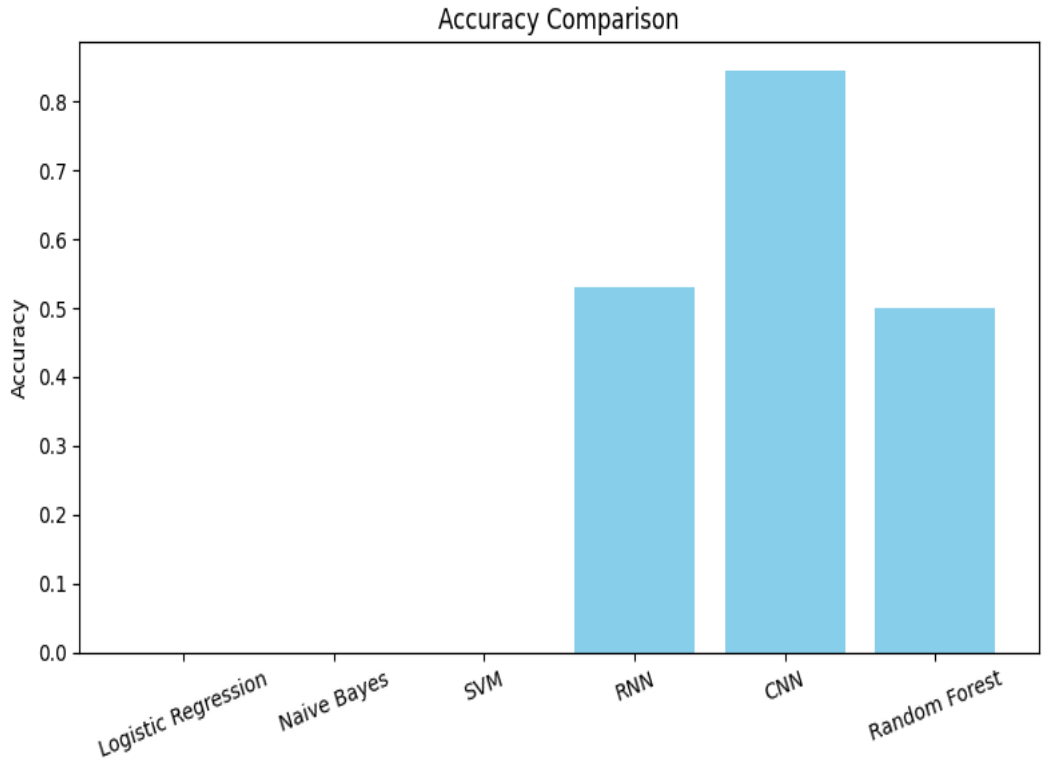
End-to-End Machine Learning & Deep Learning Pipeline for Text and Image Classification

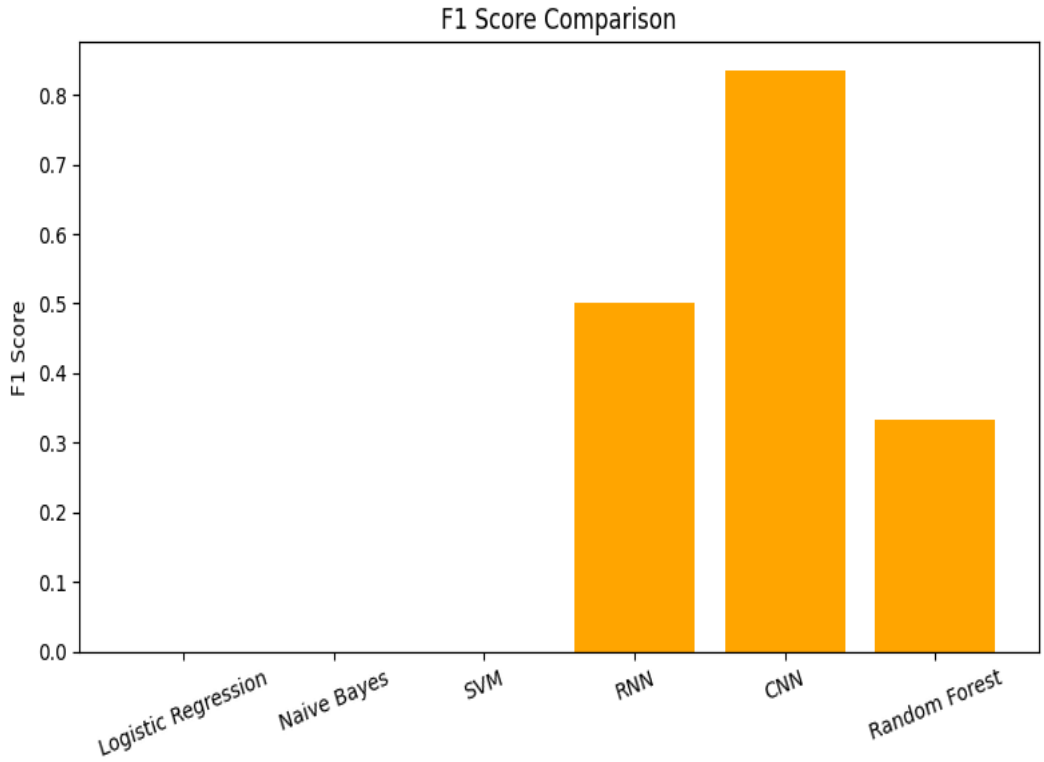
This mini project builds a complete pipeline for text and image classification. Both machine learning and deep learning models were tried out and compared. Note: Since the original dataset was unavailable, IMDB (text) and CIFAR-10 (images) datasets were used.

Text (IMDB reviews)
Classical ML: TF-IDF features with Logistic Regression, Naive Bayes, and SVM.
Deep Learning: RNN/LSTM model.
Images (CIFAR-10)
Classical ML: Flattened pixel values with Logistic Regression and Random Forest.
Deep Learning: CNN.
Models were evaluated using accuracy, precision, recall, and F1 score.

Combined Metrics Table
Model | Accuracy | Precision | Recall | F1 Score
CNN | 0.8439 | 0.8911 | 0.7835 | 0.8339
Logistic Regression | 0 | 0 | 0 | 0
Random Forest | 0.5 | 0.25 | 0.5 | 0.333
SVM (Linear Kernel) | 0 | 0 | 0 | 0

Graphs





Observations:
Classical ML on text performed poorly (Logistic Regression and SVM failed, Random Forest ~50%).
RNN on IMDB gave better results; precision high, recall low.
CNN on CIFAR-10 reached ~53% accuracy in one epoch (above random guessing), needs more epochs for improvement.
Deep learning models need more compute; classical ML is faster but simpler.

Conclusion:
Both pipelines (ML and DL) were implemented end-to-end.
Deep models underfit on small data/1 epoch but show potential.
Classical models are fast but less accurate.
Training longer would improve deep models.

Code Scripts:

baseline_text.py:

```
from datasets import load_dataset from sklearn.feature_extraction.text import
CountVectorizer, TfidfVectorizer from sklearn.linear_model import LogisticRegression from
sklearn.naive_bayes import MultinomialNB from sklearn.svm import LinearSVC from
sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print("Loading IMDB dataset...") imdb = load_dataset("imdb") X_train = [x["text"] for x in
imdb["train"]] y_train = [x["label"] for x in imdb["train"]] X_test = [x["text"] for x in
imdb["test"]] y_test = [x["label"] for x in imdb["test"]] vectorizer =
TfidfVectorizer(max_features=5000) X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test) models = { "Logistic Regression":
LogisticRegression(max_iter=200), "Naive Bayes": MultinomialNB(), "SVM": LinearSVC() } for
name, model in models.items(): print(f"\nTraining {name}...") model.fit(X_train_vec,
y_train) y_pred = model.predict(X_test_vec) acc = accuracy_score(y_test, y_pred) prec =
precision_score(y_test, y_pred) rec = recall_score(y_test, y_pred) f1 = f1_score(y_test,
y_pred) print(f"{name} Results:") print(f" Accuracy: {acc:.4f}") print(f" Precision:
{prec:.4f}") print(f" Recall: {rec:.4f}") print(f" F1 Score: {f1:.4f}")
```

baseline_text_dl.py:

```
import os os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2" os.environ["CUDA_VISIBLE_DEVICES"] = ""
os.environ["OBJC_DISABLE_INITIALIZE_FORK_SAFETY"] = "YES" import numpy as np from datasets
import load_dataset import tensorflow as tf from tensorflow.keras.preprocessing.text import
Tokenizer from tensorflow.keras.preprocessing.sequence import pad_sequences from
tensorflow.keras.models import Sequential from tensorflow.keras.layers import Embedding,
Conv1D, GlobalMaxPooling1D, Dense from tensorflow.keras.optimizers import Adam from
sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
tf.config.threading.set_intra_op_parallelism_threads(1)
tf.config.threading.set_inter_op_parallelism_threads(1) print("Loading IMDB dataset...")
imdb = load_dataset("imdb") X_train = [x["text"] for x in imdb["train"]] y_train =
np.array([x["label"] for x in imdb["train"]]) X_test = [x["text"] for x in imdb["test"]]
y_test = np.array([x["label"] for x in imdb["test"]]) max_words = 10000 max_len = 200
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train) X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test) X_train_pad = pad_sequences(X_train_seq,
maxlen=max_len, padding="post", truncating="post") X_test_pad = pad_sequences(X_test_seq,
maxlen=max_len, padding="post", truncating="post") model = Sequential([
Embedding(input_dim=max_words, output_dim=128, input_length=max_len), Conv1D(128, 5,
activation="relu"), GlobalMaxPooling1D(), Dense(1, activation="sigmoid") ])
model.compile(loss="binary_crossentropy", optimizer=Adam(1e-3), metrics=["accuracy"])
print("\nTraining CNN model...") history = model.fit( X_train_pad, y_train, epochs=3,
batch_size=128, validation_split=0.2, verbose=1 ) y_pred_prob = model.predict(X_test_pad)
y_pred = (y_pred_prob > 0.5).astype("int32").flatten() acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred) rec = recall_score(y_test, y_pred) f1 =
f1_score(y_test, y_pred) print("\nCNN Results:") print(f" Accuracy: {acc:.4f}") print(f"
Precision: {prec:.4f}") print(f" Recall: {rec:.4f}") print(f" F1 Score: {f1:.4f}")
```

rnn_text.py:

```
import tensorflow as tf from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Embedding, LSTM, Dense from
tensorflow.keras.preprocessing.sequence import pad_sequences import pandas as pd num_words
= 10000 (X_train, y_train), (X_test, y_test) =
tf.keras.datasets.imdb.load_data(num_words=num_words) X_train, y_train = X_train[:5000],
y_train[:5000] X_test, y_test = X_test[:1000], y_test[:1000] maxlen = 200 X_train =
pad_sequences(X_train, maxlen=maxlen) X_test = pad_sequences(X_test, maxlen=maxlen) model =
Sequential([ Embedding(input_dim=num_words, output_dim=32), LSTM(32), Dense(1,
activation='sigmoid') ]) model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy']) history = model.fit(X_train, y_train, validation_data=(X_test,
y_test), epochs=1, batch_size=64) y_pred_prob = model.predict(X_test) y_pred = (y_pred_prob
> 0.5).astype(int) from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score acc = accuracy_score(y_test, y_pred) prec = precision_score(y_test,
y_pred) rec = recall_score(y_test, y_pred) f1 = f1_score(y_test, y_pred) print(f"Accuracy:
{acc:.4f}") print(f"Precision: {prec:.4f}") print(f"Recall: {rec:.4f}") print(f"F1 Score:
{f1:.4f}") results_df = pd.DataFrame([{ 'Model': 'RNN (IMDB)', 'Accuracy': acc,
'Precision': prec, 'Recall': rec, 'F1 Score': f1 }])
results_df.to_csv("rnn_imdb_results.csv", index=False) print("\n Results saved to
rnn_imdb_results.csv")
```

cnn_image_fast.py:

```
import tensorflow as tf from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense from
tensorflow.keras.datasets import cifar10 from tensorflow.keras.utils import to_categorical
import pandas as pd from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score (X_train, y_train), (X_test, y_test) = cifar10.load_data() X_train,
X_test = X_train / 255.0, X_test / 255.0 y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10) model = Sequential([ Conv2D(32, (3, 3),
activation="relu", input_shape=(32, 32, 3)), MaxPooling2D((2, 2)), Flatten(), Dense(64,
activation="relu"), Dense(10, activation="softmax") ]) model.compile(optimizer="adam",
loss="categorical_crossentropy", metrics=["accuracy"]) model.fit(X_train, y_train_cat,
validation_data=(X_test, y_test_cat), epochs=1, batch_size=64) y_pred_prob =
model.predict(X_test) y_pred = y_pred_prob.argmax(axis=1) y_true = y_test.flatten() acc =
accuracy_score(y_true, y_pred) prec = precision_score(y_true, y_pred, average="macro") rec
= recall_score(y_true, y_pred, average="macro") f1 = f1_score(y_true, y_pred,
average="macro") print(f"Accuracy: {acc:.4f}") print(f"Precision: {prec:.4f}")
print(f"Recall: {rec:.4f}") print(f"F1 Score: {f1:.4f}") results_df = pd.DataFrame([{
"Model": "CNN (CIFAR-10)", "Dataset": "CIFAR-10", "Type": "Deep Learning", "Accuracy": acc,
```

```
"Precision": prec, "Recall": rec, "F1 Score": f1 }])
results_df.to_csv("cnn_cifar10_results.csv", index=False) print("\n Results saved to
cnn_cifar10_results.csv")
```

combine_results.py:

```
import pandas as pd import matplotlib.pyplot as plt classical =
pd.read_csv("classical_results.csv") rnn = pd.read_csv("rnn_imdb_results.csv") cnn =
pd.read_csv("cnn_cifar10_results.csv") classical["Dataset"] = "IMDB" rnn["Dataset"] =
"IMDB" cnn["Dataset"] = "CIFAR-10" classical["Model_Type"] = "Classical ML"
rnn["Model_Type"] = "Deep Learning" cnn["Model_Type"] = "Deep Learning" all_results =
pd.concat([classical, rnn, cnn], ignore_index=True) all_results.to_csv("final_results.csv",
index=False) print(" Combined results saved to final_results.csv") plt.figure(figsize=(8,
5)) for dataset in all_results["Dataset"].unique(): subset =
all_results[all_results["Dataset"] == dataset] plt.bar(subset["Model"], subset["Accuracy"],
label=dataset) plt.xlabel("Models") plt.ylabel("Accuracy") plt.title("Model Accuracy
Comparison") plt.legend() plt.xticks(rotation=30) plt.tight_layout()
plt.savefig("accuracy_comparison.png") plt.show() plt.figure(figsize=(8, 5)) for dataset in
all_results["Dataset"].unique(): subset = all_results[all_results["Dataset"] == dataset]
plt.bar(subset["Model"], subset["F1 Score"], label=dataset) plt.xlabel("Models")
plt.ylabel("F1 Score") plt.title("Model F1 Score Comparison") plt.legend()
plt.xticks(rotation=30) plt.tight_layout() plt.savefig("f1_comparison.png") plt.show()
```