CS634 – Data Mining
Final Project Report

Name: Kenil Pravinbhai Avaiya
NJIT UCID: ka683
Email Address: ka683@njit.edu
Date: 11-10-2025
Professor: Dr. Yasser Abdullah
Course: CS 634 101 Data Mining

Title

Comparative analysis using supervised and deep learning algorithms for binary classification on a superhero dataset.

Abstract

This final project we tried to identify the most suitable machine learning model for a binary classification task utilizing a dataset of superhero attributes. The main aim was to ascertain which algorithm would most effectively manage the intricate relationships between various features, including physical attributes, training levels, and special abilities. We put three models to the test: k-Nearest Neighbours (kNN), Random Forest, and Long Short-Term Memory (LSTM). The performance of each model was meticulously assessed through a comprehensive evaluation of several key metrics, including accuracy, precision, recall, and specificity. Based on the findings, the Random Forest algorithm emerged as the most effective solution for this specific problem, consistently delivering the most robust and reliable results across all evaluation parameters. This research provides a clear comparison and identifies the optimal approach for predictive analysis on data of this nature.

Introduction

In the field of data mining, binary classification is a core task where the goal is to predict one of two possible outcomes based on a given set of input features. This project applies this concept to a unique dataset, aiming to classify superheroes into a binary category of 'good' or 'not good' by analysing their various physical attributes and special capabilities.

To achieve this, a comparative analysis of three distinct supervised learning algorithms was undertaken:

1. **k-Nearest Neighbors (kNN):** A simple, intuitive algorithm that classifies a data point based on how its nearest neighbors in the dataset are classified.

2. **Random Forest:** An ensemble method builds many decision trees during training. Instead of a single complex tree, it combines simpler trees' results for more robust and accurate predictions.
3. **Long Short-Term Memory (LSTM):** A specialized type of Recurrent Neural Network (RNN) designed to recognize patterns in sequences of data. It can capture complex, non-linear relationships, making it suitable for structured tabular data, not just time-series or text.

Each of these algorithms was trained and evaluated on the same superhero dataset. Their performance was thoroughly assessed using standard metrics, including Accuracy, Precision, Recall, and Specificity. The primary objective of this study is to identify the model that achieves the most effective and balanced performance, providing the optimal combination of predictive accuracy and reliability for this classification task.

Important Concepts, and Principles

This section elucidates the fundamental mechanisms underlying the three machine learning algorithms employed in this project.

1. k-Nearest Neighbors (kNN)

kNN is a straightforward, instance-based learning algorithm applicable to both classification and regression tasks. Its operational principles can be comprehended through four distinct steps:

Consider a hypothetical scenario where we aim to ascertain whether a novel superhero possesses the ability to fly based on their physical attributes and cognitive abilities.

1. **Data Collection:** Compile a dataset of existing superheroes with information on their strength, intelligence, and flight capabilities.
2. **Select 'k':** We select a value 'k', which represents the number of nearest neighbors to be consulted. For instance, if we set 'k' to 3, it means we will consider the three closest neighbors for each point. Set k=3. Algorithm Overview
   The algorithm will consider the three most similar superheroes to the new one.
3. **Measure Distance:** Upon the emergence of a novel, unclassified superhero, the algorithm calculates its distance to every other superhero within the dataset employing a metric such as Euclidean distance, which represents the straight-line distance between two points.
4. **Majority Vote:** Subsequently, the algorithm identifies the "k" closest neighbors (e.g., the three most similar superheroes). The final classification (fly or not fly) is determined by the majority vote among these neighbors.

In essence, K-Nearest Neighbors (kNN) operates on the fundamental principle that entities that are similar tend to be found in proximity.

2. Random Forest

Random Forest is an ensemble learning technique that enhances predictive accuracy and mitigates overfitting.

In contrast to a single, meticulously crafted Decision Tree that may be overly tuned to the training data (a concern known as overfitting), Random Forest constructs a “forest” of numerous simpler trees.
Each tree within the forest is trained on a randomized subset of the data and a randomized subset of the features. This introduces randomness, ensuring that the individual trees are distinct from one another.
During prediction, each tree in the forest casts a vote on the outcome, and the final prediction is determined by majority voting (for classification) or averaging (for regression).

Combining multiple models makes Random Forest highly resilient and accurate. It naturally accommodates intricate data interactions, making it well-suited for our diverse superhero attributes dataset.

3. Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a specialized type of Recurrent Neural Network (RNN) architecture. While RNNs are designed for sequential data (such as time series or sentences), their conventional version encounters difficulties in retaining information from numerous steps back. LSTM addresses this challenge through an ingenious “gated” memory mechanism.

Its fundamental functionality:

Forget Gate: This gate evaluates incoming new information and the preceding context to identify which portions of existing memory are no longer pertinent and should be discarded.
Input Gate: This gate assesses the value of novel information received from the current input and determines which components should be stored in the memory cell.
Output Gate: Finally, this gate controls the output of information from the current memory cell to the subsequent step, considering the current input and the updated memory.

Despite their computational complexity and the need for meticulous tuning, Long Short-Term Memory (LSTM) networks were included in this analysis to explore their ability to discern intricate and non-linear patterns within our structured tabular data. LSTMs treat the sequence of features as a pattern that needs to be learned.

Project Structure and Workflow

1. About Datasets

Data set link: [Superheroes Data Set]

This project involves a detailed file containing information on a collection of fictional superheroes. Each entry provides comprehensive details about various physical attributes, including height, weight, and age. Additionally, it includes years of active service, training habits, and notable metrics such as public approval and power levels. Binary columns indicate the presence of specific abilities, such as super strength, flight, telepathy, shapeshifting, and others. Furthermore, a label is provided for moral alignment, indicating whether the superhero is “is_good” or “is_evil.”

Description of each field in the superhero dataset:

- **height_cm:** Height of the superhero in centimeters.
- **weight_kg:** Weight of the superhero in kilograms.
- **age:** Age of the superhero in years.
- **years_active:** Total number of years the superhero has been active.
- **training_hours_per_week:** Average number of hours the superhero dedicates to training each week.
- c**ivilian_casualties_past_year:** The number of civilian casualties attributed to the superhero within the past year.
- **power_level:** Overall power level rating of the superhero, expressed on an arbitrary or calculated scale.
- **public_approval_rating:** Public approval rating expressed as a percentage (0-100).
- **super_strength:** Indicates whether the superhero possesses superhuman strength (1) or not (0).
- **flight:** Indicates whether the superhero possesses the ability to fly (1) or not (0).
- **energy_projection:** Indicates whether the superhero has the ability to project energy (1) or not (0).
- telepathy: Indicates whether the superhero possesses telepathic abilities (1) or not (0).
- **healing_factor:** Indicates whether the superhero has accelerated self-healing abilities (1) or not (0).
- **shape_shifting:** Indicates whether the superhero has the ability to shape-shift (1) or not (0).
- **invisibility:** Indicates whether the superhero has the ability to become invisible (1) or not (0).
- **telekinesis:** Indicates whether the superhero has the ability to manipulate objects with their mind (1) or not (0).

- **is_good:** Indicates whether the superhero is classified as “good” (1) or “evil” (moral alignment).

Preliminary Data Insights
A preliminary analysis of the dataset reveals key points that informed the machine learning process:

Diverse power distribution: The `power_level` feature shows a wide range, indicating significant variance in capabilities among superheroes.
Mixed public perception: The `public_approval_rating` varies dramatically, suggesting that power or actions don’t always correlate with public support.
Balanced target variable: The binary `is_good` variable is well-balanced, preventing extensive rebalancing and allowing effective learning from both classes.
Complex feature interactions: The presence of numerical and categorical features suggests models must handle complex, non-linear relationships. For instance, the interaction between `civilian_casualties_past_year` and `public_approval_rating` could indicate a hero’s alignment.

This rich dataset provides a robust foundation for comparing the performance of k-Nearest Neighbours, Random Forest, and LSTM models on a non-trivial classification task.

2. Environment and Installing required libraries and modules

To run the project file, we need some python packages:

scikit-learn, numpy, pandas, tensorflow, Matplotlib.pyplot, Seaborn

- Pandas is used to handle datasets, load data, clean data, and prepare it for subsequent analysis.
- Numpy is essential for efficient numerical computations, data manipulation, and creating array structures that are crucial for machine learning models.
- Matplotlib.pyplot is a versatile tool used to create both basic and advanced graphs. It excels in visualizing data distributions and model results.
- Seaborn, built on top of Matplotlib, offers a high-level API that simplifies the process of creating visually appealing and statistically informative plots with minimal code and a focus on beautiful styling.
- Scikit-learn provides tools for implementing and evaluating machine learning models, such as k-Nearest Neighbors (kNN) and Random Forest.
- TensorFlow is specifically required to build and train the Long Short-Term Memory (LSTM) deep learning model for classification tasks.

In my files, if you try to run main.ipynb file. I already define the required libraires, so you do not need to install any package just run the cell and you ready to go.

```
%pip install pandas
%pip install numpy
%pip install scikit-learn
%pip install tensorflow
%pip install matplotlib
%pip install seaborn
```

If you are trying to run the main.py file, then followed bellowed the instruction.

1. **First Install Python if you do not have**

To run python file, we need to install python in our system.

- **Windows:** Download the installer from python.org and add Python to your PATH during installation.
- **Mac:** Python is usually pre-installed; for the latest version, use python.org or brew install python.

2. **Set Up a Virtual Environment**

To establish a virtual environment, locate the folder containing all the necessary files. Then, open the terminal within that folder and proceed with the following steps.

- Windows Command Prompt:
  python -m venv venv # create a python environment with name venv
  venv\Scripts\activate # activate the python environment

- Mac / Linux Terminal:
  python3 -m venv venv # create a python environment with name venv
  source venv/bin/activate # activate the python environment

3. **Install Required Python Packages**

For your file, install these libraries:

- pandas
- numpy
- scikit-learn
- Matplotlib
- Seaborn
- Tensorflow

All the required Python packages are listed in the requirements.txt file included with the project. To install these dependencies, run the following command in your terminal or command prompt (in the activated environment), making sure you are in the directory containing requirements.txt (this is uploaded in my GitHub)
pip install -r requirements.txt

This command will automatically install all packages specified in the file.
If the requirements.txt file is not available, you can manually install the necessary packages using:
pip install pandas numpy scikit-learn matplotlib seaborn tensorflow

4. **Prepare Your Data Files**

   Make sure your dataset CSV files (superhero dataset.csv) is in the same directory as your script or update the script with their full paths.

5. **Run the Python Script**

   Run your main file in your terminal.
   Windows:
   python main.py
   Mac:
   python3 main.py

3. Code Structure and Project Workflow Overview

This project is structured to enable users to perform binary classification using multiple machine learning and deep learning algorithms—including k-Nearest Neighbors (kNN), Random Forest, and Long Short-Term Memory (LSTM)—on the superhero dataset.

Workflow:

- **Dataset Loading & Selection:**
  In this section, the code presents the available dataset, loads it using pandas, and performs exploratory data analysis to give an overview of features and target distribution

```
df = pd.read_csv("superhero dataset.csv")
df.head() # print head of uploded dataset
```

	height_cm	weight_kg	age	years_active	training_hours_per_week	civilian_casualties_past_year	power_level	public_approval_rating	super_strength	fligh
0	189.9	81.9	33	10	32.6	2	28.4	62.9	0	
1	177.2	73.6	58	22	41.9	2	67.3	41.4	0	
2	193.0	81.8	47	6	22.3	0	95.9	96.8	0	
3	210.5	88.1	62	26	25.0	4	71.8	53.2	1	
4	175.3	80.7	43	28	24.9	3	81.3	36.1	0	

Loding dataset

- **Preprocessing:**
  Handles missing and duplicate values using pandas' built-in functions.
  Splits data into input features (x label) and target variable (y label).

```
df.describe()
```

	height_cm	weight_kg	age	years_active	training_hours_per_week	civilian_casualties_past_year	power_level	public_approval_rating	sup
count	1200.000000	1200.000000	1200.000000	1200.000000	1200.000000	1200.000000	1200.000000	1200.000000	1
mean	181.217833	80.422250	42.325833	15.095833	29.483500	1.893333	70.182333	64.615833	
std	18.831243	14.616758	11.323641	8.553905	10.117866	1.896915	15.286139	18.543736	
min	150.000000	45.000000	18.000000	0.000000	1.000000	0.000000	12.500000	0.000000	
25%	167.500000	70.375000	35.000000	8.000000	22.800000	1.000000	59.775000	52.650000	
50%	181.000000	80.200000	42.000000	15.000000	29.400000	1.000000	70.200000	64.900000	
75%	193.525000	90.100000	50.000000	22.000000	36.400000	3.000000	81.000000	77.425000	
max	250.000000	127.900000	84.000000	29.000000	61.000000	13.000000	100.000000	100.000000	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1200 entries, 0 to 1199
Data columns (total 17 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   height_cm                      1200 non-null   float64
 1   weight_kg                      1200 non-null   float64
 2   age                            1200 non-null   int64
 3   years_active                   1200 non-null   int64
 4   training_hours_per_week        1200 non-null   float64
 5   civilian_casualties_past_year  1200 non-null   int64
 6   power_level                    1200 non-null   float64
 7   public_approval_rating         1200 non-null   float64
 8   super_strength                 1200 non-null   int64
 9   flight                         1200 non-null   int64
 10  energy_projection              1200 non-null   int64
 11  telepathy                      1200 non-null   int64
 12  healing_factor                 1200 non-null   int64
 13  shape_shifting                 1200 non-null   int64
 14  invisibility                   1200 non-null   int64
 15  telekinesis                    1200 non-null   int64
 16  is_good                        1200 non-null   int64
dtypes: float64(5), int64(12)
memory usage: 159.5 KB
```

```
# Check for missing values and also duplicate value and if any then drop that rows
print(df.isnull().sum())
df = df.dropna() # drop rows which have null values if any
```

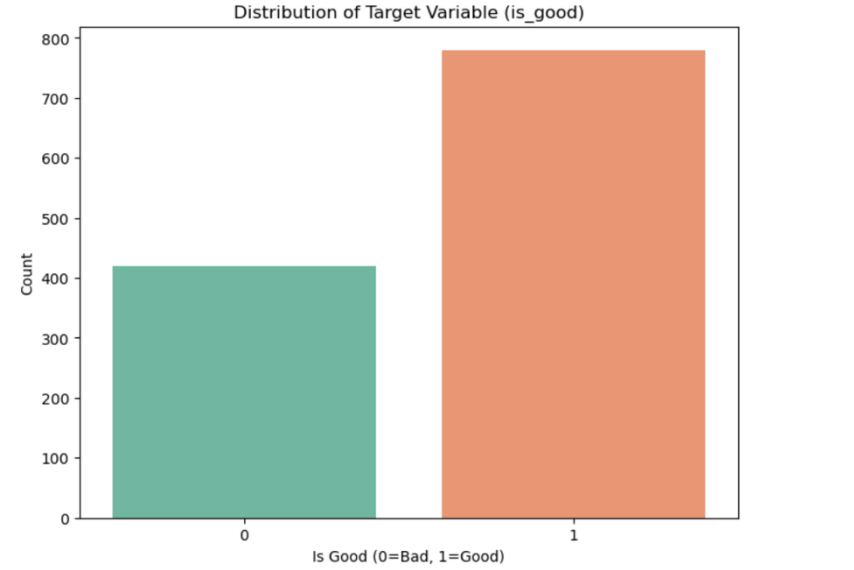
Split data into imput(fectures) and target(lable)

```
x = df.drop('is_good',axis=1)
y= df['is_good']
```

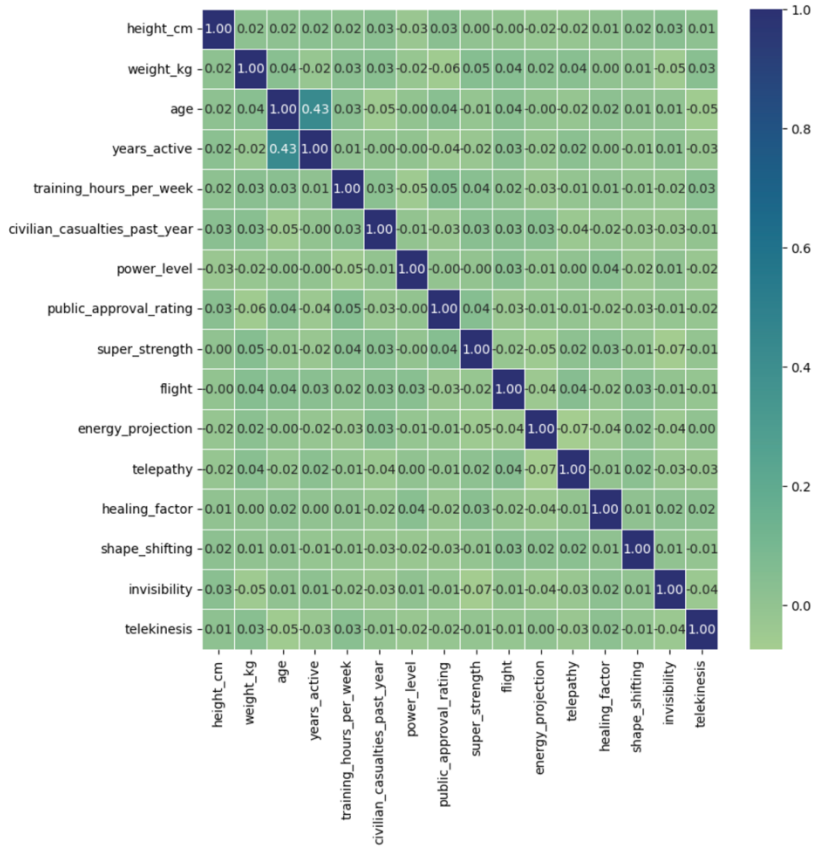
Perform EDA

- **Data Visualization:**
  The project employs matplotlib and seaborn to generate informative visualizations, including variable distributions, correlation heatmaps, and feature pair plots. These visual aids facilitate the easy observation of data spread, the detection of imbalanced classes, and the identification of significant patterns and relationships among features. This crucial step is essential for comprehending the dataset's structure and quality prior to constructing machine learning models. It draws attention to any potential issues (such as class imbalance or outliers) that could potentially impact the model's performance.

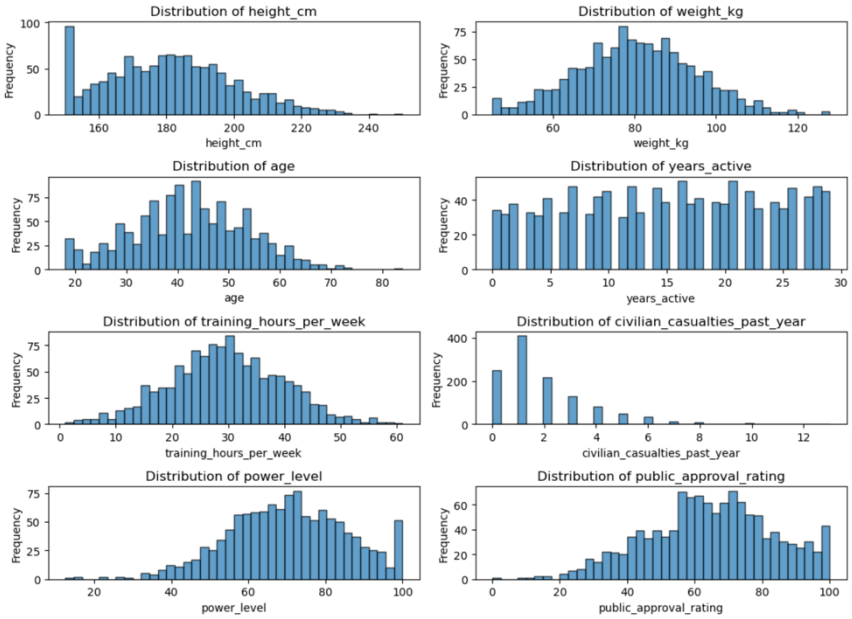
```
plt.figure(figsize=(8, 6))
sns.countplot(x='is_good', data=df,palette='Set2')
plt.title('Distribution of Target Variable (is_good)')
plt.xlabel('Is Good (0=Bad, 1=Good)')
plt.ylabel('Count')
plt.show()
```



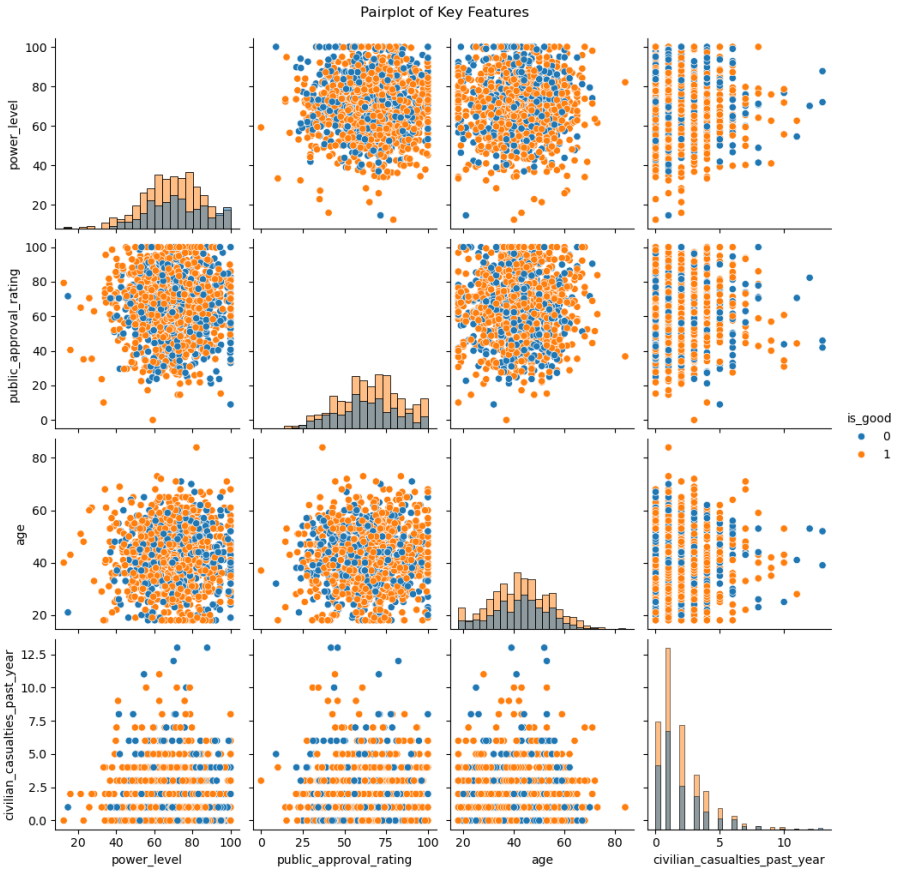
Plotting a bar graph for the target variable to illustrate the data imbalance



Plotting Correlations Among All Factors



Show features and distortions.



Use a pair plot to visualize key features.

- **Feature Standardization**

  Feature standardization, using scikit-learns Standard Scaler, normalized input features to a mean of zero and standard deviation of one. This preprocessing step ensures equal contribution of features, mitigates bias, and expedites model convergence.
  This constitutes one of the fundamental aspects, as with the assistance we receive, we will gain insights into the efficacy of our model.

```
std_scaler = StandardScaler()
normalized_x = std_scaler.fit_transform(x)

print(normalized_x[:1])
```

- **Train/Test Data Split**

  Splitting the dataset into training and testing sets is a fundamental practice in machine learning. The training set is used to train the model, allowing it to identify patterns and relationships in the data. The testing set, which remains unobserved during training, provides a realistic evaluation of the model's generalization capabilities to new, real-world data. This approach helps avoid overfitting and ensures that performance metrics accurately reflect the model's predictive power beyond the training environment. For this project, 80% of the data was used for training, while the remaining 20% was reserved for testing.

```
x_train, x_test, y_train, y_test = train_test_split(normalized_x, y, test_size=0.2, random_state=42)

# Print the shapes of the resulting sets to confirm
print("Training input data shape:", x_train.shape)
print("Testing input data shape:", x_test.shape)
print("Training lable shape:", y_train.shape)
print("Testing lable shape:", y_test.shape)
```

- **Manually Implement Performance Matrix:**

  To objectively assess model performance, the project employs a confusion matrix to categorize predictions into True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). Subsequently, additional crucial metrics are manually computed, including recall (TPR), specificity (TNR), precision, F1-score, accuracy, error rate, True Skill Statistic (TSS), Balanced Accuracy (BACC), Heidke Skill Score (HSS), Brier Score (BS), Brier Skill Score (BSS), and Area Under the ROC Curve (AUC).

  A comprehensive approach evaluates model strengths and weaknesses, providing a deeper understanding of classification effectiveness beyond accuracy.

```python
def cal_performance_metrics(cm, y_test, y_pred, y_prob):
    TN, FP, FN, TP=cm.ravel()   #.ravel() flattens the array into 1D array

    # Calculate metrics
    TPR = TP / (TP + FN) if (TP + FN) != 0 else 0  # True Positive Rate (Recall)
    TNR = TN / (TN + FP) if (TN + FP) != 0 else 0  # True Negative Rate
    FPR = FP / (FP + TN) if (FP + TN) != 0 else 0  # False Positive Rate
    FNR = FN / (FN + TP) if (FN + TP) != 0 else 0  # False Negative Rate
    TSS = TPR - FPR # True Skill Statistic
    Precision = TP / (TP + FP) if (TP + FP) != 0 else 0  # Precision
    f1_score = 2 * TP / (2 * TP + FP + FN) if (TP + FP + FN) != 0 else 0
    Accuracy = (TP + TN) / (TP + TN + FP + FN)  # Accuracy
    Error_rate = 1 - Accuracy  # Error Rate
    BACC = (TPR + TNR) / 2
    HSS = 2 * (TP * TN - FP * FN) / ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN)) if ((TP + FN) * (FN + TN) + (TP + FP) * (FP + TN)) != 0 el
    bs = brier_score_loss(y_test, y_prob)
    bs_ref = brier_score_loss(y_test, np.full_like(y_test, y_test.mean()))
    bss = (bs_ref - bs) / bs_ref if bs_ref != 0 else 0
    auc = roc_auc_score(y_test, y_prob) # Area Under the Curve


    return {
        "TN": TN, "FP": FP, "FN": FN, "TP": TP,
        "TPR": TPR, "TNR": TNR, "FPR": FPR, "FNR": FNR,"TSS":TSS,"HSS" : HSS,"BACC" : BACC,
        "Precision": Precision,"f1_score" : f1_score,  "Accuracy": Accuracy, "Error_rate": Error_rate,
        "BS" : bs, "BSS" : bss, "AUC" : auc
    }
```

Function for Performance Metrics

- **Cross-Validation with 10-Fold Splits**

  To ensure reliable model evaluation and minimize the impact of random data splits, the project employs k-fold cross-validation with 10 folds. In this procedure, the dataset is divided into ten equal parts. For each iteration, the model is trained on nine parts and tested on the remaining part. This cycle repeats ten times, with each fold serving as the test set once. After all iterations, performance metrics are averaged, providing a robust estimate of the model's ability to generalize to unseen data. 10-fold cross-validation reduces bias and variance for accurate model evaluation.

```python
# function for cross validations
def perform_10fold(normalized_x, y, model):
    cross_validation = KFold(n_splits=10, shuffle=True, random_state=42) #KFold() will split the data into 10 parts
    fold_metrics = []

    for fold, (train_index, test_index) in enumerate(cross_validation.split(normalized_x), 1):
        # Split the data
        X_train, X_test = normalized_x[train_index], normalized_x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Train and predict
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_prob = model.predict_proba(X_test)[:, 1]

        # Calculate confusion matrix and metrics
        cm = confusion_matrix(y_test, y_pred)
        # fpr, tpr, _ = roc_curve(y_test, y_prob)
        auc = roc_auc_score(y_test, y_prob)
        metrics = cal_performance_metrics(cm,y_test,y_pred,y_prob)
        metrics['Fold'] = fold
        fold_metrics.append(metrics)


    return fold_metrics
```

Cross-validation with k = 10

- **Implementing Machine Learning Models and Execution:**

1. K-Nearest Neighbors (KNN) Implementation

K-Nearest Neighbors (KNN) is a supervised classification algorithm that predicts a class based on the majority class among the k closest neighbors. The project employs scikit-learn's KNeighbors Classifier and 10-fold cross-validation to assess the model's performance.

The output of the K-Nearest Neighbors (KNN) binary classification algorithm using the loaded dataset is:

KNN Algoritham

	0	1	2	3	4	5	6	7	8	9
TN	5.000	5.000	7.000	5.000	4.000	8.000	4.000	5.000	3.000	11.000
FP	40.000	38.000	38.000	41.000	35.000	27.000	42.000	27.000	39.000	36.000
FN	8.000	7.000	7.000	4.000	15.000	7.000	5.000	4.000	11.000	8.000
TP	67.000	70.000	68.000	70.000	66.000	78.000	69.000	84.000	67.000	65.000
TPR	0.893	0.909	0.907	0.946	0.815	0.918	0.932	0.955	0.859	0.890
TNR	0.111	0.116	0.156	0.109	0.103	0.229	0.087	0.156	0.071	0.234
FPR	0.889	0.884	0.844	0.891	0.897	0.771	0.913	0.844	0.929	0.766
FNR	0.107	0.091	0.093	0.054	0.185	0.082	0.068	0.045	0.141	0.110
TSS	0.004	0.025	0.062	0.055	−0.083	0.146	0.019	0.111	−0.070	0.124
HSS	0.005	0.030	0.072	0.064	−0.095	0.176	0.023	0.144	−0.082	0.139
BACC	0.502	0.513	0.531	0.527	0.459	0.573	0.510	0.555	0.465	0.562
Precision	0.626	0.648	0.642	0.631	0.653	0.743	0.622	0.757	0.632	0.644
f1_score	0.736	0.757	0.751	0.757	0.725	0.821	0.746	0.844	0.728	0.747
Accuracy	0.600	0.625	0.625	0.625	0.583	0.717	0.608	0.742	0.583	0.633
Error_rate	0.400	0.375	0.375	0.375	0.417	0.283	0.392	0.258	0.417	0.367
BS	0.234	0.231	0.232	0.242	0.246	0.215	0.239	0.194	0.230	0.237
BSS	0.625	0.640	0.629	0.608	0.636	0.697	0.612	0.736	0.647	0.610
AUC	0.584	0.574	0.592	0.556	0.456	0.576	0.577	0.634	0.589	0.569
Fold	1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000

Average result of K-Nearest Neighbors (KNN):

	Avarage(Mean)
TN	5.700
FP	36.300
FN	7.600
TP	70.400
TPR	0.902
TNR	0.137
FPR	0.863
FNR	0.098
TSS	0.040
HSS	0.048
BACC	0.520
Precision	0.660
f1_score	0.761
Accuracy	0.634
Error_rate	0.366
BS	0.230
BSS	0.644
AUC	0.571
Fold	5.500

2. Random Forest Implementation

Random Forest, a powerful ensemble method implemented using scikit-learn, constructs multiple decision trees and computes their average outputs. It excels in handling diverse data types, provides feature importance evaluation, and is less susceptible to overfitting.

The output of the Random Forest binary classification algorithm using the loaded dataset is:

Random Forest Alogoritham

	0	1	2	3	4	5	6	7	8	9
TN	7.000	6.000	5.000	8.000	9.000	5.000	4.000	6.000	7.000	6.000
FP	38.000	37.000	40.000	38.000	30.000	30.000	42.000	26.000	35.000	41.000
FN	5.000	8.000	4.000	7.000	10.000	11.000	6.000	5.000	4.000	3.000
TP	70.000	69.000	71.000	67.000	71.000	74.000	68.000	83.000	74.000	70.000
TPR	0.933	0.896	0.947	0.905	0.877	0.871	0.919	0.943	0.949	0.959
TNR	0.156	0.140	0.111	0.174	0.231	0.143	0.087	0.188	0.167	0.128
FPR	0.844	0.860	0.889	0.826	0.769	0.857	0.913	0.812	0.833	0.872
FNR	0.067	0.104	0.053	0.095	0.123	0.129	0.081	0.057	0.051	0.041
TSS	0.089	0.036	0.058	0.079	0.107	0.013	0.006	0.131	0.115	0.087
HSS	0.104	0.042	0.069	0.091	0.124	0.016	0.007	0.165	0.139	0.101
BACC	0.544	0.518	0.529	0.540	0.554	0.507	0.503	0.565	0.558	0.543
Precision	0.648	0.651	0.640	0.638	0.703	0.712	0.618	0.761	0.679	0.631
f1_score	0.765	0.754	0.763	0.749	0.780	0.783	0.739	0.843	0.791	0.761
Accuracy	0.642	0.625	0.633	0.625	0.667	0.658	0.600	0.742	0.675	0.633
Error_rate	0.358	0.375	0.367	0.375	0.333	0.342	0.400	0.258	0.325	0.367
BS	0.230	0.237	0.221	0.228	0.222	0.204	0.242	0.199	0.230	0.220
BSS	0.632	0.631	0.647	0.630	0.671	0.712	0.608	0.728	0.647	0.638
AUC	0.595	0.524	0.638	0.615	0.571	0.635	0.530	0.621	0.551	0.676
Fold	1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000

Average result of Random Forest:

	Avarage(Mean)
TN	6.300
FP	35.700
FN	6.300
TP	71.700
TPR	0.920
TNR	0.152
FPR	0.848
FNR	0.080
TSS	0.072
HSS	0.086
BACC	0.536
Precision	0.668
f1_score	0.773
Accuracy	0.650
Error_rate	0.350
BS	0.223
BSS	0.654
AUC	0.596
Fold	5.500

3. LSTM (Long Short-Term Memory) Implementation

LSTM networks, designed for sequential data, are adapted for tabular data using TensorFlow/Keras. Despite LSTM's strengths, Random Forest outperforms it on this dataset due to data structure and overfitting concerns.

The output of the LSTM binary classification algorithm using the loaded dataset is:

	0	1	2	3	4	5	6	7	8	9
TN	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
FP	23.000	31.000	39.000	30.000	32.000	38.000	31.000	27.000	33.000	34.000
FN	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TP	64.000	56.000	48.000	57.000	54.000	48.000	55.000	59.000	53.000	52.000
TPR	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
TNR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
FPR	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
FNR	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
TSS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
HSS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
BACC	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
Precision	0.736	0.644	0.552	0.655	0.628	0.558	0.640	0.686	0.616	0.605
f1_score	0.848	0.783	0.711	0.792	0.771	0.716	0.780	0.814	0.763	0.754
Accuracy	0.736	0.644	0.552	0.655	0.628	0.558	0.640	0.686	0.616	0.605
Error_rate	0.264	0.356	0.448	0.345	0.372	0.442	0.360	0.314	0.384	0.395
BS	0.208	0.229	0.257	0.226	0.234	0.255	0.230	0.222	0.238	0.240
BSS	0.717	0.645	0.534	0.655	0.627	0.543	0.640	0.677	0.614	0.603
AUC	0.442	0.554	0.423	0.529	0.461	0.463	0.521	0.411	0.462	0.503
Fold	1.000	2.000	3.000	4.000	5.000	6.000	7.000	8.000	9.000	10.000

Average result of LSTM:

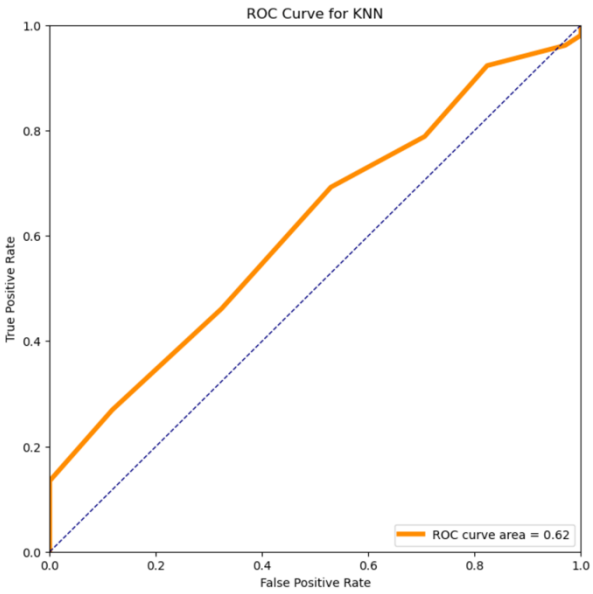
	Average(Mean)
TN	0.000
FP	31.800
FN	0.000
TP	54.600
TPR	1.000
TNR	0.000
FPR	1.000
FNR	0.000
TSS	0.000
HSS	0.000
BACC	0.500
Precision	0.632
f1_score	0.773
Accuracy	0.632
Error_rate	0.368
BS	0.234
BSS	0.625
AUC	0.477
Fold	5.500

Comparing all the Averages of Algorithms

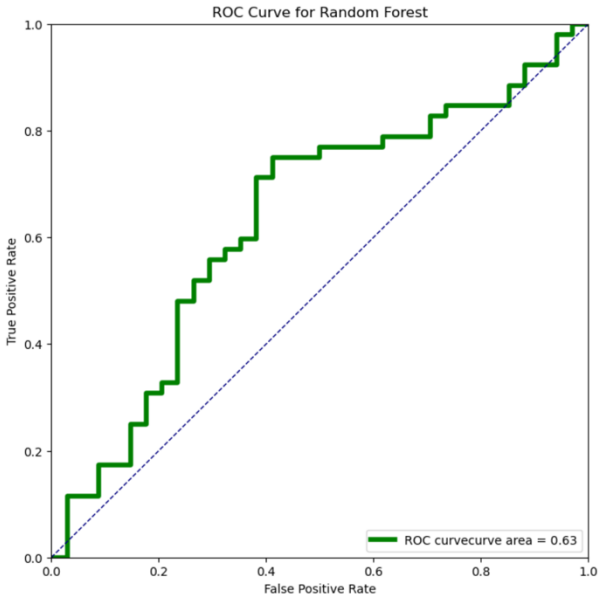
Comparing the average for all Algorithms

	kNN average	Random Forest average	LSTM average
TN	5.700000	6.300000	0.000000
FP	36.300000	35.700000	31.800000
FN	7.600000	6.300000	0.000000
TP	70.400000	71.700000	54.600000
TPR	0.902386	0.919836	1.000000
TNR	0.137145	0.152252	0.000000
FPR	0.862855	0.847748	1.000000
FNR	0.097614	0.080164	0.000000
TSS	0.039532	0.072089	0.000000
HSS	0.047592	0.085776	0.000000
BACC	0.519766	0.536044	0.500000
Precision	0.659680	0.668051	0.631877
f1_score	0.761304	0.772854	0.773167
Accuracy	0.634167	0.650000	0.631877
Error_rate	0.365833	0.350000	0.368123
BS	0.230026	0.223252	0.233940
BSS	0.643790	0.654395	0.625416
AUC	0.570702	0.595667	0.476879
Fold	5.500000	5.500000	5.500000

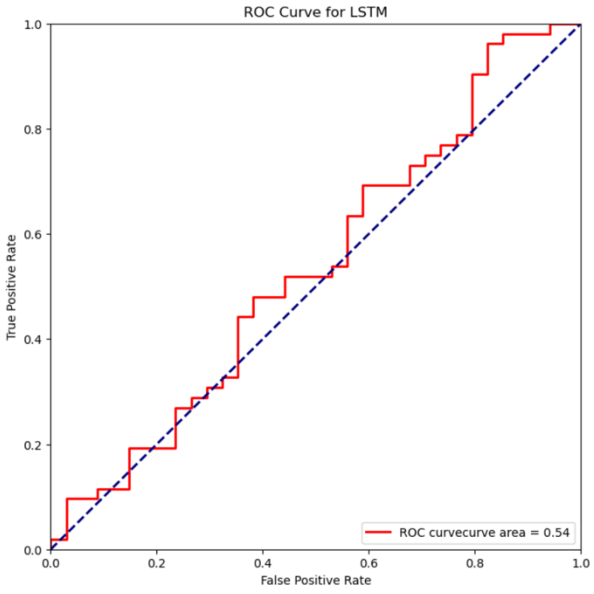
ROC curves for all:



KNN



Random Forest



LSTM

Findings:

True Positives (TP) and True Negatives (TN)

- Random Forest outperforms kNN and LSTM in identifying positive cases. It achieves the highest average True Positives (approximately 71.7), indicating accurate classification of more positive instances. kNN slightly edges out Random Forest in True Negatives (around 5.7), while LSTM struggles to correctly identify negative cases. It's True Negative rate is nearly zero, suggesting frequent mislabeling of negative instances.

False Positives (FP) and False Negatives (FN)

- LSTM has the highest false positive rate (approximately 34.4%), leading to incorrect predictions of numerous negative cases as positive. Random Forest and kNN also show significant false positives, although Random Forest achieves the lowest false negative rate (around 6.3), which is crucial for tasks where missing true positive cases would be costly or critical. kNN slightly outperforms Random Forest in terms of false negative rate.

True Positive Rate (TPR) and True Negative Rate (TNR)

- LSTM achieves a perfect TPR (1.0), capturing all actual positive instances in the data. However, it severely suffers from a low TNR (0), significantly impacting its overall accuracy. Random Forest strikes a better balance by achieving a TPR of around 0.92 and showing a slight improvement in TNR compared to kNN.

Precision and Accuracy

- Random Forest outperforms other models in terms of precision (approximately 0.67), which means that among the cases it predicts as positive, a higher percentage of them are actually positive compared to the other models. Additionally, it achieves a higher overall accuracy (approximately 65%), indicating that it makes more accurate predictions than both LSTM and kNN.

Other metrics (F1, Balanced Accuracy, Brier Score, etc.)

- All models exhibit comparable F1 Scores, with LSTM slightly outperforming others, suggesting a trade-off between precision and recall. Balanced accuracy and error rates indicate that Random Forest is the most stable across both positive and negative classes.

Why Random Forest Performs Better in This Case

The superhero dataset encompasses a diverse array of data types, including both continuous features like height, weight, and training hours and categorical features like powers and abilities. Random Forest emerges as an effective algorithm for handling such mixed data environments. It excels in managing various feature types, doesn't require extensive data scaling, and demonstrates resilience against overfitting due to its ensemble of decision trees.

In contrast:

- kNN's reliance on distance metrics and feature scaling makes it less effective with widely varying features or high dimensionality.
- LSTM, specifically designed for sequential data like time series or natural language, is not well-suited for this tabular superhero dataset. It tends to overfit rapidly without proper tuning and performs poorly on negative cases, as evidenced by its low TNR and high false positives.

Results, Challenges and Future Work

The comparative evaluation of KNN, Random Forest, and LSTM models showed clear differences in their ability to handle the superhero classification task. Random Forest achieved the highest overall accuracy and precision, indicating its strong performance in correctly identifying positive outcomes and minimizing false alarms. KNN produced moderate results, balancing both positives and negatives, but was more affected by class imbalance and feature scaling. LSTM excelled in identifying all positive cases (perfect recall) but struggled with negative instances, leading to many false positives and a lower specificity. Random Forest outperforms other models in accuracy, precision, recall, F1-score, balanced accuracy, and AUC, with the largest ROC curve area.

Throughout this project, several technical and analytical challenges were encountered. First, managing class imbalance in the dataset made it difficult for some models, especially LSTM, to correctly identify negative cases, leading to poor specificity despite high recall. Feature scaling was crucial for distance-based models like KNN, requiring careful preprocessing to ensure fair comparison across methods. Implementing and tuning the LSTM for non-sequential, tabular data also proved challenging, as overfitting and unstable results occurred without extensive parameter adjustment. Selecting optimal hyperparameters and avoiding information leakage during cross-validation required careful attention to experimental design and reproducibility.

Future improvements should concentrate on enhancing recall and minimizing bias by employing oversampling techniques like SMOTE or ADASYN. Additionally, testing sophisticated LSTM architectures and optimizing hyperparameters can further enhance accuracy. Furthermore, tools like SHAP or LIME are suggested to simplify model predictions and facilitate interpretation and explanation.

Conclusion

This project successfully implemented and compared three binary classification algorithms: K-Nearest Neighbors (KNN), Random Forest, and Long Short-Term Memory (LSTM) on the superhero dataset. Among the tested models, Random Forest consistently delivered the best overall performance, achieving the highest accuracy, precision, and balanced metrics across all cross-validation folds. KNN provided reasonable results but was more susceptible to feature scaling and class imbalance, while LSTM excelled in recall but required further tuning to enhance its generalization on tabular data. These findings emphasize the advantages of ensemble methods such as Random Forest for this type of classification task and underscore the significance of careful model selection, preprocessing, and evaluation when applying machine learning techniques to real-world datasets.

GitHub Repository

The complete source code, datasets, Python file and a Jupyter notebook documenting the process are available on my Github repository. The repository link is provided below:

Note: I used my personal GitHub account (instead of an NJIT email account).

The repository link is provided below:
https://github.com/KenilAvaiyaa/Avaiya_Kenil_finaltermproj

Appendices

- The project code and all results are available in the attached Jupyter notebook file (.ipynb). Additionally, you can execute the standalone Python file (.py) to access the same results.
- Main Python package requirements are listed in requirements.txt.
- Extra screenshots and figures are provided as images in this the report.

References

- https://scikit-learn.org/stable/modules/neighbors.html
- https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/
- https://www.geeksforgeeks.org/machine-learning/auc-roc-curve/
- https://www.kaggle.com/code/ryanholbrook/binary-classification