BITS PILANI WILP

APPLIED MACHINE LEARNING

SSZG568

ASSIGNMENT-1

TITANIC SURVIVAL PREDICTION USING SKLEARN

NAME: PRAJWAL S TELKAR

BITS ID: 2023MT12205

MTech-Software Systems

April 7, 2024

Introduction

The Titanic dataset contains information about passengers aboard the Titanic, including whether they survived or not. In this report, we aim to use machine learning techniques to predict passenger survival based on the available data. Our goal is to develop a predictive model that accurately determines the likelihood of survival for each passenger. To achieve this, we will explore the dataset, preprocess the data, and select relevant features. We will then train and evaluate machine learning models, aiming to achieve a high level of accuracy while avoiding overfitting.

Through this analysis, we seek to gain insights into the factors that influenced survival on the Titanic and demonstrate the effectiveness of machine learning in predicting outcomes based on historical data.

End To End ML Project Steps

1. Look at the Big Picture:
    - Define the objective of the project (e.g., predicting passenger survival on the Titanic).
    - Determine how the machine learning model will be used in the broader context (e.g., decision support system for maritime safety).
2. Get the Data:
    - Obtain the Titanic dataset, which contains information about passengers, including whether they survived or not.
3. Discover and Visualize the Data:
    - Explore the dataset to understand its structure and contents.
    - Use data visualization techniques to gain insights into the data (e.g., survival rates by gender or ticket class).
4. Prepare the Data for Machine Learning Algorithms:
    - Clean the data by handling missing values and encoding categorical variables.
    - Feature engineering: Create new features or transform existing ones to improve model performance.
5. Select a Model and Train It:
    - Choose a machine learning model suitable for the task (e.g., Decision Tree Classifier).
    - Split the data into training and testing sets.
    - Train the model using the training data.
6. Fine-Tune Your Model:

- Use cross-validation to fine-tune hyperparameters and avoid overfitting.
- Evaluate the model on the test set to ensure generalization to new data.

7. Present Your Solution:
   - Summarize the findings from the analysis.
   - Present the model's performance metrics and any insights gained.
8. Launch, Monitor, and Maintain Your System:
   - Deploy the model into a production environment if applicable.
   - Monitor the model's performance over time and update it as needed to maintain its effectiveness.

Get The Data

Step1: Load the dataset

The first step is to load the dataset. We will be using the Titanic dataset, which contains information about passengers on the Titanic ship, including whether or not they survived.

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df_train = pd.read_csv(r"C:\Users\Prajwal\Desktop\Mtech\Second Sem\ML\train.csv")
df_test = pd.read_csv(r"C:\Users\Prajwal\Desktop\Mtech\Second Sem\ML\test.csv")
```

```
df_train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

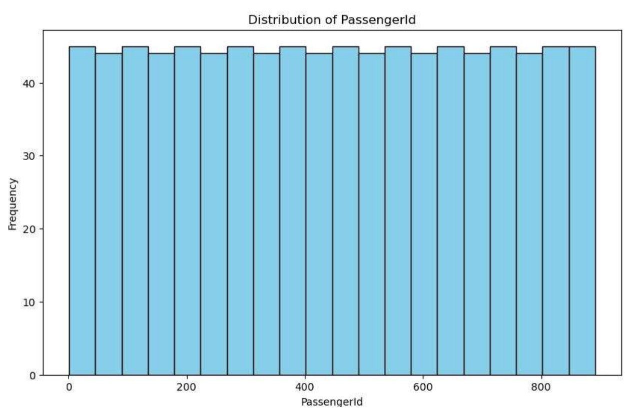
```
df_train.describe()
```

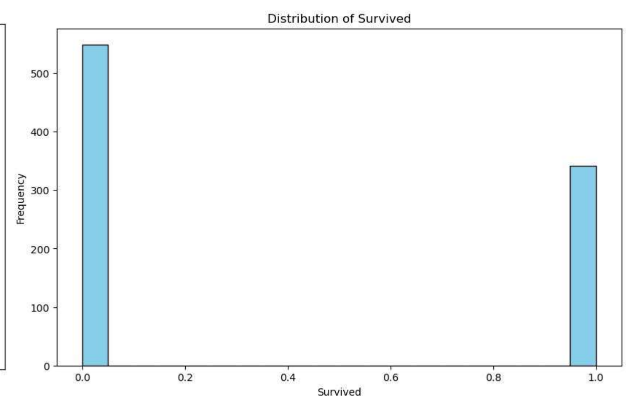
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

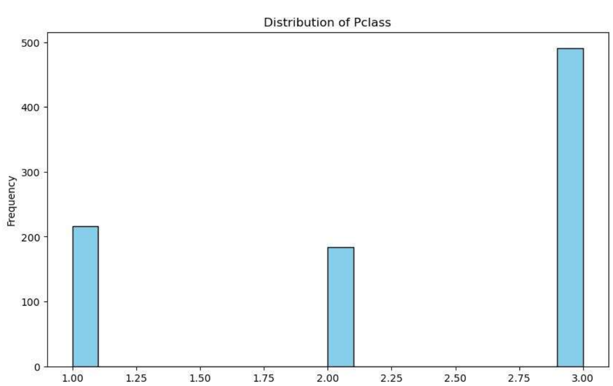
Data Visualization

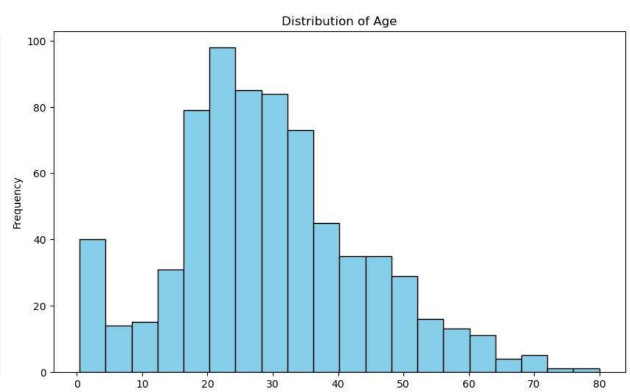
Data visualization helps us explore and communicate the patterns and trends in the dataset, which is essential for making informed decisions and building predictive models.

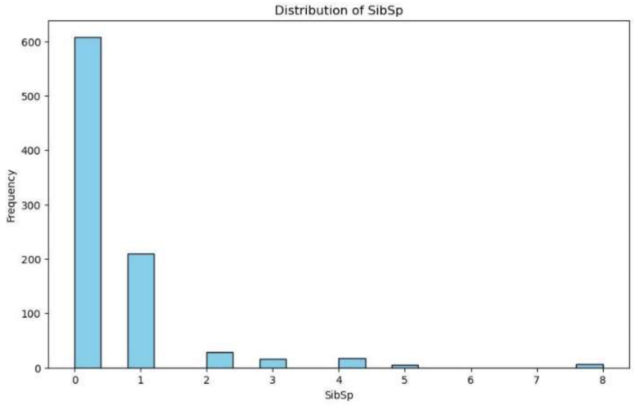
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Load the Titanic dataset
train_data = pd.read_csv(r"C:\Users\Prajwal\Desktop\Mtech\Second Sem\ML\train.csv")

# Plot histograms for numerical columns
numerical_columns = train_data.select_dtypes(include=['int64', 'float64']).columns
for col in numerical_columns:
    plt.figure(figsize=(10, 6))
    plt.hist(train_data[col].dropna(), bins=20, color='skyblue', edgecolor='black')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {col}')
    plt.show()

# Plot bar plots for categorical columns
categorical_columns = train_data.select_dtypes(include=['object']).columns
for col in categorical_columns:
    plt.figure(figsize=(8, 5))
    sns.countplot(data=train_data, x=col, palette='Set2')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.title(f'Distribution of {col}')
    plt.xticks(rotation=45)
    plt.show()
```

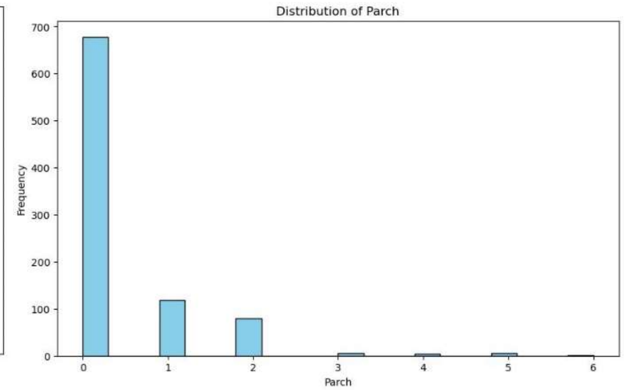


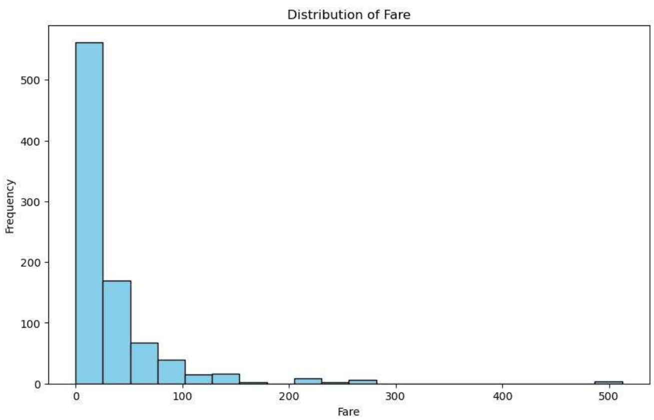


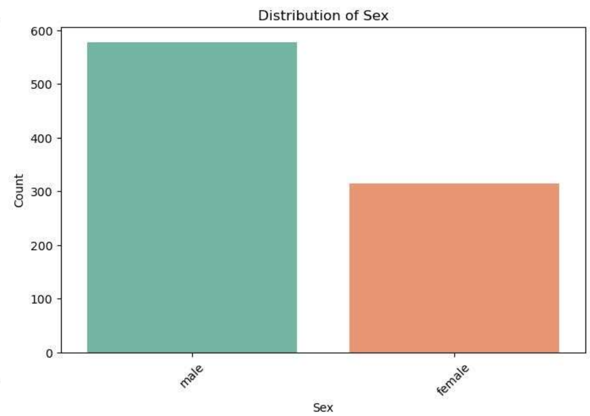


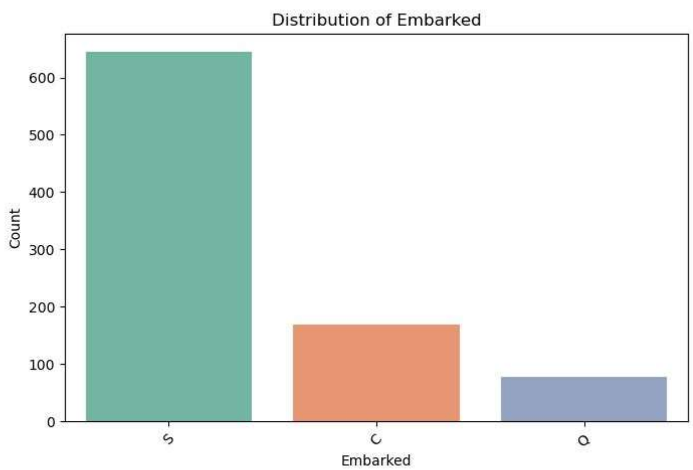












Data Preparation

In this step, we loaded the dataset and performed basic preprocessing steps to prepare the data for modeling. This included handling missing values and encoding categorical variables to convert them into a format suitable for machine learning algorithms.

Step 1: Preprocess the dataset

Before we can use the dataset for training our decision tree model, we need to preprocess it. First, we will handle missing values in the dataset. In this case, we will fill missing values in the 'Age' column with the median age, and missing values in the 'Embarked' column with the mode (most common value). Filling missing values with these central tendency measures helps to ensure that the datasets are complete and ready for further analysis and modeling.

```
# Preprocessing
# Fill missing values
df_train['Age'].fillna(df_train['Age'].median(), inplace=True)
df_test['Age'].fillna(df_test['Age'].median(), inplace=True)
df_train['Embarked'].fillna(df_train['Embarked'].mode()[0], inplace=True)
df_test['Embarked'].fillna(df_test['Embarked'].mode()[0], inplace=True)
```

Step 2: Feature Selection/Engineering

Feature engineering involves creating new features from existing ones to improve model performance. We didn't explicitly mention any new features, but this step could include creating interaction terms or transforming existing features to better represent the data.

Feature selection is the process of selecting the most relevant features for the model. In this case, we selected features such as 'Pclass' (passenger class), 'Sex', 'Age', 'SibSp' (number of siblings/spouses aboard), 'Parch' (number of parents/children aboard), 'Fare' (ticket fare), and 'Embarked' (port of embarkation).

```
# Feature selection
features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
X = pd.get_dummies(df_train[features])
print("Input matrix for the training data, X:\n", X)
```

```
Input matrix for the training data, X:
     Pclass   Age  SibSp  Parch     Fare  Sex_female  Sex_male  Embarked_C  \
0         3  22.0      1      0   7.2500       False      True       False
1         1  38.0      1      0  71.2833        True     False        True
2         3  26.0      0      0   7.9250        True     False       False
3         1  35.0      1      0  53.1000        True     False       False
4         3  35.0      0      0   8.0500       False      True       False
..      ...   ...    ...    ...      ...         ...       ...         ...
886       2  27.0      0      0  13.0000       False      True       False
887       1  19.0      0      0  30.0000        True     False       False
888       3  28.0      1      2  23.4500        True     False       False
889       1  26.0      0      0  30.0000       False      True        True
890       3  32.0      0      0   7.7500       False      True       False

     Embarked_Q  Embarked_S
0         False        True
1         False       False
2         False        True
3         False        True
4         False        True
..          ...         ...
886       False        True
887       False        True
888       False        True
889       False       False
890        True       False

[891 rows x 10 columns]
```

```
y = df_train['Survived']
print("Target Variable, y:\n", y)
```

```
Target Variable, y:
 0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

```
X_test = pd.get_dummies(df_test[features])
print("Input matrix for the test data, X_test:\n", X_test)
```

```
Input matrix for the test data, X_test:
      Pclass   Age  SibSp  Parch      Fare  Sex_female  Sex_male  Embarked_C  \
0         3  34.5      0      0    7.8292       False      True       False
1         3  47.0      1      0    7.0000        True     False       False
2         2  62.0      0      0    9.6875       False      True       False
3         3  27.0      0      0    8.6625       False      True       False
4         3  22.0      1      1   12.2875        True     False       False
..      ...   ...    ...    ...       ...         ...       ...         ...
413       3  27.0      0      0    8.0500       False      True       False
414       1  39.0      0      0  108.9000        True     False        True
415       3  38.5      0      0    7.2500       False      True       False
416       3  27.0      0      0    8.0500       False      True       False
417       3  27.0      1      1   22.3583       False      True        True

     Embarked_Q  Embarked_S
0          True       False
1         False        True
2          True       False
3         False        True
4         False        True
..          ...         ...
413       False        True
414       False       False
415       False        True
416       False        True
417       False       False

[418 rows x 10 columns]
```

Split The Data into Training and Test Sets

In the code, we import the train_test_split function from the sklearn.model_selection module, which is used to split the dataset into training and testing sets. Here, we pass the X and y dataframes to the train_test_split function, which returns four dataframes: X_train, X_test, y_train, and y_test.

The test_size parameter is set to 0.2, which means that 20% of the data is reserved for testing, and the remaining 80% is used for training. The random_state parameter is set to 42 to ensure that the split is reproducible.

```
# Split the data into training and testing sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
print("X_train:\n", X_train)
print("X_val:\n", X_val)
print("y_train:\n", y_train)
print("y_val:\n", y_val)
```

Model Selection

After we split the data into training and testing sets to train the models and evaluate their performance. The following models were trained on the training data and evaluated using cross-validation to estimate their accuracy.

1. Decision Tree Classifier
2. Random Forest

1. Decision Tree Classification

Decision Tree is a tree-like model where each internal node represents a feature or attribute, each branch represents a decision based on that feature, and each leaf node represents the outcome or target variable. Decision Tree Classifier is used to predict whether a passenger survived or not based on various features such as their age, gender, ticket class, fare, and embarkation port. The Decision Tree algorithm iteratively splits the dataset into subsets based on feature values to create a tree-like structure.

Select the hyperparameters and tune the model

Tuning the decision tree model is required to find the best hyperparameters that optimize the model's performance. The max_depth parameter controls the maximum depth of the tree, which can help prevent overfitting. The min_samples_split parameter specifies the minimum number of samples required to split an internal node, while min_samples_leaf specifies the minimum number of samples required to be at a leaf node. The best hyper parameters were 'max_depth': 3, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 2

```python
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters to tune
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [2, 5, 7, 10],
    'min_samples_leaf': [1, 2, 4, 5, 7],
    'max_features': ['sqrt', 'log2', None]
}

# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                           param_grid=param_grid,
                           cv=5,
                           n_jobs=-1,
                           verbose=2)

# Perform the grid search
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Use the best model for prediction
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
Best Hyperparameters: {'max_depth': 3, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
```

Model Evaluation and Cross Validation

Cross-validation is a technique for evaluating machine learning models like the Decision Tree Classifier. It involves splitting the dataset into multiple subsets or folds, training the model on a subset, and testing it on a different subset. This process helps ensure that the model is robust and avoids overfitting. In our analysis, we conducted cross-validation with different fold numbers, including 5, 9, and 11. The best mean cross-validation accuracy was achieved with 11 folds, suggesting that this configuration provided the most reliable estimate of the model's performance. This outcome is likely due to the increased diversity in training and testing data combinations.

1. **First Pass:**
   Set Cross Validation to 5

```python
# Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(max_depth=3, min_samples_split=2, min_samples_leaf=5, random_state=42)
dt_classifier.fit(X_train, y_train)

# Model evaluation
# Cross-validation
cv_scores = cross_val_score(dt_classifier, X, y, cv=5)
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean CV accuracy: {cv_scores.mean()}")

# Predictions on the test set
y_pred = dt_classifier.predict(X_test)

# Output predictions to a file
output = pd.DataFrame({'PassengerId': df_test.PassengerId, 'Survived': y_pred})
output.to_csv(r"C:\Users\Prajwal\Desktop\Mtech\Second Sem\ML\titanic_predictions.csv", index=False)
```

```
Cross-validation scores: [0.82122905 0.81460674 0.81460674 0.78651685 0.81460674]
Mean CV accuracy: 0.8103132257862031
```

2. **Second Pass:**
   Set Cross Validation to 9

```python
# Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(max_depth=3, min_samples_split=2, min_samples_leaf=5, random_state=42)
dt_classifier.fit(X_train, y_train)

# Model evaluation
# Cross-validation
cv_scores = cross_val_score(dt_classifier, X, y, cv=9)
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean CV accuracy: {cv_scores.mean()}")

# Predictions on the test set
y_pred = dt_classifier.predict(X_test)

# Output predictions to a file
output = pd.DataFrame({'PassengerId': df_test.PassengerId, 'Survived': y_pred})
output.to_csv(r"C:\Users\Prajwal\Desktop\Mtech\Second Sem\ML\titanic_predictions.csv", index=False)
```

```
Cross-validation scores: [0.82828283 0.82828283 0.77777778 0.8989899  0.81818182 0.77777778
 0.78787879 0.83838384 0.7979798 ]
Mean CV accuracy: 0.8170594837261503
```

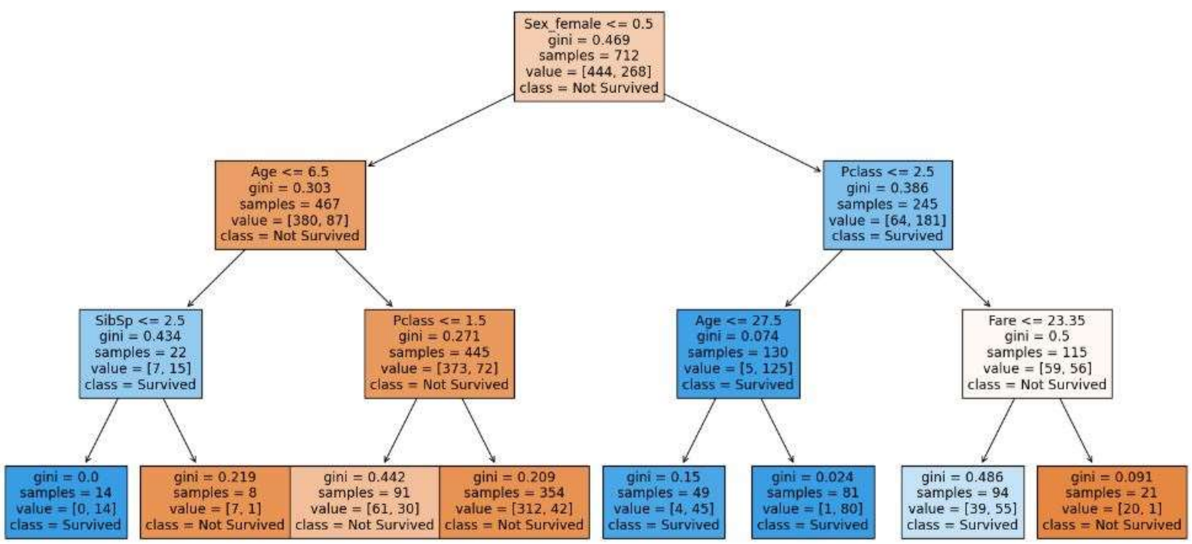
3. **Third Pass:**
   Set Cross Validation to 11

```
# Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(max_depth=3, min_samples_split=2, min_samples_leaf=5, random_state=42)
dt_classifier.fit(X_train, y_train)

# Model evaluation
# Cross-validation
cv_scores = cross_val_score(dt_classifier, X, y, cv=11)
print(f"Cross-validation scores: {cv_scores}")
print(f"Mean CV accuracy: {cv_scores.mean()}")

# Predictions on the test set
y_pred = dt_classifier.predict(X_test)

# Output predictions to a file
output = pd.DataFrame({'PassengerId': df_test.PassengerId, 'Survived': y_pred})
output.to_csv(r"C:\Users\Prajwal\Desktop\Mtech\Second Sem\ML\titanic_predictions_dt_classification.csv", index=False)
```

```
Cross-validation scores: [0.82716049 0.81481481 0.82716049 0.80246914 0.86419753 0.83950617
 0.77777778 0.80246914 0.7654321  0.88888889 0.81481481]
Mean CV accuracy: 0.8204264870931538
```

Visualize The Decision Tree Using Plot_Tree

```
from sklearn import tree
import matplotlib.pyplot as plt

# Get feature names as a list
feature_names = X.columns.tolist()

# Visualize the Decision Tree
plt.figure(figsize=(20,10))
tree.plot_tree(dt_classifier, filled=True, feature_names=feature_names, class_names=['Not Survived', 'Survived'])
plt.show()
```



2.Random Forest Classifier

Random Forest Classifier is an ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes (classification) or the average prediction (regression) of the individual trees. It can handle a larger number of features and is less prone to overfitting compared to a single decision tree. The model can be trained using the same features as the Decision Tree Classifier and then evaluated based on accuracy and other relevant metrics.

Select the hyperparameters and tune the model

Similar to Decision Tree, we tune the hyper parameters and select the best one, max_depth, n_estimators, min_samples_split and min_samples_leaf hyper parameters are evaluated against the model and best hyper parameters are selected to tune the model. Here the best fit hyper parameters were ‘max_depth': 10, ‘min_samples_leaf': 7, ‘min_samples_split': 2, ‘n_estimators': 100

```
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Define the pipeline with an imputer and the random forest classifier
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),  # Use mean to impute missing values
    ('classifier', RandomForestClassifier(random_state=42))
])

# Define the hyperparameters to tune
param_grid = {
    'classifier__n_estimators': [50, 100],
    'classifier__max_depth': [10, 20],
    'classifier__min_samples_split': [2, 5, 7, 10],
    'classifier__min_samples_leaf': [1, 2, 4, 5, 7]
}

# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=pipeline,
                           param_grid=param_grid,
                           n_jobs=-1,
                           verbose=2)

# Perform the grid search
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Use the best model for prediction
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# Output predictions to a file
output = pd.DataFrame({'PassengerId': df_test.PassengerId, 'Survived': y_pred})
output.to_csv(r"C:\Users\Prajwal\Desktop\Mtech\Second Sem\ML\titanic_predictions_rf_classification.csv", index=False)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
Best Hyperparameters: {'classifier__max_depth': 10, 'classifier__min_samples_leaf': 7, 'classifier__min_samples_split': 2, 'cla
ssifier__n_estimators': 100}
```

Model Evaluation and Cross Validation

1. **First Pass:**
   Set Cross Validation to 5

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=2, min_samples_leaf=7, random_state=42)
rf_classifier.fit(X_train, y_train)

# Model evaluation

# Random Forest
rf_cv_scores = cross_val_score(rf_classifier, X, y, cv=5)
print(f"Cross-validation scores: {rf_cv_scores}")
print(f"Random Forest - Mean CV accuracy: {rf_cv_scores.mean()}")
```

```
Cross-validation scores: [0.81564246 0.82022472 0.83707865 0.79213483 0.83707865]
Random Forest - Mean CV accuracy: 0.8204318624066287
```

2. **Second Pass:**
   Set Cross Validation to 9

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=2, min_samples_leaf=7, random_state=42)
rf_classifier.fit(X_train, y_train)

# Model evaluation

# Random Forest
rf_cv_scores = cross_val_score(rf_classifier, X, y, cv=9)
print(f"Cross-validation scores: {rf_cv_scores}")
print(f"Random Forest - Mean CV accuracy: {rf_cv_scores.mean()}")
```

```
Cross-validation scores: [0.7979798  0.78787879 0.78787879 0.8989899  0.83838384 0.78787879
 0.77777778 0.84848485 0.82828283]
Random Forest - Mean CV accuracy: 0.8170594837261503
```

3. **Third Pass:**
   Set Cross Validation to 11

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

# Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=2, min_samples_leaf=7, random_state=42)
rf_classifier.fit(X_train, y_train)

# Model evaluation

# Random Forest
rf_cv_scores = cross_val_score(rf_classifier, X, y, cv=11)
print(f"Cross-validation scores: {rf_cv_scores}")
print(f"Random Forest - Mean CV accuracy: {rf_cv_scores.mean()}")
```

```
Cross-validation scores: [0.80246914 0.81481481 0.80246914 0.82716049 0.90123457 0.85185185
 0.81481481 0.81481481 0.7654321  0.86419753 0.83950617]
Random Forest - Mean CV accuracy: 0.8271604938271605
```

Results

Comparing Decision Tree Classifier and Random Forest Classifier

```
# # Decision Tree
print(f"Decision Tree - Mean CV accuracy: {cv_scores.mean()}")
# Random Forest
print(f"Random Forest - Mean CV accuracy: {rf_cv_scores.mean()}")

# Compare the results
print(f"Decision Tree vs Random Forest: {np.mean(rf_cv_scores) - np.mean(cv_scores)}")
```

```
Decision Tree - Mean CV accuracy: 0.8204264870931538
Random Forest - Mean CV accuracy: 0.8271604938271605
Decision Tree vs Random Forest: 0.006734006734006703
```

The Decision Tree Classifier achieved a mean cross-validation accuracy of 0.82042(~82.04%), while the Random Forest Classifier achieved a slightly higher mean accuracy of 0.82716(~82.71%). The difference in mean accuracy between the two models was 0.00673(~0.67%), indicating that the Random Forest model performed slightly better.

Project Folder uploaded in Google Drive for reference

- https://drive.google.com/drive/folders/1V4SEO4llhNgAyHC5nc-ea6C_sdYMyd88?usp=drive_link

Predicted Values:

1. **Decision Tree Model (CSV file)**
    - titanic_predicted_values_dt_classification.csv
2. **Random Classifier Model (CSV file)**
    - titanic_predictions_rf_classification.csv

Code:

- 2023mt12205_ml_assignment1.ipynb

Jupyter NoteBook PDF :

- 2023mt12205-Assignment1-ML-JupyterCode.pdf

Source:

- https://www.kaggle.com/competitions/titanic/overview
- https://www.kaggle.com/competitions/titanic/data

References:

- https://www.kaggle.com/competitions/titanic/overview
- https://www.kaggle.com/competitions/titanic/data
- Implementing Decision Tree Algorithm for Classification with Titanic Dataset in Python | by Dr. Soumen Atta, Ph.D. | Medium
- https://www.geeksforgeeks.org/titanic-survival-prediction-using-tensorflow-in-python/
- Aurelien Geron, “Hands-On Machine Learning with Scikit-Learn, Keras and Tensorflow”, O’Reilly, 2020

- Supervised Machine Learning: Regression and Classification - Coursera