Introduction

The objective of this project is to classify sentiment from text data using four machine learning models. This task is a binary classification. Predicting whether a tweet is positive or negative.

Dataset

The data set was download from kaggle: [Sentiment140 dataset with 1.6 million tweets]
The dataset consists of pre-processed labeled text samples. It originally contains 1.6 million tweets that are extracted using twitter api. Each sample has been labeled with 0 =positive and 4 = negative. The dataset contains 5 columns: 'sentiment', 'id', 'date', 'query', 'user', and 'text'.

In data_prep.py, I load the entire dataset as a csv file. And kept only two relevant columns “sentiment’ and “text”. Convert sentiment labels and simplify it to 0(negative) and 1(positive) for consistency. In addition to this, I cleaned up the text using regular expressions. Removed parts that are irrelevant such as URLs, mentions and hashtags. And also removed duplicate and empty samples.

The 1.6million data are shuffled and only randomly select 160k samples for reducing computational complexity. The rest of samples are splitted into Training data, Validation data and Test data.

- Training data (70%) : 107471 samples
    - Positive : 53088 samples
    - Negative : 54383 samples
- Validation data (15%) : 23029 samples
    - Positive: 11,483
    - Negative: 11,546
- Test Data (15%) : 23030 samples
    - Positive: 11,545
    - Negative: 11,485

The dataset is balanced across both classes, ensuring fair evaluation of each model.

Feature Extraction

- Bag-of-words:
    - Text data is converted into numerical representation using CountVectorizer with a vocabulary of the top 5000 most frequent words.
    - Stop words like “the”, “and” are removed to focus on meaningful words.
    - The resulting feature matrix is sparse, with one row per text sample and one column per word.

Approaches

The following models were implemented and compared:

- Logistic Regression (implement from scratch)
    - Models the probability of a data point belonging to a specific class using sigmoid function
    - Mini-batch gradient descent was used to optimize the cross-entropy loss function.
    - Hyperprameters:
        - Learning rate:0.01

          This learning rate provided a good balance between convergence speed and stability. Smaller values made training too slow, while larger values may overshoot.

        - Batch size: 10

          Batch size of 10 resulted in faster convergence due to frequent updates. Larger batch sizes may lead to slower convergence.

        - Epochs: 50
- Naive Bayes (implement from scratch)
    - Assumes conditional independence between all features. It calculates the posterior probability for each class using Bayes' theorem.
    - Log Prior: Calculated from the proportion of positive and negative samples in the training set.
    - Log Likelihood : Computed for each word in the vocabulary for both classes using Laplace (add 1) smoothing to avoid zero probability.
    - Prediction: For given samples, the class with the highest posterior probability is chosen.
- Random Forest (Using sklearn RandomForestClassifier)
    - Using ensemble learning method that builds multiple decision trees and combines their predictions to produce final outputs.
    - Bootstrapping: for each tree, a random subset of the data is selected with replacement.
    - Feature selection: At each split in the decision tree, a random subset of features are selected. This will be controlled by max_features.
    - Hyperprameters:
        - Number of Trees (n_estimors): 100

          Number of decision trees in the ensemble. Large numbers will improve stability of the model but will increase the training time dramatically. To balance the stability and the training time I decided to set it to 100.

        - Maximum depth (max_depth): 20

          The maximum depth of each tree. Limiting the depth can prevent overfitting.

        - Features per split (max_features): 'sqrt'

        The number of features considered when finding the best split. There are three options: ‘sqrt’ square root of the total number of features, ‘log2’ Logarithm base 2 of the total features, ‘None’ Use all features. Choosing ‘sqrt’ will balance the training time and the performance. Using all features will increase the training time dramatically.

        - Class weight (class weight): ‘balanced’

          This parameter handles class imbalance. Since my data is really balanced in two classes. Therefore I set it to ‘balanced’.

        - n_jobs: -1

          Number of CPU cores for training and prediction. Set it to -1 uses all available cores.

- Support Vector Machine (Using sklearn LinearSVC from sklearn.svm)
    - SVM is to find the hyperplane that best separates data points from two classes.
    - Regularization : To prevent overfitting and control the trade-off between margin size and classification error
    - Hyperparamters:
        - Regularization Parameter (C):

          This parameter controls the trade-off between maximizing the margin and minimizing classification error. Smaller C will cause larger margin, allowing misclassified points for better generalization. Larger C will have a smaller margin, penalizing misclassification more heavily. Choosing C=1.0 balances the trade off.

Evaluation Metric

- Measure of Success

  The primary goal of this project is to classify the sentiment of text data accurately. Either positive or negative. To achieve this, multiple evaluation metrics are used to measure the success of the model.

- Accuracy:

  Measure the proportion of correctly classified samples to the total number of samples

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Samples}$$

- Precision:

  Measure the proportion of true positive predictions out of all positive predictions.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- Recall:

  Measure the proportion of true positive predictions out of all actual positives.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negatives}$$

- F1-Score:

  The harmonic mean of precision and recall. It balanced the trade-off between the two metrics.

$$F1 = 2 \bullet \frac{Precision \bullet Recall}{Precision + Recall}$$

- Confusion Matrix:
  Confusion matrix can show us a detailed breakdown of true positives , true negatives, false positives, and false negatives. It will help us understand the distribution of misclassifications. And easy to tell which class is favored by the model.

- Goal:

  The goal of this task is to build a robust sentiment classifier that performs well on unseen data. However, the success of models may depend on following factors: Accuracy might overestimate performance if the dataset is imbalanced, only high precision or recall may not reflect a well-rounded model.

Results

- Four models were implemented and evaluated. Each model was trained on a balanced dataset and evaluated on separate validation and test sets.

Model	Dataset	Accuracy	Precision	Recall	F1-Score

Logistic Regression	Validation	74%	73%	77%	75%
Logistic Regression	Test	75%	74%	78%	76%
Naive Bayes	Validation	74%	75%	73%	74%
Naive Bayes	Test	75%	76%	74%	75%
Random Forest	Validation	70%	66%	81%	73%
Random Forest	Test	70%	67%	81%	73%
SVM	Validation	74%	72%	78%	75%
SVM	Test	75%	74%	79%	76%

- Logistic Regression:
  - Achieved a balanced performance with an accuracy of 74% and a strong F1-Score of 75%. This model shows a good balance between precision and recall.
- Naive Bayes:
  - This model performs similarly to Logistic Regression, achieving accuracy of 74%. However, it slightly outperformed Logistic Regression in terms of presion and slightly underperformed in recall. Its F1-score is similar to Logistic Regression.
- Random Forest:
  - Performance is slightly worse than all other models, with an accuracy of 70%. Although it has the highest recall, this comes at the cost of a significantly lower precision. The F1-Score is also lower.
- SVM
  - Same accuracy as Logistic Regression and F1-Score. But with slightly lower precision on the validation set. And its recall matches Logistic Regression showing it performs well in identifying true positives.

Compare to baselines

- Baseline Performance

  Logistic Regression and Naive Bayes both serve as baselines. Showing strong results and competitive F1-Scores. Both models are simple to implement and perform well given the balanced class dataset.

- Improvement over Baseline

  SVM slightly outperforms Naive Bayes and Logistic Regression, making it a better option for this task. However, Random forest underperforms compared to the baselines due to its disadvantages on the ability to generalize effectively in high-dimensional feature spaces. This suggests that tree-based methods require further feature engineering or reducing the dimension of features. Either way might increase the computational cost dramatically.

Interpretation of the result

The dataset is balanced which ensures fair performance across all models. No significant bias toward any class is observed in the test results. The Bag-of-words representation of feature shows an effective feature space. However, its high dimensionality and sparsity seem to challenge models like Random Forest, which rely on feature splits. SVM's ability to create decision boundaries make it more effective for this task. Naive Bayes' assumption of feature independence makes the problem simpler, leading to competitive but not superior performance compared to logistic regression and SVM. The lower performance of Random Forest suggests that ensemble methods may struggle with unoptimized and spared features. Random Forest might perform better if using techniques like dimensionality reduction or selecting other feature representations.

How to Execute the code

- pip install -r requirements.txt To install required dependencies.
- python src/main.py <model> to execute the code
    - Replace model with 'logistic', 'naive_bayes','random_forest', and 'svm' for the desired model.

Data sets are already cleaned and splitted.

To redo this process run python src/data_prep.py