CPEN 355 Final Project: Titanic Passenger Survival Prediction

Aloysio Kouzak Campos da Paz
Student ID 58687526

Collaboration statement I completed the assignment with the help of the internet and ChatGPT for understanding the concepts behind the assignment and for generating Python code.

Here is a link to the chat I had with ChatGPT:
https://chat.openai.com/share/1434449a-e92c-41d6-b5e4-48c529f51c6e

1 Problem Definition

The Titanic was a ship that sailed between the years 1911 and 1912. Unfortunately the ship sank and many of people on board died. Data records about the passengers inside the ship allow us to train a machine learning model to test if any of the features of passengers predict whether they survived or not. For example, does being a women or a men impact your survival rate? Does being rich or poor impact your survival rate?

We can see this problem as a binary classification task that predicts "survived" or "not survived" for each passenger in the ship.

2 Dataset

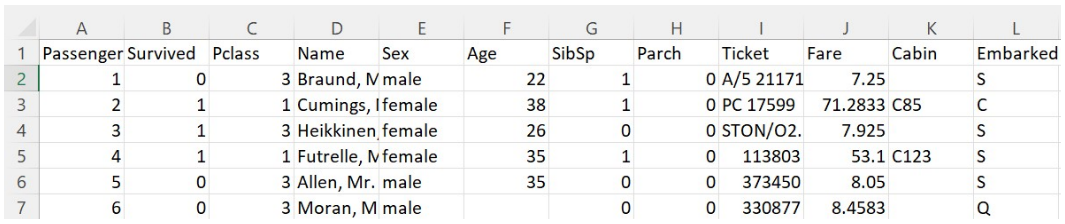
I have used the dataset about Titanic passengers from Kaggle, which had hosted a competition on predicting Titanic survival. This data is open to the public once you accept Kaggle's terms and conditions. The data comes in a CSV format that is already organized in terms of columns being features and rows being different passengers. This organization speeds up the training of models.

2.1 Data exploration

Kaggle provides a test and a train dataset. I noticed the test dataset has no labels informing if passengers survived or not, because this dataset is used in Kaggle competitions where you report your predictions without knowing the real answer. So I ignored the test CSV and only used the training CSV in this project.

The dataset has 12 features for each passenger, which include both categorical and numerical features: Passenger number, if they survived, passenger class, name, sex, age, number of siblings and spouses, number of parents and children, ticket number, fare paid, cabin number, and the location the passenger embarked.

We don't have information about most passenger's cabin number. Information about age is also incomplete, although we have this data for most passengers. You can open this data in Excel and see how it was organized in the CSV in the table below.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Passenger	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	0	3	Braund, M	male	22	1	0	A/5 21171	7.25		S
3	2	1	1	Cumings, I	female	38	1	0	PC 17599	71.2833	C85	C
4	3	1	3	Heikkinen,	female	26	0	0	STON/O2.	7.925		S
5	4	1	1	Futrelle, M	female	35	1	0	113803	53.1	C123	S
6	5	0	3	Allen, Mr.	male	35	0	0	373450	8.05		S
7	6	0	3	Moran, M	male		0	0	330877	8.4583		Q

3 Method

We will train a random forest, a support vector machine, and a neural network on the training dataset, and compare their predictive accuracy's. We will train the 3 different machine learning models following the same steps:

1. Preprocess data and engineer new features.
2. Define model and perform hyperparameter tuning.
3. Cross-validate manually and evaluate performance.

The algorithm for data preprocessing and feature engineering is like follows:

```
Load the dataset
Engineer features
Preprocess data, such as filling missing data and removing
    some labels from training data
Prepare the data for training by splitting data into X and y
```

The algorithm for defining the model is like follows:

```
(for neural net) Define a function to create the model
(for random forest and SVM) Create a preprocessing and
    modeling pipeline
(for neural net) Wrap the model so it can be used by
    scikit-learn
```

The algorithm for hyperparameter tuning is like follows:

```
Define hyperparameter search space
Use GridSearchCV
Print best paramters and best accuracy
```

The algorithm for manual cross validation is like follows:

```
Split data into k folds
Fit the model with the best parameters on the training data
Predict and evaluate on the test set
Print classification report for each fold
```

We will use the Google Colab environment for running the code due to:

1. Free GPU and RAM that will assist in training the machine learning models.
2. Ease of downloading libraries and quick setup of the programming environment

4 Experiments

4.1 Data Preprocessing and feature engineering

To make a fair comparison across models, we will use the same preprocessed data to train them. For the numerical features, we will fill in the missing values using the median of the values of that category of missing values. For the categorical features, we will fill in the missing values with the string "missing".

Furthermore we will drop a few of the features from the training data: "survived", "name", "ticket" and "cabin". We drop "survived" because that will be our target label instead of a feature. And we dropped name and ticket because everyone's names and ticket numbers are completely different strings so the model won't be able to learn anything from this label. We dropped the cabin features because of too many missing values.

We will also engineer three new features that are similar across all models:

Engineered Feature	**Description**
Family Size	Sum of Siblings/Spouses and Parents/Children
Is Alone	1 if no family members (Family Size = 1), 0 otherwise
Title	Extracted from the individual's name

Table 1: Description of Engineered Features

4.2 Hyperparameter tuning space

We will use GridsearchCV to cross validate the model on the data and find its best hyperparameters.

Random Forest

Parameter	**Values**
N Estimators	100, 200
Max Features	sqrt
Max Depth	5, 10
Min Samples Split	2, 5
Min Samples Leaf	1, 2

Neural Network

Parameter	**Values**
Batch Size	10, 20, 30
Epochs	10, 20
Optimizer	adam, rmsprop

SVM

Parameter	**Values**
C	0.1, 1, 10
Gamma	0.001, 0.01, 0.1, 1
Kernel	rbf, poly, sigmoid

4.3 Optimal Hyperparameters Found using GridSearchCV

Random Forest

Parameter	**Value**
Max Depth	10
Max Features	sqrt
Min Samples Leaf	2
Min Samples Split	2
Number of Estimators	100
Best Accuracy	0.8350

Neural Network

Parameter	**Value**
Batch Size	20
Epochs	20
Optimizer	rmsprop
Best Accuracy	0.8316

SVM

Parameter	**Value**
C	10
Gamma	0.01
Kernel	rbf
Best Accuracy	0.84

4.4 Manual cross-validation results

Even though GridsearchCV does cross validation, we will also cross validate each model manually to have a more nuanced and comprehensive understanding of how well each model performs on the whole dataset.

We split the data into 5 folds and test the performance of the model that has the tuned hyperparameters.

Random Forest

Fold	**Accuracy**
1	0.8436
2	0.8258
3	0.8146
4	0.8427
5	0.8539
Mean CV Accuracy	0.8361

Neural Network

Fold	Accuracy
1	0.8379
2	0.8370
3	0.8258
4	0.8427
5	0.8483
Mean CV Accuracy	0.8383

SVM

Fold	Accuracy
1	0.8436
2	0.8258
3	0.8315
4	0.8315
5	0.8483
Mean CV Accuracy	0.8361

5 Results Analysis

5.1 Comparison across models

Model	Runtime for GridSearchCV using 5 folds (seconds)	Mean CV Accuracy
Random Forest	49	0.8361
Neural Network	180 seconds	0.8294
SVM	40	0.8361

Table 2: Runtime and Accuracy Comparison of Models Performing GridSearchCV

When finding hyperparameters and cross validating using GridSearchCV, the neural network had the longest runtime. It took 3 minutes. The SVM model was the fastest, taking 40 seconds.

The mean cross validation accuracy of all three models was very similar, around 83 percent. I did not expect this similarity, as I hypothesized that the random forest would have the highest predictive accuracy.

5.2 Overfitting vs. Underfitting

Since the accuracies of all three models were consistently around 83 percent for all 5 folds in cross validation (all accuracies were within 5 percent of the best estimators for each model), the models have neither underfitted nor overfitted.

5.3 Limitations

I noticed that the runtime for hyperparameter tuning in the neural network and random forest lasts more than a few minutes if we pick more than a couple hyperparameters to tune. Therefore, I explored a limited set of hyperparameters per model. Better hyperparameters may exist for each of the models I tested that were not identified in this work.

Furthermore, I noticed the neural network found that rmsprop and batch size = 20 was better in one GridSearchCV with 3 folds, and when running GridSearchCV with 5 folds the best parameters were adam with batch size 20. This inconsistent result makes me unsure of what is the ideal optmizer to use with the Titanic dataset.

Import libraries

```
#libraries for all models
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix


#libraries for random forest
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier

#libraries for neural network
!pip install scikeras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from scikeras.wrappers import KerasClassifier

#libraries for SVM
!pip install scikit-learn
from sklearn.svm import SVC
```

```
     Requirement already satisfied: scikeras in /usr/local/lib/python3.10/dist-packages (0.12.0)
     Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (23
     Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras)
     Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1
     Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.
     Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1
     Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-l
     Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
     Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (
     Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1
     Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (
     Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-l
```

Load data and preprocess it

```
#
# Load the dataset
train_data = pd.read_csv('train.csv')

# Feature Engineering
train_data['FamilySize'] = train_data['SibSp'] + train_data['Parch'] + 1
train_data['IsAlone'] = np.where(train_data['FamilySize'] > 1, 0, 1)
train_data['Title'] = train_data['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)

# Preprocessing Steps
numeric_features = ['Age', 'Fare', 'FamilySize']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])
    ('scaler', StandardScaler())])

categorical_features = ['Embarked', 'Sex', 'Pclass', 'Title', 'IsAlone']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Prepare the data
X = train_data.drop(['Survived', 'Name', 'Ticket', 'Cabin'], axis=1)
y = train_data['Survived']

#X = preprocessor.fit_transform(X)
```

Random forest

Create model

```
# Create a preprocessing and modelling pipeline
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('classifier', RandomForestClassifier(random_state=42))])
```

Hyperparameter tuning

```
# Define hyperparameter search space
param_grid = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_features': ['sqrt'],  # Changed from ['auto', 'sqrt'] to just ['sqrt']
    'classifier__max_depth' : [5, 10],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2]
}
# Hyperparameter tuning using Grid Search with Cross-Validation

cv = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
cv.fit(X, y)

print("Best parameters found: ", cv.best_params_)
print("Best accuracy found: ", cv.best_score_)
```

```
Best parameters found:  {'classifier__max_depth': 10, 'classifier__max_features': 'sqrt', 'classifier__min_sa
Best accuracy found:  0.8350260498399347
```

Cross validate model

```
best_model = cv.best_estimator_

# Stratified K-Fold cross-validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold_no = 1
cv_scores = []
for train_idx, test_idx in kfold.split(X, y):
    # Fit the model with the best parameters on the training data
    best_model.fit(X.iloc[train_idx], y.iloc[train_idx])

    # Evaluate the model
    y_pred = best_model.predict(X.iloc[test_idx])
    accuracy = accuracy_score(y.iloc[test_idx], y_pred)
    cv_scores.append(accuracy)

    print(f'Fold {fold_no}')
    print(f'Accuracy: {accuracy}')
    # Print classification report for each fold
    print(f'Fold Classification Report:\n{classification_report(y.iloc[test_idx], y_pred > 0.5)}')
    print('Confusion Matrix:')
    print(confusion_matrix(y.iloc[test_idx], y_pred))
    print('-' * 30)

    fold_no += 1
print('Cross-validated accuracy scores:', cv_scores)
print('Mean CV Accuracy:', np.mean(cv_scores))
```

```
   macro avg       0.82      0.81      0.81       178
weighted avg       0.82      0.83      0.82       178

Confusion Matrix:
[[97 13]
 [18 50]]
------------------------------
Fold 3
Accuracy: 0.8146067415730337
Fold Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.93      0.86       110
```

```
Fold 5
Accuracy: 0.8539325842696629
Fold Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.91      0.88       109
           1       0.84      0.77      0.80        69

    accuracy                           0.85       178
   macro avg       0.85      0.84      0.84       178
weighted avg       0.85      0.85      0.85       178

Confusion Matrix:
[[99 10]
 [16 53]]
------------------------------
Cross-validated accuracy scores: [0.8435754189944135, 0.8258426966292135, 0.8146067415730337, 0.842696629
Mean CV Accuracy: 0.8361308141359614
```

Neural Network

Create a neural network model & tune hyperparameters

```
X_nn = preprocessor.fit_transform(X)

# Define a function to create the model (needed for KerasClassifier)
def neural_network(learning_rate=0.001, dropout_rate=0.2, optimizer = 'adam'):
    model = Sequential([
        Dense(64, activation='relu', input_dim=X_nn.shape[1]),
        Dropout(dropout_rate),
        Dense(32, activation='relu'),
        Dropout(dropout_rate),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Wrap the model so it can be used by scikit-learn
model = KerasClassifier(build_fn=neural_network, verbose=0)
# Define hyperparameter search space
param_grid = {'batch_size': [10, 20, 30], 'epochs': [10, 20], 'optimizer': ['adam', 'rmsprop']}

# Hyperparameter tuning using Grid Search with Cross-Validation
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=5)
grid_result = grid.fit(X_nn, y)

print("Best parameters found: ", grid.best_params_)
print("Best accuracy found: ", grid.best_score_)
```

```
/usr/local/lib/python3.10/dist-packages/scikeras/wrappers.py:915: UserWarning: ``build_fn`` will be renamed t
  X, y = self._initialize(X, y)
Best parameters found:  {'batch_size': 20, 'epochs': 20, 'optimizer': 'rmsprop'}
Best accuracy found:  0.8372795179210344
```

Cross validate

```
best_model = grid_result.best_estimator_

# Cross-validation
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold_no = 1
cv_scores = []
for train_idx, test_idx in kfold.split(X_nn, y):
    # Fit the model with the best parameters on the training data
    best_model.fit(X_nn[train_idx], y.iloc[train_idx])

    # Predict and evaluate on the test set
    y_pred = best_model.predict(X_nn[test_idx])
    accuracy = accuracy_score(y.iloc[test_idx], y_pred > 0.5)
    cv_scores.append(accuracy)
    print(f'Fold {fold_no}')
    print(f'Accuracy: {accuracy}')

    # Print classification report for each fold
    print(f'Fold Classification Report:\n{classification_report(y.iloc[test_idx], y_pred > 0.5)}')
    print('Confusion Matrix:')
    print(confusion_matrix(y.iloc[test_idx], y_pred))
    print('-' * 30)
    fold_no += 1

print('Cross-validated accuracy scores:', cv_scores)
print('Mean CV Accuracy:', np.mean(cv_scores))
```

```
Confusion Matrix:
[[103   7]
 [ 20  48]]
------------------------------
/usr/local/lib/python3.10/dist-packages/scikeras/wrappers.py:915: UserWarning: ``build_fn`` will be rename
  X, y = self._initialize(X, y)
Fold 5
Accuracy: 0.8539325842696629
Fold Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.94      0.89       109
           1       0.88      0.72      0.79        69

    accuracy                           0.85       178
   macro avg       0.86      0.83      0.84       178
weighted avg       0.86      0.85      0.85       178

Confusion Matrix:
[[102   7]
 [ 19  50]]
------------------------------
Cross-validated accuracy scores: [0.8268156424581006, 0.8370786516853933, 0.8258426966292135, 0.848314606
Mean CV Accuracy: 0.8383968363567889
```

Support Vector Machine

Create SVM model and tune hyperparameters

```
# Create a preprocessing and modelling pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', SVC())])

# Hyperparameter tuning using Grid Search with Cross-Validation
param_grid = {
    'classifier__C': [0.1, 1, 10],
    'classifier__gamma': [0.001, 0.01, 0.1, 1],
    'classifier__kernel': ['rbf', 'poly', 'sigmoid']
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

# Best parameters and best score
print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters found:  {'classifier__C': 10, 'classifier__gamma': 0.01, 'classifier__kernel': 'rbf'}
Best cross-validation score: 0.84
```

Cross validate SVM

```
# Stratified K-Fold cross-validation for detailed evaluation
best_model = grid_search.best_estimator_
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold_no = 1
cv_scores = []
for train, test in kfold.split(X, y):
    best_model.fit(X.iloc[train], y.iloc[train])
    y_pred = best_model.predict(X.iloc[test])

    print(f'Fold {fold_no}')
    accuracy = accuracy_score(y.iloc[test], y_pred > 0.5)
    cv_scores.append(accuracy)
    print(f'Accuracy: {accuracy}')

    # Print classification report for each fold
    print('Classification Report:')
    print(classification_report(y.iloc[test], y_pred))
    print('Confusion Matrix:')
    print(confusion_matrix(y.iloc[test], y_pred))


    print('-' * 30)

    fold_no += 1

print('Cross-validated accuracy scores:', cv_scores)
print('Mean CV Accuracy:', np.mean(cv_scores))
```

```
Fold 1
Accuracy: 0.8435754189944135
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       110
           1       0.82      0.77      0.79        69

    accuracy                           0.84       179
   macro avg       0.84      0.83      0.83       179
weighted avg       0.84      0.84      0.84       179

Confusion Matrix:
[[98 12]
 [16 53]]
------------------------------
Fold 2
Accuracy: 0.8258426966292135
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       110
           1       0.79      0.74      0.76        68

    accuracy                           0.83       178
   macro avg       0.82      0.81      0.81       178
weighted avg       0.82      0.83      0.82       178

Confusion Matrix:
[[97 13]
 [18 50]]
------------------------------
Fold 3
Accuracy: 0.8314606741573034
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.92      0.87       110
           1       0.84      0.69      0.76        68

    accuracy                           0.83       178
   macro avg       0.83      0.80      0.81       178
weighted avg       0.83      0.83      0.83       178

Confusion Matrix:
[[101   9]
 [ 21  47]]
------------------------------
Fold 4
Accuracy: 0.8314606741573034
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.89      0.87       110
           1       0.81      0.74      0.77        68

```