Titanic Survival Prediction Using Machine Learning

1. Introduction

The RMS Titanic's sinking on April 15, 1912, remains a significant maritime disaster, with over 1,500 lives lost. Recent advances in machine learning allow for new insights into the factors that influenced passenger survival.

This project uses machine learning to predict Titanic passenger survival based on demographic and socio-economic factors. Applying machine learning to historical data like the Titanic's passenger list exemplifies how advanced analytics can reveal new insights into well-known events, helping inform future decisions. Factors like age, gender, and socio-economic status, key to Titanic survival, remain relevant in modern disaster risk assessments. Accurate predictive models could help identify high-risk groups in emergencies, improving safety measures and resource allocation.

This project demonstrates the potential of machine learning in historical analysis and modern safety applications, aiming to both predict survival and gain valuable insights.

2. Problem Formulation

2.1 Machine Learning Problem

We formalize the Titanic survival prediction as a supervised binary classification task. The objective is to develop a model that predicts whether a passenger survived the Titanic disaster based on specific features. This problem is suitable for classification algorithms such as logistic regression and k-nearest neighbors (KNN).

2.2 Data Points, Features, and Labels

Each data point represents an individual passenger from the Titanic. After preprocessing, the dataset comprises 889 passengers, each described by a set of features and associated with a binary label indicating survival.

Features:

- *Pclass*: Passenger ticket class (1st, 2nd, or 3rd)
- *Sex*: Gender (Male or Female)
- *Age*: Age in years
- *SibSp*: Number of siblings/spouses aboard
- *ParCh*: Number of parents/children aboard
- *Fare*: Ticket fare

Label:

- *Survived*: Binary variable where 1 indicates survival and 0 indicates non-survival

Pclass and *Fare* are related to the economic status of the passenger (what class they are traveling in and how much they can afford to pay for a ticket). *Sex* and *Age* relate to the physical attributes of the passenger, such as strength and endurance. *SibSp* and *ParCh* can affect the passengers mental state and willpower. *Hometown, Destination* and *Embarked* are excluded since the task is classification, and these columns have too much variance. *Cabin*, *Ticket*, *Body*, and *Lifeboat* are excluded since there is not enough data. The rest are excluded due to redundancy.

This binary classification setup is appropriate for algorithms like logistic regression, which models the probability of survival, and KNN, which classifies based on similarities among passengers.

2.3 Dataset Source

The dataset used is the Titanic extended dataset by Fesenko (2019), available on Kaggle. It provides a comprehensive view of Titanic passengers, including additional features that may offer deeper insights into survival factors. The dataset contains both categorical and numerical features, necessitating careful preprocessing to handle missing values and prepare the data for modeling.

2.4 Suitability of Classification Methods

Predicting passenger survival on the Titanic is inherently a binary classification problem. Logistic regression is suitable because it models the probability of a binary outcome and provides interpretable coefficients indicating the influence of each feature. KNN is appropriate as it makes no assumptions about data distribution and can capture non-linear relationships by classifying based on the proximity of data points in feature space.

3. Methods

3.1 Dataset Description and Preprocessing

The preprocessed dataset consists of 889 passengers. We take the following steps for preprocessing:

1. Handling Missing Values:

- Dropped rows with missing values in essential columns to ensure data completeness.
- Imputed missing *Age* values with the median age to preserve the central tendency.

2. Data Cleaning:

- Removed irrelevant columns (*Id, WikiId*, *Name_wiki*, *Age_wiki*, *Embarked*, *Ticket*, *Cabin*, *Body*, *Lifeboat*, *Hometown*, *Class*, *Destination* etc.) to streamline the dataset. Some were dropped due to redundancy, and others since there was not enough data.

The final dataset contains 11 features and is ready for model training.

3.2 Feature Selection

Features were selected based on their expected relevance to survival. Features with high missing values or low relevance were excluded to improve model performance.

3.3 Model Selection

Two models were selected for this classification task:

- *Logistic Regression*: Chosen for its ability to model binary outcomes and provide interpretable coefficients that indicate the impact of each feature on survival probability.
- *K-Nearest Neighbors (KNN)*: Selected to capture potential non-linear relationships by classifying passengers based on the characteristics of their nearest neighbors in the feature space.

3.4 Loss Functions

In Logistic Regression, we utilize the logistic loss function, which penalizes misclassifications and is suitable for probability estimation in binary outcomes. KNN is evaluated using classification accuracy, corresponding to the 0-1 loss function, which measures the proportion of incorrect predictions.

3.5 Model Validation

We chose a three-way split of the data: Training set (70%) which was used to train the models, Validation set (15%), used for hyperparameter tuning and model selection, and Test set (15%), used for final evaluation of the chosen model.

According to Xu and Goodacre (2018), a 70/30 split is a practice that offers a practical trade-off between training and testing needs. We further split the 30% into two equally sized validation and test sets to ensure a balance between them.

This split ensures that we have enough data to train the models effectively, while still retaining substantial portions for validation (model selection) and unbiased final testing. The validation set allows us to tune hyperparameters (particularly important for KNN) without overfitting to the test set, while the test set provides an unbiased estimate of the final model's performance on unseen data.

For the models, we use the following:

1. Log Loss: Used during training of the Logistic Regression model. It penalizes confident misclassifications more heavily, encouraging the model to output well-calibrated probabilities.

2. Accuracy: Used for both models, it measures the proportion of correct predictions (both true positives and true negatives) among the total number of cases examined. While simple, it provides an intuitive measure of model performance.
3. Confusion Matrix: Used to provide a more detailed breakdown of model performance, showing true positives, true negatives, false positives, and false negatives.
4. Classification Report: Provides precision, recall, and F1-score for each class, offering a more comprehensive view of model performance, especially useful for imbalanced datasets.

For the KNN model, we performed hyperparameter tuning by testing different values of k (number of neighbors) on the validation set to find the optimal value.

4. Results and Model comparison

4.1 Model Performance Comparison

Both Logistic Regression and KNN models were trained on the training set and evaluated on the validation set. Here are the results:

Logistic Regression:

- Validation Log Loss: 0.3804
- Validation Accuracy: 0.8582

K-Nearest Neighbors:

- Best k: 27
- Validation Accuracy: 0.8731

The KNN model slightly outperformed the Logistic Regression model on the validation set in terms of accuracy. However, the difference is relatively small (about 1.5 percentage points).

Based on the validation set performance, we selected the KNN model with k=27 as our final model. This model was then evaluated on the test set, which had been held out during the entire model selection process.

4.2 Model Interpretation

The KNN model achieved an accuracy of 77.61% on the test set with k=27, which is a decent performance given the complexity of the problem and the limited features available.

From the confusion matrix, we can see that:

- True Negatives (correctly predicted non-survivors): 70
- False Positives (incorrectly predicted survivors): 13
- False Negatives (incorrectly predicted non-survivors): 17
- True Positives (correctly predicted survivors): 34

The model seems to be slightly better at predicting non-survivors (84% recall for class 0) than survivors (67% recall for class 1). This could be due to the imbalance in the dataset, with more non-survivors than survivors.

The precision for predicting non-survivors (0.80) is higher than for survivors (0.72), indicating that when the model predicts a passenger did not survive, it's correct 80% of the time, whereas when it predicts survival, it's correct 72% of the time.

5. Conclusion

This project aimed to predict Titanic passenger survival using machine learning techniques. We compared two models: Logistic Regression and K-Nearest Neighbors. The KNN model slightly outperformed Logistic Regression on the validation set and was chosen as the final model.

The final KNN model achieved an accuracy of 77.61% on the test set, demonstrating its ability to predict survival with reasonable accuracy given the available features. The model showed better performance in predicting non-survival compared to survival, which could be attributed to the imbalance in the dataset.

In conclusion, this project demonstrates the potential of machine learning in historical analysis and highlights its possible applications in modern safety planning. While our model provides valuable insights, there's still much to explore in this rich dataset and problem domain.

References

Fesenko, P. (2019) *Titanic Extended Dataset (Kaggle + Wikipedia)* [Dataset]. Kaggle. [online] Available at: https://www.kaggle.com/datasets/pavlofesenko/titanic-extended (Accessed: 19 September 2024).

Xu, Y., & Goodacre, R. (2018). *On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning*. Journal of Analysis and Testing, 2(3), 249–262. Available at: https://link.springer.com/article/10.1007/s41664-018-0068-2 (Accessed: 19 September 2024).

Appendix

Use of Generative AI Tools

We utilized generative AI tools, specifically ChatGPT by OpenAI and Claude by Anthropic, throughout this project to enhance various aspects of our work. During ideation, these tools helped us refine our focus on the Titanic survival prediction problem. They provided insights into selecting appropriate machine learning methods like logistic regression and KNN, and offered best practices for implementation and validation. In choosing the dataset, the AI tools guided us to the Titanic extended dataset by Fesenko (2019), ensuring a comprehensive analysis with enriched features. For writing and structuring the paper, ChatGPT and Claude assisted in organizing the content and refining the language, enabling us to present complex information clearly and concisely. We integrated the AI-generated content responsibly, reviewing all material to ensure it aligned with academic integrity standards.

titanic

October 9, 2024

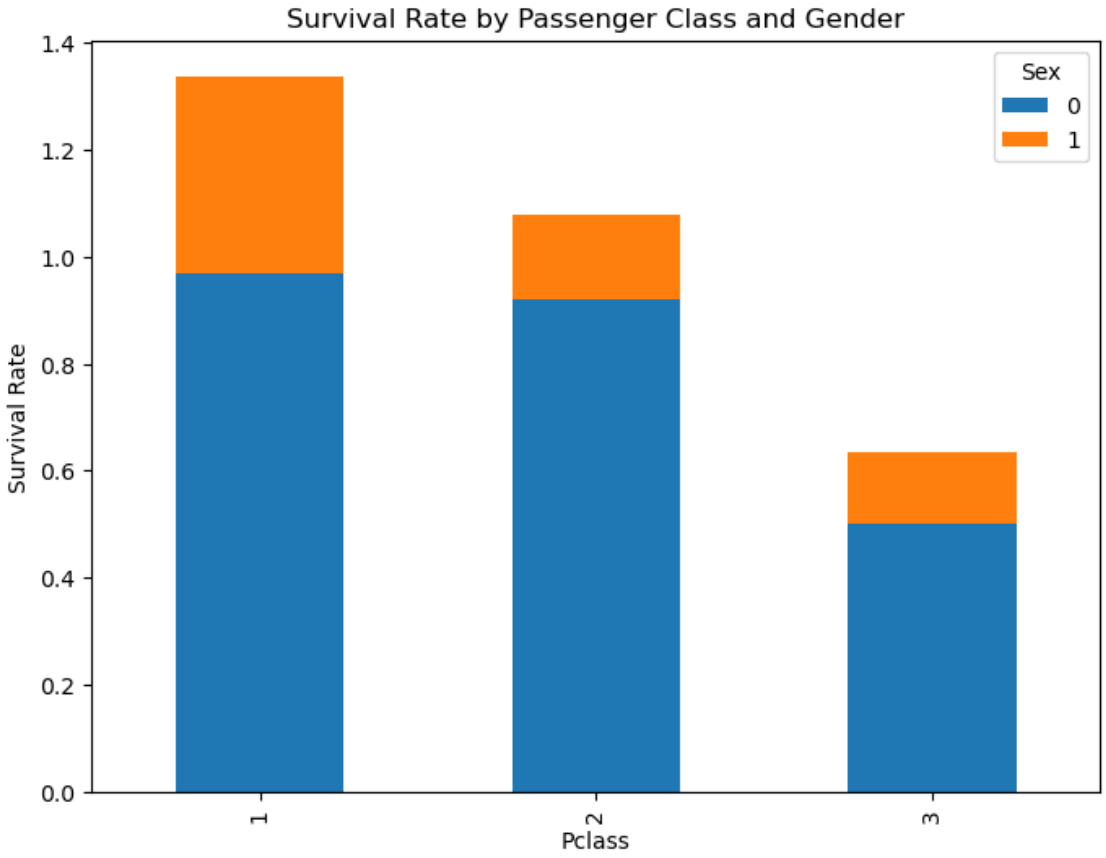
[1]:
```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report, log_loss
```

[2]:
```
# Load the Titanic dataset
df = pd.read_csv('full.csv')

# Drop rows with missing values in essential columns
essential_columns = ['Pclass', 'Sex', 'SibSp', 'Parch', 'Fare', 'Survived']
df = df.dropna(subset=essential_columns)

# Fill missing values in the 'Age' column with the median age
df['Age'] = df['Age'].fillna(df['Age'].median())

# Drop irrelevant columns that are not needed for the model
df.drop(columns=[
    'WikiId', 'Name_wiki', 'Age_wiki', 'Embarked',
    'Ticket', 'Cabin', 'Body', 'Lifeboat',
    'Hometown', 'Class', 'Name', 'Boarded',
    'Destination', 'PassengerId'
], inplace=True)

# Encode categorical variable 'Sex'
df['Sex'] = df['Sex'].map({'male': 1, 'female': 0})

# Rename columns for clarity and consistency
df.columns = ['Survived', 'Pclass', 'Sex',
              'Age', 'SibSp', 'ParCh', 'Fare']
```

```
# Verify that there are no missing values in the dataset
print("Missing values per column:\n", df.isna().any())

# Print the shape of the dataset (number of rows and columns)
print("\nDataset shape:", df.shape)

# Display the binarized dataframe
print(df.head())
```

```
Missing values per column:
 Survived    False
Pclass      False
Sex         False
Age         False
SibSp       False
ParCh       False
Fare        False
dtype: bool

Dataset shape: (891, 7)
   Survived  Pclass  Sex   Age  SibSp  ParCh     Fare
0       0.0       3    1  22.0      1      0   7.2500
1       1.0       1    0  38.0      1      0  71.2833
2       1.0       3    0  26.0      0      0   7.9250
3       1.0       1    0  35.0      1      0  53.1000
4       0.0       3    1  35.0      0      0   8.0500
```

[3]:
```
# Group by Pclass and Sex, and calculate the survival rate
survival_rates = df.groupby(['Pclass', 'Sex'])['Survived'].mean().unstack()

# Plot the survival rates
survival_rates.plot(kind='bar', stacked=True, figsize=(8, 6))

# Add labels and title
plt.ylabel('Survival Rate')
plt.title('Survival Rate by Passenger Class and Gender')

# Display the plot
plt.show()
```



[4]:
```
# Define features and target variable
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'ParCh', 'Fare']
X = df[features]
y = df['Survived']

# Split the data into training (70%), validation (15%), and test (15%) sets
# First, split into training and temp sets
X_train, X_temp, y_train, y_temp = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y)

# Then, split the temp set equally into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.50, random_state=42, stratify=y_temp)

# Print the sizes of the datasets
print("\nDataset sizes:")
print("Training set size:", X_train.shape)
print("Validation set size:", X_val.shape)
print("Test set size:", X_test.shape)
```

```
Dataset sizes:
Training set size: (623, 6)
Validation set size: (134, 6)
Test set size: (134, 6)
```

[5]:
```python
# Feature Scaling
scaler = StandardScaler()

# Fit the scaler on the training data
X_train_scaled = scaler.fit_transform(X_train)

# Apply the transformation to validation and test data
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

[6]:
```python
# Logistic Regression
print("\nTraining Logistic Regression model...")
log_reg = LogisticRegression(solver='liblinear')
log_reg.fit(X_train_scaled, y_train)

# Predict probabilities and compute log loss on the validation set
y_val_pred_proba = log_reg.predict_proba(X_val_scaled)[:, 1]
log_loss_val_logreg = log_loss(y_val, y_val_pred_proba)

# Predict classes and compute accuracy on the validation set
y_val_pred_logreg = log_reg.predict(X_val_scaled)
accuracy_val_logreg = accuracy_score(y_val, y_val_pred_logreg)

print("Logistic Regression Validation Log Loss:", log_loss_val_logreg)
print("Logistic Regression Validation Accuracy:", accuracy_val_logreg)
```

```
Training Logistic Regression model…
Logistic Regression Validation Log Loss: 0.38050430871629354
Logistic Regression Validation Accuracy: 0.8507462686567164
```

[7]:
```python
# K-Nearest Neighbors
print("\nHyperparameter tuning for KNN...")
k_range = range(1, 31)
val_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_val_pred = knn.predict(X_val_scaled)
    accuracy = accuracy_score(y_val, y_val_pred)
```

```
    val_scores.append(accuracy)

# Find the best k
best_k = k_range[np.argmax(val_scores)]
best_accuracy_knn = max(val_scores)
print("Best k for KNN:", best_k)
print("KNN Validation Accuracy with best k:", best_accuracy_knn)

# Retrain KNN with the best k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train_scaled, y_train)

# Predict and compute accuracy on the validation set
y_val_pred_knn = knn_best.predict(X_val_scaled)
accuracy_val_knn = accuracy_score(y_val, y_val_pred_knn)

print("KNN Validation Accuracy:", accuracy_val_knn)
```

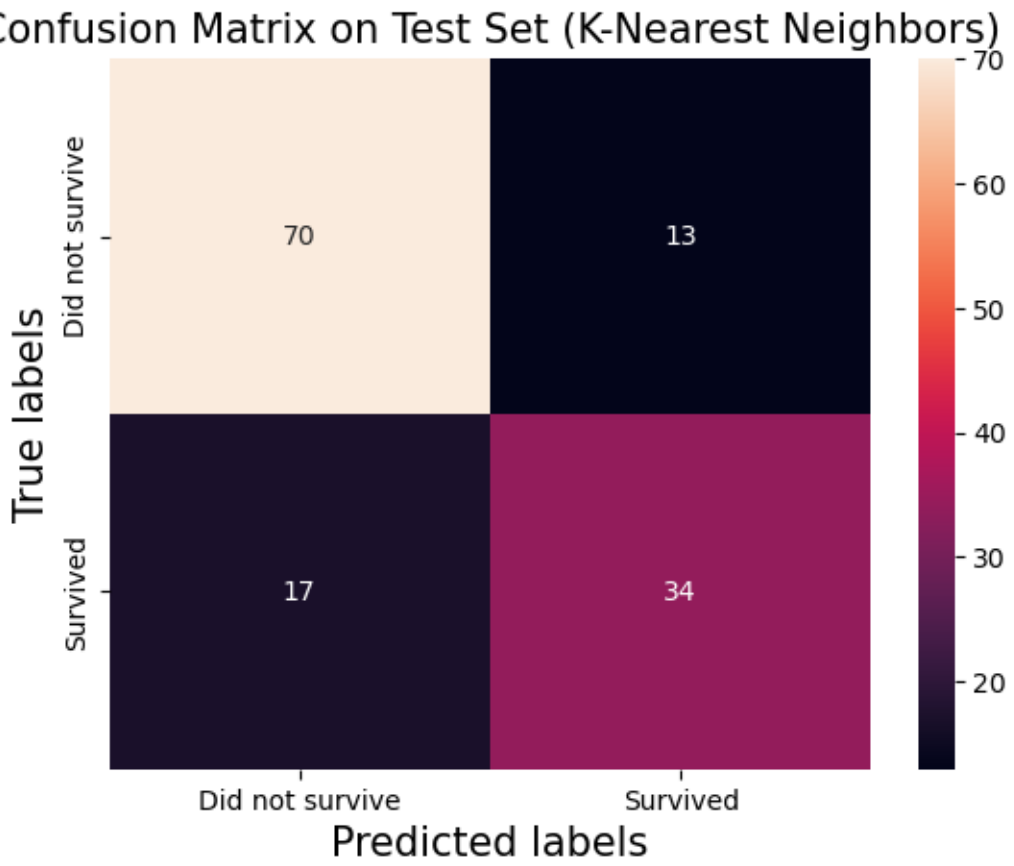
```
Hyperparameter tuning for KNN…
Best k for KNN: 27
KNN Validation Accuracy with best k: 0.8731343283582089
KNN Validation Accuracy: 0.8731343283582089
```

[8]:
```
# Compare models based on validation performance
print("\nComparing models based on validation accuracy...")
if accuracy_val_logreg > accuracy_val_knn:
    print("Logistic Regression performs better on the validation set.")
    best_model = log_reg
    model_name = 'Logistic Regression'
else:
    print("KNN performs better on the validation set.")
    best_model = knn_best
    model_name = 'K-Nearest Neighbors'

# Evaluate the chosen model on the test set
print(f"\nEvaluating the {model_name} model on the test set...")
if model_name == 'Logistic Regression':
    y_test_pred_proba = best_model.predict_proba(X_test_scaled)[:, 1]
    log_loss_test = log_loss(y_test, y_test_pred_proba)
    y_test_pred = best_model.predict(X_test_scaled)
    accuracy_test = accuracy_score(y_test, y_test_pred)
    print("Test Log Loss:", log_loss_test)
    print("Test Accuracy:", accuracy_test)
else:
    y_test_pred = best_model.predict(X_test_scaled)
    accuracy_test = accuracy_score(y_test, y_test_pred)
```

```
    print("Test Accuracy:", accuracy_test)

# Compute confusion matrix and classification report
conf_mat_test = confusion_matrix(y_test, y_test_pred)
print("\nConfusion Matrix on Test Set:\n", conf_mat_test)
print("\nClassification Report on Test Set:\n", classification_report(y_test,␣
 ↪y_test_pred))
```

```
Comparing models based on validation accuracy…
KNN performs better on the validation set.

Evaluating the K-Nearest Neighbors model on the test set…
Test Accuracy: 0.7761194029850746

Confusion Matrix on Test Set:
 [[70 13]
 [17 34]]

Classification Report on Test Set:
              precision    recall  f1-score   support

         0.0       0.80      0.84      0.82        83
         1.0       0.72      0.67      0.69        51

    accuracy                           0.78       134
   macro avg       0.76      0.76      0.76       134
weighted avg       0.77      0.78      0.77       134
```

[9]:
```
# Visualize the confusion matrix
ax = plt.subplot()
sns.heatmap(conf_mat_test, annot=True, fmt='g')
ax.set_xlabel('Predicted labels',fontsize=15)
ax.set_ylabel('True labels',fontsize=15)
ax.set_title(f'Confusion Matrix on Test Set ({model_name})',fontsize=15)
ax.xaxis.set_ticklabels(['Did not survive', 'Survived'])
ax.yaxis.set_ticklabels(['Did not survive', 'Survived'])

plt.show()
```


Confusion Matrix on Test Set (K-Nearest Neighbors)
True labels
Did not survive
Survived
70
13
17
34
Did not survive
Survived
Predicted labels
70
60
50
40
30
20

[ ]: