```
import pandas as pd
import numpy as np
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.pipeline import Pipeline
```

```
from google.colab import files
files.upload()
```

Show hidden output

```
df = pd.read_csv("Full-Economic-News-DFE-839861.csv", encoding='latin-1')
df.head()
```

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	positivity	positivity:confidence	relevance	re
0	842613455	False	finalized	3	12/5/2015 17:48:27	3.0	0.6400	yes	
1	842613456	False	finalized	3	12/5/2015 16:54:25	NaN	NaN	no	
2	842613457	False	finalized	3	12/5/2015 01:59:03	NaN	NaN	no	
3	842613458	False	finalized	3	12/5/2015 02:19:39	NaN	0.0000	no	
4	842613459	False	finalized	3	12/5/2015 17:48:27	3.0	0.3257	yes	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.12/dist-packages (0.14.0)
Requirement already satisfied: numpy<3,>=1.25.2 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (2.0.
Requirement already satisfied: scipy<2,>=1.11.4 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (1.16
Requirement already satisfied: scikit-learn<2,>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn)
Requirement already satisfied: joblib<2,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn) (1.5.
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from imbalanced-learn
```

```
try:
    from imblearn.pipeline import Pipeline
    from imblearn.over_sampling import RandomOverSampler
    from imblearn.under_sampling import RandomUnderSampler
    IMBLEARN_AVAILABLE = True
except ImportError:
    from sklearn.pipeline import Pipeline
    IMBLEARN_AVAILABLE = False
```

```
df.dropna(subset=['text'], inplace=True)
df = df[['text', 'relevance']]
df.head()
```

	text	relevance
0	NEW YORK -- Yields on most certificates of dep...	yes
1	The Wall Street Journal Online</br></br>The Mo...	no
2	WASHINGTON -- In an effort to achieve banking ...	no
3	The statistics on the enormous costs of employ...	no
4	NEW YORK -- Indecision marked the dollar's ton...	yes

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.shape
```

```
(8000, 2)
```

```
df = df[df['relevance'].isin(['yes', 'no'])]
df['relevance'] = df['relevance'].apply(lambda x: 1 if x == 'yes' else 0)
```

```
def preprocess_text(text):
    if not isinstance(text, str):
        return ""
    text = text.lower()  # Lowercasing
    text = re.sub(r'<.*?>', '', text) # Remove HTML tags
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\d+', '', text)  # Remove digits
    return text
```

```
df.isnull().sum()
```

	0
text	0
relevance	0

dtype: int64

```
df['clean_text'] = df['text'].apply(preprocess_text)
```

```
df.head()
```

	text	relevance	clean_text
0	NEW YORK -- Yields on most certificates of dep...	1	new york yields on most certificates of depos...
1	The Wall Street Journal Online</br></br>The Mo...	0	the wall street journal onlinethe morning brie...
2	WASHINGTON -- In an effort to achieve banking ...	0	washington in an effort to achieve banking re...
3	The statistics on the enormous costs of employ...	0	the statistics on the enormous costs of employ...
4	NEW YORK -- Indecision marked the dollar's ton...	1	new york indecision marked the dollars tone a...

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
X = df['clean_text']
y = df['relevance']
```

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

```
classifiers = {
    "Naive Bayes": MultinomialNB(),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    # Refinement: Set dual=False for performance when n_samples > n_features
    "SVM (Balanced)": LinearSVC(class_weight="balanced", max_iter=2000, random_state=42, dual=False)
}
```

```
resampling_techniques = {
    "Original": None,
}
resampling_techniques["Oversampled"] = RandomOverSampler(random_state=42)
resampling_techniques["Undersampled"] = RandomUnderSampler(random_state=42)


feature_limits = [40000, 10000, 1000]


for limit in feature_limits:
    print(f"\n{'='*35} RESULTS FOR MAX_FEATURES = {limit} {'='*35}")
    vectorizer = CountVectorizer(stop_words='english', max_features=limit)

    for resample_name, resampler in resampling_techniques.items():
        print(f"\n--- Technique: {resample_name} Data ---")

        for name, clf in classifiers.items():
            # Skip Naive Bayes for undersampled data, as it's not a good fit
            if resample_name == "Undersampled" and name == "Naive Bayes":
                 print(f"\n--- Classifier: {name} (Skipped for Undersampling) ---")
                 continue

            # Create the pipeline
            if resampler:
                pipeline = Pipeline([
                    ('vectorizer', vectorizer),
                    ('resampler', resampler),
                    ('classifier', clf)
                ])
            else:
                 pipeline = Pipeline([
                    ('vectorizer', vectorizer),
                    ('classifier', clf)
                ])

            # Train and predict
            pipeline.fit(X_train, y_train)
            y_pred = pipeline.predict(X_test)

            # Avoid redundant runs for the balanced SVM with resampling
            if 'Balanced' in name and resampler is not None:
                continue

            print(f"\n--- Classifier: {name} ---")
            print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
            print("Classification Report:")
            print(classification_report(y_test, y_pred, target_names=['non-relevant (0)', 'relevant (1)'], zero_division=0))
```

```
=================================== RESULTS FOR MAX_FEATURES = 40000 ===================================

--- Technique: Original Data ---

--- Classifier: Naive Bayes ---
Accuracy: 0.7649
Classification Report:
                  precision    recall  f1-score   support

non-relevant (0)       0.88      0.83      0.85      1315
    relevant (1)       0.37      0.48      0.42       284

        accuracy                           0.76      1599
       macro avg       0.63      0.65      0.64      1599
    weighted avg       0.79      0.76      0.78      1599


--- Classifier: Logistic Regression ---
Accuracy: 0.7886
Classification Report:
                  precision    recall  f1-score   support

non-relevant (0)       0.85      0.91      0.88      1315
    relevant (1)       0.36      0.25      0.29       284

        accuracy                           0.79      1599
       macro avg       0.60      0.58      0.58      1599
    weighted avg       0.76      0.79      0.77      1599


--- Classifier: SVM (Balanced) ---
Accuracy: 0.7573
Classification Report:
                  precision    recall  f1-score   support

non-relevant (0)       0.85      0.86      0.85      1315
```

```
           relevant (1)       0.31      0.30      0.30       284

               accuracy                           0.76      1599
              macro avg       0.58      0.58      0.58      1599
           weighted avg       0.75      0.76      0.76      1599


--- Technique: Oversampled Data ---

--- Classifier: Naive Bayes ---
Accuracy: 0.6854
Classification Report:
                   precision    recall  f1-score   support

non-relevant (0)       0.90      0.69      0.78      1315
    relevant (1)       0.31      0.65      0.42       284

        accuracy                           0.69      1599
       macro avg       0.61      0.67      0.60      1599
    weighted avg       0.80      0.69      0.72      1599
```

Start coding or generate with AI.