```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV,
StratifiedKFold
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix, roc_curve

pd.set_option('display.max_colwidth', 200)
```

```
# Try common path
candidates = [Path('SMSSpamCollection')]
path = None
for c in candidates:
    if c.exists():
        path = c
        break
if path is None:
    path = Path('SMSSpamCollection')  # change if needed

print('Using file:', path)
```

```
df = pd.read_csv(path, sep='\t', header=None, names=['label', 'message'])
df.head()
```

```
def clean_text(s):
    s = str(s).lower()
    s = re.sub(r'[^a-z0-9\s]', ' ', s)
    s = re.sub(r'\s+', ' ', s).strip()
    return s

# Encode
df['label_num'] = df['label'].map({'ham':0, 'spam':1})
# Clean message
df['clean_msg'] = df['message'].apply(clean_text)
# Engineered features
spam_tokens = ['free', 'win', 'winner', 'call', 'claim', 'urgent', 'prize', 'congrat', 'cash']
```

```
def msg_features(s):
    words = s.split()
    length = len(s)
    word_count = len(words)
    avg_w = np.mean([len(w) for w in words]) if words else 0
    digits = sum(c.isdigit() for c in s)
    punct = sum(1 for c in s if not c.isalnum() and not c.isspace())
    spam_tok_count = sum(1 for t in spam_tokens if t in s)
    return pd.Series({'msg_len': length, 'word_count': word_count, 'avg_word_len': avg_w,
'n_digits': digits, 'n_punct': punct, 'spam_tok': spam_tok_count})
```

```
feat_df = df['clean_msg'].apply(msg_features)
df = pd.concat([df, feat_df], axis=1)
```

```
df.head()
```

```
X = df[['clean_msg', 'msg_len', 'word_count', 'avg_word_len', 'n_digits', 'n_punct',
'spam_tok']]
y = df['label_num']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)
X_train.shape, X_test.shape, y_train.mean(), y_test.mean()
```

```
class TextSelector(BaseEstimator, TransformerMixin):
    def __init__(self, key):
        self.key = key
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.key]

class NumberSelector(BaseEstimator, TransformerMixin):
    def __init__(self, keys):
        self.keys = keys
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.keys].values
```

```
text_transform = Pipeline([
    ('selector', TextSelector('clean_msg')),
    ('tfidf', TfidfVectorizer(ngram_range=(1,2), min_df=2))
])
```

```
num_features = ['msg_len', 'word_count', 'avg_word_len', 'n_digits', 'n_punct', 'spam_tok']
num_transform = Pipeline([
    ('selector', NumberSelector(num_features))
])
```

```
from scipy.sparse import hstack
```

```
def build_feature_matrix(X, fit_vectorizer=None):
    if fit_vectorizer is None:
        vec = TfidfVectorizer(ngram_range=(1,2), min_df=2)
        X_text = vec.fit_transform(X['clean_msg'])
        return X_text, vec
    else:
        vec = fit_vectorizer
        X_text = vec.transform(X['clean_msg'])
        return X_text, vec
```

```
# Test building
X_text_train, vect = build_feature_matrix(X_train)
X_num_train = X_train[num_features].values
from scipy.sparse import csr_matrix
X_train_comb = hstack([X_text_train, csr_matrix(X_num_train)])
X_train_comb.shape
```

```
# Train MultinomialNB
nb = MultinomialNB()
nb.fit(X_train_comb, y_train)

# Prepare test features
X_text_test, _ = build_feature_matrix(X_test, fit_vectorizer=vect)
X_num_test = X_test[num_features].values
X_test_comb = hstack([X_text_test, csr_matrix(X_num_test)])
```

```
# Predictions
nb_pred = nb.predict(X_test_comb)
nb_proba = nb.predict_proba(X_test_comb)[:,1]
```

```
# Logistic Regression (with default L2, use solver 'liblinear' for small datasets)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1000, solver='liblinear')
lr.fit(X_train_comb, y_train)
lr_pred = lr.predict(X_test_comb)
lr_proba = lr.predict_proba(X_test_comb)[:,1]
```

```
# Evaluation helper
```

```
def evaluate_model(y_true, y_pred, y_proba=None):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    auc = roc_auc_score(y_true, y_proba) if y_proba is not None else np.nan
    cm = confusion_matrix(y_true, y_pred)
    return {'accuracy':acc, 'precision':prec, 'recall':rec, 'f1':f1, 'roc_auc':auc,
'confusion_matrix':cm}
```

```
nb_metrics = evaluate_model(y_test, nb_pred, nb_proba)
lr_metrics = evaluate_model(y_test, lr_pred, lr_proba)
```

```
print('Naive Bayes:', nb_metrics)
print('Logistic Reg :', lr_metrics)
```

```
def cv_scores(model, X_comb, y, cv=5):
    skf = StratifiedKFold(n_splits=cv, shuffle=True, random_state=42)
    acc = cross_val_score(model, X_comb, y, cv=skf, scoring='accuracy')
    prec = cross_val_score(model, X_comb, y, cv=skf, scoring='precision')
    rec = cross_val_score(model, X_comb, y, cv=skf, scoring='recall')
    f1s = cross_val_score(model, X_comb, y, cv=skf, scoring='f1')
    return {'accuracy': acc, 'precision': prec, 'recall': rec, 'f1': f1s}

# Note: cross_val_score expects estimators that accept dense/sparse X; our X_train_comb is
sparse
```

```
nb_cv = cv_scores(nb, X_train_comb, y_train, cv=5)
lr_cv = cv_scores(lr, X_train_comb, y_train, cv=5)
```

```
print('NB CV accuracy mean:', nb_cv['accuracy'].mean())
print('LR CV accuracy mean:', lr_cv['accuracy'].mean())
```

```
from sklearn.model_selection import GridSearchCV

nb_param_grid = {'alpha':[0.1, 0.5, 1.0, 1.5, 2.0]}
nb_gs = GridSearchCV(MultinomialNB(), nb_param_grid, cv=5, scoring='f1', n_jobs=-1)
nb_gs.fit(X_train_comb, y_train)
print('NB best params:', nb_gs.best_params_, 'best score:', nb_gs.best_score_)
```

```
lr_param_grid = {'C':[0.01, 0.1, 1, 10], 'penalty':['l1','l2']}
lr_gs = GridSearchCV(LogisticRegression(solver='liblinear', max_iter=2000), lr_param_grid,
cv=5, scoring='f1', n_jobs=-1)
lr_gs.fit(X_train_comb, y_train)
print('LR best params:', lr_gs.best_params_, 'best score:', lr_gs.best_score_)
```

```
nb_best = nb_gs.best_estimator_
lr_best = lr_gs.best_estimator_

nbb_pred = nb_best.predict(X_test_comb)
nbb_proba = nb_best.predict_proba(X_test_comb)[:,1]
```

```
lrb_pred = lr_best.predict(X_test_comb)
lrb_proba = lr_best.predict_proba(X_test_comb)[:,1]
```

```
print('NB (best) metrics:', evaluate_model(y_test, nbb_pred, nbb_proba))
print('LR (best) metrics:', evaluate_model(y_test, lrb_pred, lrb_proba))
```

```
plt.figure()
fpr, tpr, _ = roc_curve(y_test, nbb_proba)
plt.plot(fpr, tpr, label=f'NB (AUC={roc_auc_score(y_test, nbb_proba):.3f})')

fpr2, tpr2, _ = roc_curve(y_test, lrb_proba)
plt.plot(fpr2, tpr2, label=f'LR (AUC={roc_auc_score(y_test, lrb_proba):.3f})')
```

```
plt.plot([0,1],[0,1],'--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend()
plt.show()
```

```
# Confusion matrices
import seaborn as sns
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
cm = confusion_matrix(y_test, nbb_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('NB Confusion Matrix')
plt.xlabel('Pred')
plt.ylabel('True')
```

```
plt.subplot(1,2,2)
cm2 = confusion_matrix(y_test, lrb_pred)
sns.heatmap(cm2, annot=True, fmt='d', cmap='Greens')
plt.title('LR Confusion Matrix')
plt.xlabel('Pred')
plt.ylabel('True')
plt.show()
```