Lab 9: Ensemble Classifications

R Abhijit Srivathsan

2448044

Titanic Survival Prediction Using Multiple Classifiers

This notebook demonstrates how to build and evaluate four different classifiers on the Titanic dataset to predict passenger survival:

1. **SVM**
2. **Multilayer Perceptron (MLP)**
3. **Random Forest**
4. **Gradient Boosting**

We will compare their performances and record observations.

In [15]:

```
# -----------------------------
# 1. Importing Libraries
# -----------------------------
import numpy as np
import pandas as pd

# For splitting dataset & performance metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Classifiers
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# For data preprocessing
from sklearn.preprocessing import LabelEncoder
```

```python
# Ignore warnings for cleaner output (optional)
import warnings
warnings.filterwarnings('ignore')
```

2. Loading the Dataset

In [16]:
```python
# ----------------------------
# 2. Loading the dataset
# ----------------------------
df = pd.read_csv('train.csv')
df.head()
```

Out[16]:

	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

A quick look at the data:

- **PassengerId**: Unique ID for each passenger
- **Survived**: 0 = No, 1 = Yes
- **Pclass**: Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
- **Name**: Passenger name
- **Sex**: Passenger gender
- **Age**: Passenger age in years
- **SibSp**: # of siblings/spouses aboard
- **Parch**: # of parents/children aboard

- **Ticket**: Ticket number
- **Fare**: Passenger fare
- **Cabin**: Cabin number
- **Embarked**: Port of Embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

3. Preprocessing

We will:

1. Handle missing values (particularly in the `Age` and `Embarked` columns).
2. Drop unnecessary columns (`PassengerId` , `Name` , `Ticket` , `Cabin`).
3. Convert categorical features (`Sex` , `Embarked`) to numeric.
4. Split the dataset into features and labels. "

In [17]:
```
# ----------------------------
# 3. Data Preprocessing
# ----------------------------

# Check missing values
df.isnull().sum()
```

Out[17]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [18]:

```python
# Fill missing 'Age' with median
df['Age'].fillna(df['Age'].median(), inplace=True)

# Fill missing 'Embarked' with the most frequent value
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# Drop columns that are not essential or have too many NaNs
df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# Convert categorical columns into numeric
le_sex = LabelEncoder()
df['Sex'] = le_sex.fit_transform(df['Sex'])  # 0 or 1

le_embarked = LabelEncoder()
df['Embarked'] = le_embarked.fit_transform(df['Embarked'])  # 0,1,2 for S,C,Q (depending on order)

df.head()
```

Out[18]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	7.2500	2
1	1	1	0	38.0	1	0	71.2833	0
2	1	3	0	26.0	0	0	7.9250	2
3	1	1	0	35.0	1	0	53.1000	2
4	0	3	1	35.0	0	0	8.0500	2

Final Feature Set

Now our data should look something like:

- **Survived** (label)
- **Pclass**
- **Sex** (0 or 1)
- **Age**

- **SibSp**
- **Parch**
- **Fare**
- **Embarked** (encoded as 0,1,2)

We'll separate the target variable `Survived` from the features.

In [19]:
```
# Separate features and target
X = df.drop('Survived', axis=1)
y = df['Survived']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

X_train.shape, X_test.shape
```

Out[19]:
```
((712, 7), (179, 7))
```

4. Training Models and Evaluating Performance

We will train each of the following classifiers on the training data:

1. **SVM (Support Vector Classifier)**
2. **Multilayer Perceptron**
3. **Random Forest**
4. **Gradient Boosting**

After training, we'll predict on the test set and compare accuracies.

In [20]:
```
# ----------------------------
# 4.1 Support Vector Machine
# ----------------------------
```

```
svm_model = SVC()
svm_model.fit(X_train, y_train)

y_pred_svm = svm_model.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)

print("SVM Accuracy: {:.4f}".format(accuracy_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
```

```
SVM Accuracy: 0.6592
Classification Report:
               precision    recall  f1-score   support

           0       0.64      0.94      0.76       105
           1       0.76      0.26      0.38        74

    accuracy                           0.66       179
   macro avg       0.70      0.60      0.57       179
weighted avg       0.69      0.66      0.61       179

Confusion Matrix:
 [[99  6]
 [55 19]]
```

In [21]:

```
# ----------------------------
# 4.2 Multilayer Perceptron
# ----------------------------
mlp_model = MLPClassifier(max_iter=500, random_state=42)
mlp_model.fit(X_train, y_train)

y_pred_mlp = mlp_model.predict(X_test)
accuracy_mlp = accuracy_score(y_test, y_pred_mlp)

print("MLP Accuracy: {:.4f}".format(accuracy_mlp))
print("Classification Report:\n", classification_report(y_test, y_pred_mlp))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_mlp))
```

```
MLP Accuracy: 0.7654
Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.80      0.80       105
           1       0.72      0.72      0.72        74

    accuracy                           0.77       179
   macro avg       0.76      0.76      0.76       179
weighted avg       0.77      0.77      0.77       179

Confusion Matrix:
 [[84 21]
 [21 53]]
```

In [22]:

```
# ----------------------------
# 4.3 Random Forest
# ----------------------------
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

print("Random Forest Accuracy: {:.4f}".format(accuracy_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.8212
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.88      0.85       105
           1       0.81      0.74      0.77        74

    accuracy                           0.82       179
   macro avg       0.82      0.81      0.81       179
weighted avg       0.82      0.82      0.82       179

Confusion Matrix:
 [[92 13]
 [19 55]]
```

In [23]:

```python
# -----------------------------
# 4.4 Gradient Boosting
# -----------------------------
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_model.fit(X_train, y_train)

y_pred_gb = gb_model.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)

print("Gradient Boosting Accuracy: {:.4f}".format(accuracy_gb))
print("Classification Report:\n", classification_report(y_test, y_pred_gb))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_gb))
```

```
Gradient Boosting Accuracy: 0.8101
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.89      0.85       105
           1       0.81      0.70      0.75        74

    accuracy                           0.81       179
   macro avg       0.81      0.79      0.80       179
weighted avg       0.81      0.81      0.81       179

Confusion Matrix:
 [[93 12]
 [22 52]]
```

5. Comparing Results

In [24]:

```
# Compare the four accuracy scores
model_results = {
    'SVM': accuracy_svm,
    'MLP': accuracy_mlp,
    'Random Forest': accuracy_rf,
    'Gradient Boosting': accuracy_gb
}

results_df = pd.DataFrame.from_dict(model_results, orient='index', columns=['Accuracy'])
results_df.sort_values(by='Accuracy', ascending=False, inplace=True)
results_df
```

Out[24]:

	Accuracy
Random Forest	0.821229
Gradient Boosting	0.810056
MLP	0.765363
SVM	0.659218

Final Observations and Conclusions

- **Random Forest** achieved the highest accuracy (**0.821229**), indicating it most effectively captured the relationships in the data among all tested models.
- **Gradient Boosting** followed closely (**0.810056**), reinforcing the typical strength of ensemble methods on tabular datasets like Titanic.
- **Multilayer Perceptron (MLP)** (**0.765363**) outperformed the baseline SVM but did not surpass the ensemble methods, suggesting that further hyperparameter tuning (e.g., adjusting hidden layers, learning rate, or regularization) could improve its performance.
- **SVM** recorded the lowest accuracy (**0.659218**), indicating that in its default configuration, it may not be well-suited for this dataset. SVMs often require careful tuning of parameters (such as (C), kernel choice, and gamma) to reach competitive results.

Overall, **ensemble methods** (Random Forest and Gradient Boosting) proved most effective in this case, which is a common outcome when dealing with structured data like the Titanic dataset.