xgboost분류.py

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
import xgboost as xgb

# Step 1: Load the Titanic dataset
# Assuming the data is downloaded from Kaggle and placed in the working directory
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

# Step 2: Data Preprocessing
# Handle missing values
train_df['Age'].fillna(train_df['Age'].median(), inplace=True)
test_df['Age'].fillna(test_df['Age'].median(), inplace=True)
train_df['Embarked'].fillna(train_df['Embarked'].mode()[0], inplace=True)
test_df['Fare'].fillna(test_df['Fare'].median(), inplace=True)

# Drop unnecessary columns
train_df.drop(['Cabin', 'Ticket', 'Name', 'PassengerId'], axis=1, inplace=True)
test_passenger_ids = test_df['PassengerId']  # Save for submission
test_df.drop(['Cabin', 'Ticket', 'Name', 'PassengerId'], axis=1, inplace=True)

# Encode categorical variables
label_encoder = LabelEncoder()
train_df['Sex'] = label_encoder.fit_transform(train_df['Sex'])
test_df['Sex'] = label_encoder.transform(test_df['Sex'])

train_df['Embarked'] = label_encoder.fit_transform(train_df['Embarked'])
test_df['Embarked'] = label_encoder.transform(test_df['Embarked'])

# Split train data into features and target
X = train_df.drop('Survived', axis=1)
y = train_df['Survived']

# Split into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Define XGBoost model
model = xgb.XGBClassifier(objective='binary:logistic', random_state=42)

# Step 4: Parameter optimization using GridSearchCV (which includes cross-validation)
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0]
}

```

```python
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)
grid_search.fit(X_train, y_train)

# Best parameters and model
best_params = grid_search.best_params_
print("Best parameters found: ", best_params)
best_model = grid_search.best_estimator_

# Step 5: Evaluate model performance on validation set
y_pred_val = best_model.predict(X_val)

# Performance metrics
accuracy = accuracy_score(y_val, y_pred_val)
precision = precision_score(y_val, y_pred_val)
recall = recall_score(y_val, y_pred_val)
f1 = f1_score(y_val, y_pred_val)
roc_auc = roc_auc_score(y_val, y_pred_val)

print("\nModel Performance on Validation Set:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC AUC: {roc_auc:.4f}")

# Cross-validation scores on full train data
cv_scores = cross_val_score(best_model, X, y, cv=5, scoring='accuracy')
print("\nCross-Validation Accuracy Scores: ", cv_scores)
print("Mean CV Accuracy: ", cv_scores.mean())

# Step 6: Train the best model on full train data
best_model.fit(X, y)

# Step 7: Predict on test data
test_predictions = best_model.predict(test_df)

# Create submission file
submission = pd.DataFrame({
    'PassengerId': test_passenger_ids,
    'Survived': test_predictions
})
submission.to_csv('submission.csv', index=False)
print("\nPredictions saved to submission.csv")
```