis / sigmoid**

$$z = \theta^T x, \quad h_\theta(x) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Outputs probability:

$$\hat{y} = P(Y=1 \mid x)$$

github.com+3en.wikipedia.org+3medium.com+3cocalc.com.

2. **Log-Odds / Logit**

$$\text{\textbackslash logit}(p) = \ln \frac{p}{1-p} = \theta^T x$$

3. **Odds ratio interpretation**
   - A unit increase in xjx_jxj multiplies odds by e^{θ_j} en.wikipedia.org+1ssq.github.io+1.

4. Cost Function & Model Training

- **Likelihood**:

$$P(y \mid x; \theta) = h_\theta(x)^y \left[1 - h_\theta(x)\right]^{1-y}$$

- **Log-likelihood** (maximize → equivalent to minimize negative):

$$\ell(\theta) = \sum_{i=1}^{N} \left[y^{(i)} \ln h_\theta(x^{(i)}) + (1 - y^{(i)}) \ln(1 - h_\theta(x^{(i)})) \right]$$

- **Cost (Cross-Entropy Loss)**
  J(θ)=−1Nℓ(θ) J(\theta) = -\frac{1}{N} \ell(\theta)J(θ)=−N1ℓ(θ)
  en.wikipedia.orgssq.github.ioaman.ai

- **Gradient ascent / descent update rule**:

$$J(\theta) = -\frac{1}{N}\ell(\theta)$$

5. Visualization & Interpretation

- Training with a 3-feature space (1,pos_sum,neg_sum):
    - Plot each tweet in (pos_sum, neg_sum) plane, colored by label quizlet.com+2cocalc.com+2deeplearning.ai+2.
    - Learned θ defines a decision boundary (line where:

      $\theta^T x = 0$)

      $$\text{neg} = \frac{-\theta_0 - \theta_1 \cdot \text{pos}}{\theta_2}$$

      θ_0: bias/intercept term

      θ_1: weight for the number of **positive words** in the tweet

      θ_2: weight for the number of **negative words** in the tweet

    - Visual lines also show growth direction (gradient orientation).

Thus, tweets on one side are classified positive; the other side as negative — enabling high separability in this space .

6. Assignment Lab Highlights

1. **Implement `process_tweet()`**: Preprocess and tokenize raw tweets.
2. **Build feature extractor `build_freqs()`**: Produce vocabulary and frequency matrix.
3. **Train logistic regression**: Compute cost, gradient updates, track convergence.
4. **Evaluate**: On test data; compute accuracy, positives vs negatives.
5. **Error analysis**: Review misclassified tweets—identify preprocessing or feature flaws. cocalc.com+1aman.ai+1aman.ai

WEEK - 2

1. Probability & Bayes' Rule

- **Basic probability**:

$$P(A) = \frac{\text{Number of occurrences of } A}{\text{Total number of occurrences in the sample space}}$$

- **Conditional probability**:
  P(A∣B)=P(A∩B)P(B) P(A|B) = \frac{P(A \cap B)}{P(B)}P(A∣B)=P(B)P(A∩B)

- **Bayes' Rule**:
  P(Y∣X)=P(X∣Y) P(Y)P(X) P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)}P(Y∣X)=P(X)P(X∣Y)P(Y)
  Used to reverse conditional probabilities and build generative classifiers
  measurespace.netlify.app+10mooc-list.com+10opencampus.gitbook.io+10medium.com+1en.wikipedia.org+1deeplearning.ai+3coursera.org+3mooc-list.com+3.

2. Naïve Bayes Model for Sentiment Analysis

a. Motivation

- We have two classes: **positive** and **negative** tweets.
- Goal: given a tweet, evaluate which class it's more likely in.

b. Independence assumption ("naïve")

- Treat each word's occurrence as **independent** given class — simplifies calculation:

$$P(Y|X) = \frac{P(X|Y)\,P(Y)}{P(X)}$$

  medium.com+1community.deeplearning.ai+1en.wikipedia.org.

c. Training: build vocabulary & word frequencies

1. Preprocess tweets (lowercase, tokenize, remove stop-words, stem).
2. Count how many times each word appears in positive tweets vs negative tweets.

3. Compute **conditional probabilities**:

$$P(w \mid \text{pos}) = \frac{\text{count}(w, \text{pos})}{\sum_{w'} \text{count}(w', \text{pos})}$$

3. Laplacian (Add‑one) Smoothing

- Addresses zero‑frequency: if a word never appears in a class, probability becomes zero.
- **Smoothed formula**:

$$P(w|c) = \frac{\text{count}(w, c) + 1}{\sum_{w'} \text{count}(w', c) + V}$$

where V = size of vocabulary. Ensures no zero probabilities and sums properly measurespace.netlify.app+15medium.com+15en.wikipedia.org+15.

4. Log‑Likelihood & Log‑Odds Ratio

- To avoid underflow when multiplying many small probabilities:
  Take **logarithm** and convert products to **sums**:

$$\log \frac{P(\text{pos}|\mathbf{x})}{P(\text{neg}|\mathbf{x})} = \log \frac{P(\text{pos})}{P(\text{neg})} + \sum_{i=1}^{n} \log \frac{P(x_i|\text{pos})}{P(x_i|\text{neg})}$$

- Interpret this score:
  - 0 → Predict **positive**
  - < 0 → Predict **negative** medium.com+1community.deeplearning.ai+1en.wikipedia.org.

5. Training the Naïve Bayes Classifier

Since NB is *generative*, learning is just counting, not gradient descent:

1. Preprocess and tokenize tweets.
2. Build vocab frequency tables for each class.
3. Compute smoothed conditional probabilities.
4. Compute **log‑prior**:

$$\log \frac{N_{\text{pos}}}{N_{\text{neg}}}$$

arxiv.org+6medium.com+6ben-mccloskey20.medium.com+6.

5. Compute **log-likelihood ratio** per word:

$$\lambda_w = \log \frac{P(w|\text{pos})}{P(w|\text{neg})}$$

medium.com+1en.wikipedia.org+1.

6. Testing / Inference

For each new tweet:

- Sum up log-priors with log-likelihoods of its words (ignore unseen words—they're neutral).

- Decision rule:

$$\hat{\text{score}} = \log \frac{P(\text{pos})}{P(\text{neg})} + \sum_{w \in \text{tweet}} \lambda_w$$

  → *Positive* if score > 0, else *Negative*.
- Evaluate accuracy by comparing predictions with labels en.wikipedia.org+13medium.com+13jiaqifang.medium.com+13.

7. Naïve Bayes Assumptions & Use Cases

- **Assumption**: features (words) are independent and word order is ignored — may misclassify sentences like *"not good"*.
- **Applications**: Despite simplicity, fast and effective for:
    - Spam filtering, authorship detection, information retrieval, word disambiguation medium.com.

WEEK - 3

1. Vector Space Models (VSM)

- **Objective**: Represent words or documents as vectors to capture semantic relationships.
- **Key idea** (Firth, 1957):
  "You shall know a word by the company it keeps."
  github.com+9aman.ai+9aman.ai+9
- Representations:
  - **Word-by-word**: co-occurrence counts within a context window kkk

    $$v_w = [\ldots, \mathrm{count}(w, u), \ldots]$$

  - **Word-by-document**: frequencies of words in entire documents/categories.

Example: With k=2k = 2k=2, “data” co-occurs with “simple” twice → entry in the co-occurrence matrix is 2 aman.ai+1en.wikipedia.org+1.

2. Similarity Measures

(a) Euclidean Distance

$$\|v - u\|_2 = \sqrt{\sum_{i=1}^{n} (v_i - u_i)^2}$$

Returns the straight-line distance between vectors .

(b) Cosine Similarity

Measures orientation similarity, ignoring magnitude:

$$\cos(\theta) = \frac{v \cdot u}{\|v\| \, \|u\|} = \frac{\sum_i v_i u_i}{\sqrt{\sum_i v_i^2} \, \sqrt{\sum_i u_i^2}}$$

Ranges: −1 (opposite) to +1 (identical)
en.wikipedia.org+15marcossilva.github.io+15en.wikipedia.org+15en.wikipedia.org.
Favored for text due to normalization of document length bias.

3. Vector Arithmetic & Analogy Tasks

Using word embeddings (like Word2Vec, GloVe), you can solve analogies:

$$v_{\text{man}} - v_{\text{woman}} + v_{\text{king}} \approx v_{\text{queen}}$$

Similarly, capital-city analogies:

$$v_{\text{USA}} - v_{\text{WashingtonDC}} + v_{\text{Russia}} \approx v_{\text{Moscow}}$$

medium.com

4. Dimensionality Reduction & PCA

(a) Why PCA?

Word embeddings are high-dimensional (e.g., 300D). PCA reduces dimensions (typically to 2 or 3) to **visualize** semantic clusters medium.com.

(b) PCA Algorithm

1. **Mean-normalize** data.
2. Compute covariance matrix Σ.
3. Perform SVD:

   $$\Sigma = U \Lambda U^T$$

   - U: eigenvectors ("directions")
   - Λ: eigenvalues ("variance explained")
4. Select the top k components and project:

   $$Z = X \cdot U_k$$ where columns of Uk are the top kkk eigenvectors knowledge.uzpg.me+2marcossilva.github.io+2medium.com+2medium.com+1aman.ai+1.

(c) Interpretation

The principal components are orthogonal axes capturing descending variance. Projecting embeddings onto them shows semantic clusters (e.g., "city" & "town").

5. Sample Formula Recap

1. **Euclidean distance**
   $\|v-u\|_2 = \sqrt{\sum_i (v_i - u_i)^2}$
2. **Cosine similarity**
   $\cos(v,u) = \frac{\sum_i v_i u_i}{\|v\| \, \|u\|}$
3. **PCA projection**
   $Z = XU_k$, where U_k contains top eigenvectors of covariance matrix

WEEK - 4

1. Transforming Word Vectors (Linear Mapping)

To translate between languages using word embeddings, we learn a **linear transformation matrix** RRR that maps English embeddings X to French embeddings Y:

$$XR \approx Y$$

We train RRR via least-squares minimization using the **Frobenius norm**:

$$\min_R \|XR - Y\|_F^2$$

- **Frobenius norm**:

  $$\|A\|_F^2 = \sum_{i,j} a_{ij}^2$$

- **Gradient** with respect to RRR:

$$\nabla_R = 2X^T(XR - Y)$$

Update rule via gradient descent:

$$R := R - \alpha \cdot \frac{2}{m} X^T(XR - Y)$$

Where:

- $X \in \mathbb{R}^{m \times d}$: English word vectors (m words, d dims)
- $Y \in \mathbb{R}^{m \times d}$: Corresponding French word vectors
- α: learning rate

elastic.co+13marcossilva.github.io+13measurespace.netlify.app+13

2. K-Nearest Neighbors (K-NN)

Once we have an English word vector transformed into French space (x*R*), identify its translation by finding the **closest** vector(s) in French embeddings using K-NN:

- **Classification** by finding kkk nearest neighbors via cosine or Euclidean similarity aman.ai

For k=1, translation is the nearest neighbor in the mapped space.

3. Hash Tables & Locality-Sensitive Hashing (LSH)

Exact K-NN is computationally expensive in high dimensions. LSH provides a fast, approximate alternative:

- **LSH idea**: Hash similar vectors into the **same bucket** with high probability arxiv.org+4en.wikipedia.org+4measurespace.netlify.app+4
- **Hash via random hyperplanes** (SimHash):

$$h_i(v) = \text{sign}(r_i \cdot v), \quad r_i \sim N(0, I)$$

Hash key: concatenation of multiple bits,

$$h(v) = [h_1(v), h_2(v), \ldots, h_k(v)]$$

- Use LLL independent hash tables (multiple sets of random hyperplanes) to amplify performance:
  - **AND-construction**: concatenation reduces false positives (p_1^k, p_2^k)
  - **OR-construction**: union over tables reduces false negatives (probability (probability $1 - (1 - p_1)^L$) arxiv.org+5en.wikipedia.org+5aman.ai+5

These hashed buckets yield a **small candidate set** to search with exact distance—yielding high speed at slight accuracy trade-off.

4. Document Search

Same techniques apply to sentences or documents:

- Represent document vectors
- Transform if necessary
- Use K-NN or approximate K-NN via LSH to find semantically similar documents

  coursera.org+7aman.ai+7elastic.co+7mooc-list.com+8measurespace.netlify.app+8github.com+8measurespace.netlify.app+2elastic.co+2github.com+2

---

5. Practical Workflow

1. **Load** pretrained embeddings X (English), Y (French).
2. **Learn** mapping RRR using aligned word pairs via gradient descent.
3. For a new English word/phrase:
   - Compute xR.
   - Use **LSH** to retrieve candidate candidates quickly.
   - Compute exact similarity among candidates.
   - Return top k translation(s).

4. Use the same for **document search** (sentence similarity).

Why it Matters

- **Linear mapping** captures bilingual semantic alignment.
- **K-NN** recovers similar semantic neighbors for translation or retrieval.
- **LSH** enables scalable, high-dimensional search with speed and practicality.