Mini Programming Assignment: Naive Bayes vs Logistic Regression

(lightweight version for the course project cycle)

Due: November 13rd

1 Goal & What You Will Submit

Compare **Naive Bayes (NB)** and **Logistic Regression (LR)** on *one* small dataset of your choice (from the options below). Keep the pipeline simple and reproducible.

Submit:

- A short PDF (2–3 pages) with results & discussion.
- A single script or notebook that reproduces your numbers.

2 Datasets (pick exactly *one*) & How to Download

All options can be downloaded programmatically in Python; no manual accounts or scraping required.

Option A: Text — SMS Spam (binary classification)

Task: *spam* vs *ham*.
Download (Hugging Face Datasets):

```
pip install datasets

from datasets import load_dataset
ds = load_dataset("sms_spam")
# ds["train"] has columns: "sms" (text), "label" (0=ham, 1=spam)
```

Option B: Text — IMDB Reviews (binary sentiment)

Task: *positive* vs *negative*.
Download (Hugging Face Datasets):

```
pip install datasets

from datasets import load_dataset
ds = load_dataset("imdb")
# ds["train"]["text"], ds["train"]["label"] (0=neg, 1=pos)
```

Option C: Tabular — UCI Adult / Census Income (binary)

Task: income $>$ \$50K vs $\leq$ \$50K.
Download (OpenML via scikit-learn):

```
pip install scikit-learn

from sklearn.datasets import fetch_openml
adult = fetch_openml(name="adult", version=2, as_frame=True)
X = adult.data    # mixed categorical/numeric
y = adult.target  # " <=50K" / " >50K"
```

Option D: Images — MNIST Digits (10 classes, keep it small)

Task: classify digits 0–9 using flattened pixels.
Download (OpenML via scikit-learn):

```
pip install scikit-learn
```

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml("mnist_784", version=1, as_frame=False)
X = mnist.data / 255.0  # scale to [0,1]
y = mnist.target.astype(int)
```

Tip: To keep runtime light, you may subsample (e.g., 10,000 train / 2,000 test).

3 What To Do (Step-by-Step)

1. **Split the data** into train/validation/test (e.g., 70/15/15). Use a fixed random seed.
2. **Preprocess features**:
   - Text: use bag-of-words or TF–IDF (`sklearn.feature_extraction.text`).
   - Tabular: one-hot encode categoricals; standardize numeric (`ColumnTransformer`).
   - Images: flatten pixels to vectors; optionally scale to $[0, 1]$.
3. **Train Naive Bayes** appropriate to the data:
   - Text: `MultinomialNB` (with Laplace smoothing).
   - Tabular w/ continuous: `GaussianNB`.
   - Images: `GaussianNB` or `BernoulliNB` (if you binarize pixels).
4. **Train Logistic Regression** (binary or multiclass as needed) with ℓ_2 regularization. Tune the strength:
   - In scikit-learn: `LogisticRegression(C={0.1,1,10}, max_iter=200)`.
   - Pick C on the validation set; then retrain on train+val with the best C.
5. **Evaluate on the test set**: accuracy and F1 (binary F1 for binary tasks; macro-F1 for MNIST).
6. **Inspect one simple diagnostic**: for NB, list top 10 indicative features per class (e.g., highest log-prob ratio for words); for LR, show top 10 weights by magnitude.

4 Minimal Code Skeleton (scikit-learn)

You may adapt the following pieces (pseudocode-level):

```
# Text example with TF-IDF + MultinomialNB vs LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score, f1_score
```

```
# X_text, y loaded as lists/arrays
X_tr, X_tmp, y_tr, y_tmp = train_test_split(X_text, y, test_size=0.30,
                                            random_state=42, stratify=y)
X_val, X_te, y_val, y_te = train_test_split(X_tmp, y_tmp, test_size=0.50,
                                            random_state=42, stratify=y_tmp)

nb = make_pipeline(TfidfVectorizer(), MultinomialNB(alpha=1.0))
nb.fit(X_tr, y_tr)
pred_nb = nb.predict(X_te)

best = None; bestC=None
for C in [0.1, 1, 10]:
    lr = make_pipeline(TfidfVectorizer(),
                       LogisticRegression(C=C, max_iter=200))
    lr.fit(X_tr, y_tr)
    f1 = f1_score(y_val, lr.predict(X_val), average="binary")
    if best is None or f1 > best: best, bestC, best_lr = f1, C, lr

pred_lr = best_lr.fit(list(X_tr)+list(X_val), list(y_tr)+list(y_val)).predict(X_te)

print("NB  acc/f1:", accuracy_score(y_te, pred_nb),
                     f1_score(y_te, pred_nb, average="binary"))
print("LR  acc/f1:", accuracy_score(y_te, pred_lr),
                     f1_score(y_te, pred_lr, average="binary"))
```

5 Report Checklist (2–3 pages)

- **Dataset**: name, task, size (after any subsampling), feature type.
- **Setup**: preprocessing choices, model variants, hyperparameters tried.
- **Results table**: test Accuracy and F1 for NB vs LR (and validation F1 used to pick C).
- **One figure or table**: e.g., top features (NB log-odds or LR weights).
- **Discussion (short)**: Which model won? Any signs of over/underfitting? How do model assumptions match the data? One concrete takeaway.

6 Grading (100 points)

- Correct pipeline & metrics (40)
- Clear writeup & table/figure (25)
- Sensible hyperparameter tuning for LR (20)
- Reproducible code, fixed seed, and readable structure (15)

7 (Optional) Tiny Extension Ideas

Do **one**, only if you have time; cap total runtime < 10 minutes:

- Calibrate LR probabilities (`CalibratedClassifierCV`) and compare Brier score.

- Class imbalance handling: compare accuracy vs macro-F1 on a downsampled version.
- For MNIST, compare GaussianNB vs BernoulliNB (binarized pixels).

8 Environment Notes

We recommend Python 3.10+, `scikit-learn`, and (if using text datasets) `datasets`. Please list exact package versions and a one-line command to run your script or notebook.

End of mini assignment.