

Project 2

NLP Classification Project

Code is displayed with syntax highlighting similar to Jupyter/Python editors, with the project figures included.

Project Workflow

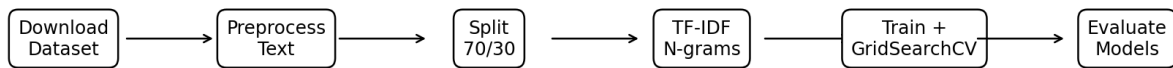


Figure 1. Project workflow.

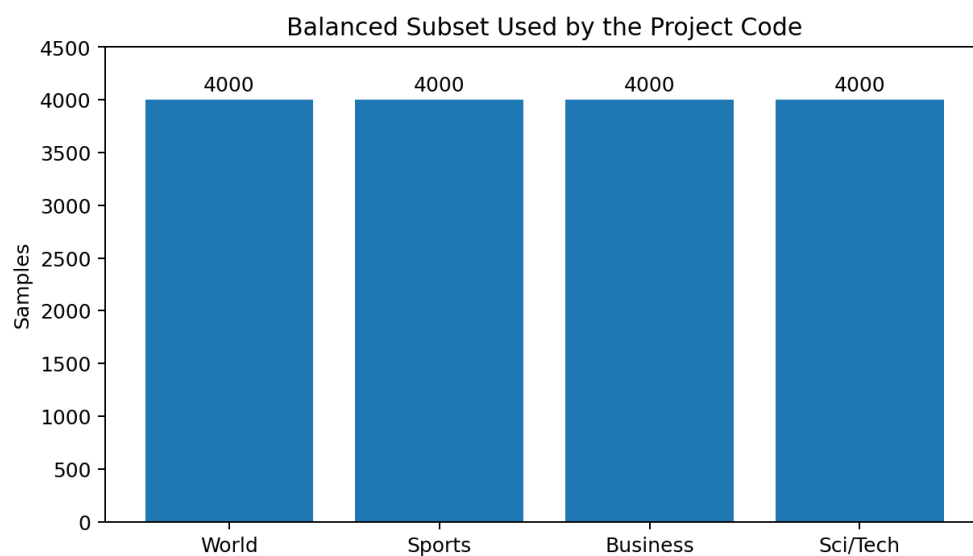


Figure 2. Balanced class sampling.

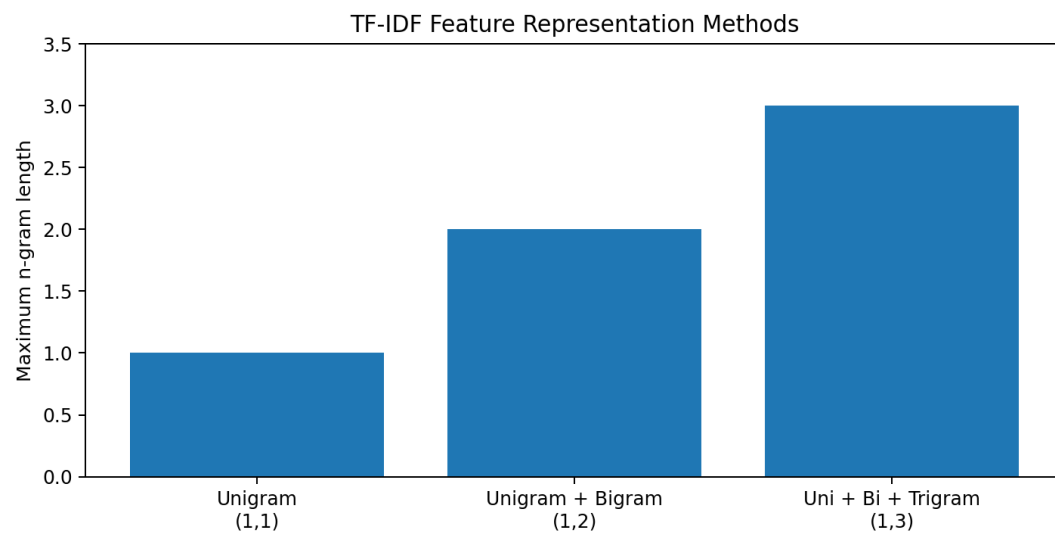


Figure 3. TF-IDF n-gram feature methods.

Project Code

NLP Classification Project

This project follows the required NLP classification workflow:

1. Download an NLP classification dataset.
2. Apply text preprocessing.
3. Split the dataset.
4. Apply feature representation methods.
5. Train and evaluate machine-learning models.
6. Save the results and trained best model.

Dataset: AG News Classification Dataset

Task: News topic classification into four classes.

1. Install / import required libraries

If a package is missing, uncomment the installation line.

```
# !pip install pandas numpy scikit-learn nltk matplotlib seaborn kagglehub joblib

import os
import re
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import joblib

import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

nltk.download("punkt", quiet=True)
nltk.download("punkt_tab", quiet=True)
nltk.download("stopwords", quiet=True)

RANDOM_STATE = 42
```

2. Download the NLP dataset

This project uses the **AG News Classification Dataset** from Kaggle.

The code below uses `kagglehub` to download it. If you already downloaded the dataset manually, set `DATASET\_DIR` to the folder containing `train.csv` and `test.csv`.

```
# Option A: Download automatically from Kaggle
try:
    import kagglehub
    DATASET_DIR = kagglehub.dataset_download("amananandrai/ag-news-classification-dataset")
    print("Dataset downloaded to:", DATASET_DIR)
except Exception as e:
    print("Automatic Kaggle download was not available:", e)
    print("Download the dataset manually from Kaggle, then set DATASET_DIR below.")
    DATASET_DIR = "."

print("DATASET_DIR =", DATASET_DIR)
```

3. Read and prepare the dataset

AG News contains:

- `Class Index`
- `Title`
- `Description`

We combine **Title + Description** into one text field.

```
train_file = os.path.join(DATASET_DIR, "train.csv")
test_file = os.path.join(DATASET_DIR, "test.csv")

# Some Kaggle folders may contain another nested folder.
if not os.path.exists(train_file):
    for root, dirs, files in os.walk(DATASET_DIR):
        if "train.csv" in files and "test.csv" in files:
            train_file = os.path.join(root, "train.csv")
            test_file = os.path.join(root, "test.csv")
            break

print("Train file:", train_file)
print("Test file :", test_file)

training = pd.read_csv(train_file, encoding="ISO-8859-1")
testing = pd.read_csv(test_file, encoding="ISO-8859-1")

print("Original training shape:", training.shape)
print("Original testing shape :", testing.shape)
display(training.head())

# Keep the project practical on a normal laptop:
# sample a balanced subset from each class from the original train and test files.
# Change SAMPLE_PER_CLASS_TRAIN / TEST to None if you want to use all rows.

SAMPLE_PER_CLASS_TRAIN = 3000
SAMPLE_PER_CLASS_TEST = 1000

if SAMPLE_PER_CLASS_TRAIN is not None:
    training = (
        training.groupby("Class Index", group_keys=False)
        .apply(lambda x: x.sample(n=min(SAMPLE_PER_CLASS_TRAIN, len(x)), random_state=RANDOM_STATE))
        .reset_index(drop=True)
    )

if SAMPLE_PER_CLASS_TEST is not None:
    testing = (
        testing.groupby("Class Index", group_keys=False)
        .apply(lambda x: x.sample(n=min(SAMPLE_PER_CLASS_TEST, len(x)), random_state=RANDOM_STATE))
        .reset_index(drop=True)
    )

training["text"] = training["Title"].fillna("") + " " + training["Description"].fillna("")
testing["text"] = testing["Title"].fillna("") + " " + testing["Description"].fillna("")

all_data = pd.concat(
    [training[["text", "Class Index"]], testing[["text", "Class Index"]]],
    ignore_index=True
)

print("Combined shape:", all_data.shape)
print(all_data["Class Index"].value_counts().sort_index())
```

4. Apply data preprocessing

The preprocessing steps are:

- Lowercasing
- Remove URLs
- Remove HTML tags
- Remove special characters, numbers, and punctuation
- Tokenization
- Remove English stop words
- Stemming using `PorterStemmer`

```
stemmer = PorterStemmer()
stop_words = set(stopwords.words("english"))

def preprocess_text(text):
    text = str(text).lower()

    # Remove URLs
    text = re.sub(r"http\S+|www\.\S+", "", text)

    # Remove HTML tags
    text = re.sub(r"<.*?>", "", text)

    # Remove special characters, numbers, and punctuation
    text = re.sub(r"[^a-zA-Z\s]", " ", text)

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stop words
    tokens = [word for word in tokens if word not in stop_words]

    # Stemming
    stemmed_tokens = [stemmer.stem(word) for word in tokens]

    return " ".join(stemmed_tokens)

all_data["clean_text"] = all_data["text"].apply(preprocess_text)
all_data = all_data.dropna(subset=["clean_text", "Class Index"])
```

```
all_data = all_data[all_data["clean_text"].str.strip() != ""]
display(all_data[["text", "clean_text", "Class Index"]].head())
```

5. Split the dataset

The project explicitly requires a split, so the combined cleaned dataset is divided into:

- **70% Training**
- **30% Testing**

`stratify=y` preserves the class distribution.

```
X = all_data["clean_text"].values
y = all_data["Class Index"].values

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    random_state=RANDOM_STATE,
    stratify=y
)

print("Training samples:", len(X_train))
print("Testing samples :", len(X_test))
print("Training class distribution:")
print(pd.Series(y_train).value_counts().sort_index())
```

6. Feature representation methods

The project applies **TF-IDF** with:

- Unigrams `(1,1)`
- Unigrams + Bigrams `(1,2)`
- Unigrams + Bigrams + Trigrams `(1,3)`

We compare all three.

```
feature_methods = {
    "TFIDF_Unigram": (1, 1),
    "TFIDF_Unigram_Bigram": (1, 2),
    "TFIDF_Unigram_Bigram_Trigram": (1, 3),
}

vectorized_data = {}

for name, ngram_range in feature_methods.items():
    vectorizer = TfidfVectorizer(
        ngram_range=ngram_range,
        max_features=30000
    )
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)

    vectorized_data[name] = {
        "vectorizer": vectorizer,
        "X_train": X_train_vec,
        "X_test": X_test_vec,
    }

print(name, "->", X_train_vec.shape)
```

7. Evaluation function

The evaluation metrics are:

- Accuracy
- Precision
- Recall
- F1 Score
- Classification Report
- Confusion Matrix

```
CLASS_NAMES = {
    1: "World",
    2: "Sports",
    3: "Business",
    4: "Sci/Tech"
}

def evaluate_model(model_name, feature_name, y_true, y_pred, plot_cm=True):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average="weighted", zero_division=0)
    recall = recall_score(y_true, y_pred, average="weighted", zero_division=0)
    f1 = f1_score(y_true, y_pred, average="weighted", zero_division=0)

    print("\n" + "=" * 70)
    print("Model:", model_name)
    print("Feature representation:", feature_name)
    print(f"Accuracy : {accuracy:.4f}")
```

```

print(f"Precision: {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1 Score : {f1:.4f}")
print("\nClassification Report:")
print(classification_report(
    y_true,
    y_pred,
    labels=[1,2,3,4],
    target_names=[CLASS_NAMES[i] for i in [1,2,3,4]],
    zero_division=0
))

if plot_cm:
    cm = confusion_matrix(y_true, y_pred, labels=[1,2,3,4])
    plt.figure(figsize=(6, 5))
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=[CLASS_NAMES[i] for i in [1,2,3,4]],
        yticklabels=[CLASS_NAMES[i] for i in [1,2,3,4]]
    )
    plt.title(f"Confusion Matrix - {model_name}\n{feature_name}")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.tight_layout()
    plt.show()

return {
    "Model": model_name,
    "Feature": feature_name,
    "Accuracy": accuracy,
    "Precision": precision,
    "Recall": recall,
    "F1 Score": f1,
}

```

8. Train and evaluate models

The machine-learning models are:

- Random Forest
- Logistic Regression
- Decision Tree
- Support Vector Machine
- K-Nearest Neighbors

To keep runtime practical, GridSearchCV is applied with compact parameter grids.

```

models_and_grids = {
    "RandomForest": (
        RandomForestClassifier(random_state=RANDOM_STATE, n_jobs=-1),
        {
            "n_estimators": [100, 200],
            "max_depth": [None, 20]
        }
    ),
    "LogisticRegression": (
        LogisticRegression(random_state=RANDOM_STATE, max_iter=1000),
        {
            "solver": ["liblinear", "lbfgs"]
        }
    ),
    "DecisionTree": (
        DecisionTreeClassifier(random_state=RANDOM_STATE),
        {
            "criterion": ["gini", "entropy"],
            "max_depth": [None, 20, 30]
        }
    ),
    "SVM": (
        SVC(random_state=RANDOM_STATE),
        {
            "kernel": ["linear", "rbf"]
        }
    ),
    "KNN": (
        KNeighborsClassifier(),
        {
            "n_neighbors": [3, 5, 7],
            "weights": ["uniform", "distance"]
        }
    )
}

results = []
trained_models = {}

# Set to False if you want faster execution and no confusion matrix for every run.
PLOT_ALL_CONFUSION_MATRICES = False

for feature_name, data in vectorized_data.items():
    Xtr = data["X_train"]
    Xte = data["X_test"]

```

```

print("\n" + "#" * 80)
print("FEATURE REPRESENTATION:", feature_name)
print("#" * 80)

for model_name, (model, param_grid) in models_and_grids.items():
    print(f"\nTraining {model_name} using {feature_name} ...")

    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grid,
        cv=3,
        scoring="accuracy",
        n_jobs=-1
    )

    grid_search.fit(Xtr, y_train)
    best_model = grid_search.best_estimator_
    y_pred = best_model.predict(Xte)

    print("Best parameters:", grid_search.best_params_)

    metrics = evaluate_model(
        model_name,
        feature_name,
        y_test,
        y_pred,
        plot_cm=PLOT_ALL_CONFUSION_MATRICES
    )

    metrics["Best Parameters"] = str(grid_search.best_params_)
    results.append(metrics)
    trained_models[(feature_name, model_name)] = best_model

```

9. Compare all models

```

results_df = pd.DataFrame(results)
results_df = results_df.sort_values(
    by=["Accuracy", "F1 Score"],
    ascending=False
).reset_index(drop=True)

display(results_df)

results_df.to_csv("model_comparison_results.csv", index=False)
print("Saved: model_comparison_results.csv")

```

10. Show the best model and confusion matrix

```

best_row = results_df.iloc[0]

best_feature = best_row["Feature"]
best_model_name = best_row["Model"]
best_model = trained_models[(best_feature, best_model_name)]
best_vectorizer = vectorized_data[best_feature]["vectorizer"]

X_test_best = vectorized_data[best_feature]["X_test"]
best_predictions = best_model.predict(X_test_best)

print("BEST MODEL")
print(best_row)

evaluate_model(
    best_model_name,
    best_feature,
    y_test,
    best_predictions,
    plot_cm=True
)

```

11. Save the best trained model and vectorizer

```

joblib.dump(best_model, "best_nlp_classifier.joblib")
joblib.dump(best_vectorizer, "best_tfidf_vectorizer.joblib")

print("Saved:")
print("- best_nlp_classifier.joblib")
print("- best_tfidf_vectorizer.joblib")

```

12. Test the model on a new news sentence

```

def predict_news_topic(text):
    clean = preprocess_text(text)
    vec = best_vectorizer.transform([clean])
    prediction = best_model.predict(vec)[0]
    return CLASS_NAMES.get(prediction, str(prediction))

sample_news = "The company announced a new artificial intelligence chip for mobile devices."
print("Text:", sample_news)
print("Predicted topic:", predict_news_topic(sample_news))

```