```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer
from sklearn.model_selection import learning_curve
```

```
# Load the dataset
data = pd.read_csv("credit_rating.csv")
```

```
# Drop the S.No. columns
data = data.drop(data.columns[data.columns.str.contains('S.No')], axis=1)
```

```
# Handle missing values if any
data.dropna(inplace=True)
```

```
data.head()
```

	CHK_ACCT	Duration	History	Purpose of credit	Credit Amount	Balance in Savings A/C	Employment	Install_rate
0	0DM	6	critical	radio-tv	1169	unknown	over-seven	4
1	less-200DM	48	duly-till-now	radio-tv	5951	less100DM	four-years	2
2	no-account	12	critical	education	2096	less100DM	seven-years	2
3	0DM	42	duly-till-now	furniture	7882	less100DM	seven-years	2
4	0DM	24	delay	new-car	4870	less100DM	four-years	3

5 rows × 21 columns

```
label_encoder = LabelEncoder()
for col in data.columns[data.dtypes == 'object']:
    data[col] = label_encoder.fit_transform(data[col])
```

```
data.head()
```

	CHK_ACCT	Duration	History	Purpose of credit	Credit Amount	Balance in Savings A/C	Employment	Install_rate
0	0	6	2	6	1169	4	2	4
1	1	48	4	6	5951	1	0	2
2	2	12	2	2	2096	1	3	2
3	0	42	4	3	7882	1	3	2
4	0	24	3	4	4870	1	0	3

5 rows × 21 columns

```
# Save feature names
feature_names = data.columns.tolist()
print(feature_names)
```

```
['CHK_ACCT', 'Duration', 'History', 'Purpose of credit', 'Credit Amount', 'Balance in Savings A/C', 'Employment', 'Install_rate',
```

```
# Splitting the data into features (X) and target variable (y)
X = data.drop('Credit classification', axis=1)
y = data['Credit classification']
```

```
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Apply SMOTE for class imbalance
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

```
# Train Logistic Regression with cross-validation
lr_classifier = LogisticRegression(penalty='l2', solver='liblinear')
cv_lr_scores = cross_val_score(lr_classifier, X_train, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Scores for Logistic Regression:", cv_lr_scores)
print("Mean Accuracy (Logistic Regression):", np.mean(cv_lr_scores))
```

```
Cross-Validation Scores for Logistic Regression: [0.75       0.75       0.70089286 0.75446429 0.76785714]
Mean Accuracy (Logistic Regression): 0.7446428571428572
```

```
# Get LR Predictions
lr_classifier.fit(X_train, y_train)
lr_pred_prob = lr_classifier.predict_proba(X_test)
```

```
# Feature Engineering for Stacking
stacking_features = lr_pred_prob
```

```
# Train DecisionTreeClassifier on stacking features
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(stacking_features, y_test)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
# Generate predictions from DecisionTreeClassifier
dt_pred_prob = dt_classifier.predict_proba(stacking_features)
```

```
# Augment stacking features with predictions from DecisionTreeClassifier
augmented_stacking_features = np.concatenate((stacking_features, dt_pred_prob), axis=1)
```

```
# Split data into training and validation sets for final model evaluation
X_train_final, X_val, y_train_final, y_val = train_test_split(augmented_stacking_features, y_test, test_size=0.2, random_state=42)
```

```
# Train final meta-model (Logistic Regression) using augmented features
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
grid_search = GridSearchCV(LogisticRegression(), param_grid, cv=5)
grid_search.fit(X_train_final, y_train_final)
```

```
▸ GridSearchCV
▸ estimator: LogisticRegression
▸ LogisticRegression
```

```
# Print Best Parameters
print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

```
Best parameters: {'C': 0.01}
Best score: 1.0
```

```
# Use best model from GridSearchCV
best_lr_model = grid_search.best_estimator_
```

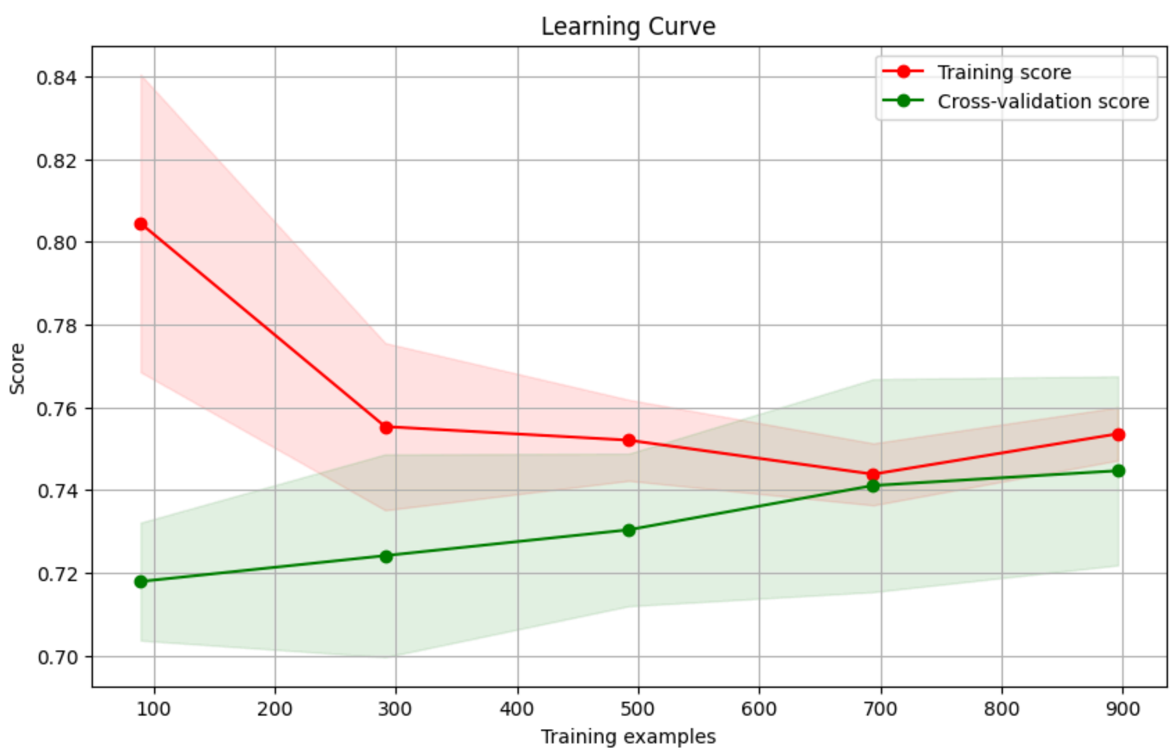
```
# Predict using the best model on validation set
final_pred_val = best_lr_model.predict(X_val)
```

```
# Calculate evaluation metrics for final stacked model on validation set
final_accuracy_val = accuracy_score(y_val, final_pred_val)
final_precision_val = precision_score(y_val, final_pred_val, average='weighted')
final_recall_val = recall_score(y_val, final_pred_val, average='weighted')
final_f1_val = f1_score(y_val, final_pred_val, average='weighted')

print("\nFinal Stacked Model Metrics on Validation Set:")
print("Accuracy:", final_accuracy_val)
print("Precision:", final_precision_val)
print("Recall:", final_recall_val)
print("F1 Score:", final_f1_val)
```

```
Final Stacked Model Metrics on Validation Set:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

```
def plot_learning_curve_with_scores(estimator, X, y, ylim=None, cv=None, n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure(figsize=(10, 6))
    plt.title("Learning Curve")
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring='accuracy')

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")

    # Print numerical scores
    print("Training Scores:", train_scores_mean)
    print("Validation Scores:", test_scores_mean)

    return plt

# Plot learning curves with numerical scores
plot_learning_curve_with_scores(lr_classifier, X_train, y_train, cv=5, n_jobs=-1)
plt.show()
```

```
Training Scores: [0.80449438 0.75532646 0.75203252 0.74380403 0.75357143]
Validation Scores: [0.71785714 0.72410714 0.73035714 0.74107143 0.74464286]
```




```
# Provided training scores
training_scores = np.array([0.80449438, 0.75532646, 0.75203252, 0.74380403, 0.75357143])
```

```
# Provided validation scores
validation_scores = np.array([0.71785714, 0.72410714, 0.73035714, 0.74107143, 0.74464286])

# Calculate mean training and validation scores
mean_training_score = np.mean(training_scores)
mean_validation_score = np.mean(validation_scores)

# Compute the difference between the mean training score and the mean validation score
difference = mean_training_score - mean_validation_score

# Calculate percentage of overfitting
percentage_overfitting = (difference / mean_training_score) * 100

print("Percentage of Overfitting: {:.2f}%".format(percentage_overfitting))
```

```
Percentage of Overfitting: 3.97%
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.