**Assignment: Building a Text‑Classification Pipeline & Word‑Embedding Exploration (2)**

**Overview**

In this assignment you will design **an end‑to‑end Natural Language Processing (NLP) pipeline** that turns raw text into actionable insights. You will:

1. **Pre‑process** and normalise the text (tokenisation, stop‑word removal, stemming, lemmatisation, etc.).
2. **Engineer features** using both shallow (Bag‑of‑Words/TF‑IDF/N‑grams) *and* dense distributed representations (Word2Vec/GloVe/FastText).
3. **Build and evaluate models** using:
   * Generative classifier → **Multinomial Naïve Bayes**
   * Discriminative classifier → **Logistic Regression** or **Linear SVM**
   * (Optional ⭐) Markov‑chain language model for simple text generation.
4. **Compare approaches** and reflect on trade‑offs (generative vs discriminative, sparse vs dense vectors).

Learning outcomes: You will consolidate the material covered in the modules listed below and practise reproducible ML engineering.

**Provided Topics & Skills to Demonstrate**

* Text normalisation techniques
* Tokenisation & stop‑word filtering
* Stemming and lemmatisation
* N‑gram language models & basic Markov models
* Feature‐vector representations: Bag‑of‑Words, CountVectorizer, TF‑IDF
* Naïve Bayes sentiment classification
* Generative vs. discriminative classifiers
* Word embeddings (training or utilising pre‑trained)

**Real‑life Use‑Case Framing**

Before diving into the pipeline, **choose a concrete, real‑world scenario and identify a stakeholder who would benefit** (e.g., emergency services triaging disaster tweets, e‑commerce managers analysing product reviews, telecoms filtering SMS spam). Frame **all** deliverables—exploratory analysis, modelling decisions, evaluation metrics, and discussion—around how your solution addresses that stakeholder's pain‑points. **Projects submitted as generic text‑classification demos without a clearly articulated use‑case will incur a deduction of up to 25 pts.**

**Dataset Samples**

| **Domain** | **Dataset** | **Link** |
| --- | --- | --- |
| Movie reviews (sentiment) | IMDb Large Movie Review | <https://ai.stanford.edu/~amaas/data/sentiment/> |
| Product reviews (sentiment) | Amazon Reviews (Kernels lite, balanced) | <https://s3.amazonaws.com/amazon-reviews-pds/readme.html> |
| Disaster tweet classification | Kaggle – Real vs. Fake Disaster Tweets | <https://www.kaggle.com/competitions/nlp-getting-started> |
| News category classification | AG News | <https://www.di.ens.fr/> ~leon / Challenges/AGN/ |
| Spam detection | SMS Spam Collection | <https://archive.ics.uci.edu/dataset/228> |

You may propose another public, CC‑BY‐friendly corpus (Recommended).

**Project Tasks & Milestones**

**1. Data Acquisition & Exploration (10 pts)**

* Download & inspect the raw corpus; summarise size, classes, class balance.
* Show 3–5 representative examples per class.

**2. Pre‑processing Pipeline (20 pts)**

* Implement a clean, reusable pipeline that performs **lower‑casing, regex cleaning, tokenisation, stop‑word removal, and stemming *or* lemmatisation**.
* Demonstrate the impact on at least one example string.

**3. Feature Engineering (20 pts)**

* **Sparse**: Build both **Bag‑of‑Words** and **TF‑IDF** feature matrices (uni‑gram + optional bi‑gram).
* **Dense**: Train **Word2Vec (CBOW or Skip‑Gram)** on the training split *or* use a pre‑trained embedding (e.g. gensim‑downloader) and average the token vectors to obtain document‑level embeddings.
* (Optional ⭐) Train a simple **character 3‑gram Markov chain** and generate 5 sample sentences.

**4. Modelling & Evaluation (30 pts)**

* Split data into train/validation/test (suggested 70/10/20).
* Train **Multinomial Naïve Bayes** on the sparse representations.
* Train **Logistic Regression *or* Linear SVM** on both sparse and dense representations.
* Evaluate using **accuracy + precision, recall, F1‑score** (macro‑average if classes > 2).
* Present results in a concise table.

**5. Analysis & Discussion (10 pts)**

* Compare generative vs. discriminative performance.
* Discuss how N‑gram size and embedding choice affected results.
* Reflect on speed, memory, and explainability.

**6. Reproducibility & Code Quality (10 pts)**

* All code must be **well‑commented** and organised in a single Jupyter notebook **or** Python script package with clear function boundaries.
* Use **requirements.txt** / environment yaml.
* Seed all random operations.

**Deliverables**

1. **Notebook/Script** (.ipynb or package) with narrative markdown, code, and output.
2. **README.md** summarising project, setup, and results (≤ 1 page).
3. **requirements.txt** or environment.yml.
4. (Optional) generated sample texts if doing the Markov ⭐ task.

Submit via Git repository

**Grading Rubric (100 pts)**

| **Component** | **Points** |
| --- | --- |
| Data Exploration & Visualisation | 10 |
| Pre‑processing Pipeline | 20 |
| Feature Engineering | 20 |
| Modelling & Metrics | 20 |
| Analysis & Discussion | 10 |
| Code Quality & Reproducibility | 10 |
| Documentation (README) | 10 |

**Hints & Recommended Libraries**

* *pandas*\*\*, **numpy**, \*\***matplotlib** for data handling & plots
* **nltk**\* or \*\***spaCy** for tokenisation, stop‑words, lemmatisation
* **scikit‑learn** for vectorisers and classifiers
* **gensim** for Word2Vec
* Use Pipeline / ColumnTransformer In scikit‑learn to chain steps cleanly.

**Extension Ideas**

* Compare classical embeddings vs. **sentence‑transformer** (BERT) embeddings.
* Build a small **Streamlit** or **Gradio** demo to classify new user‑entered text.
* Hyper‑parameter search with **Optuna**.