ONE-DAY AHEAD FOREX DIRECTIONAL MOVEMENT PREDICTION

A PROJECT REPORT

Submitted in partial fulfillment of the academic requirements for the award of the degree of :

MASTER OF SCIENCE

in

MATHEMATICS WITH DATA SCIENCE (Sem - II)

Submitted by ,

AHAN ROY (Roll no.: E1773U23006)

SHASWAT PATRA (Roll no.: E1773U23012)

Under the guidance of ,

Mr. KARTIK SAHOO

(Guest Faculty, IMA)

INSTITUTE OF MATHEMATICS AND APPLICATIONS,

BHUBANESWAR, ODISHA, INDIA - 751029

MAY 2024

Contents

1 Introduction

The foreign exchange (FOREX) market is the largest and most liquid financial market in the world, where currency pairs are traded around the clock. This market operates 24 hours a day, five days a week, and is influenced by a multitude of factors including economic indicators, geopolitical events, and market sentiment. Predicting the directional movement of currency pairs—whether the price will rise or fall the next day—can provide significant advantages for traders and financial institutions, aiding in decision-making processes that can lead to substantial financial gains. Despite extensive research and advancements in financial modeling, accurately forecasting short-term currency movements remains a challenging task due to the market's inherent volatility and complexity.

2 Problem Statement

The objective of this project is to develop a machine learning model using Support Vector Machine (SVM) to predict the one-day ahead directional movement (upward or downward) of the EUR/JPY currency pair. This model will leverage historical market data and employ machine learning techniques, specifically Support Vector Machine (SVM), to achieve high predictive accuracy.

3 Dataset Overview

3.1 Source

The dataset for the EUR/JPY currency pair is sourced from Yahoo Finance, which provides historical financial data for various assets, including currencies, stocks, and commodities. The specific URL for the dataset is Yahoo Finance - EUR/JPY Historical Data.

3.2 Time Period

The dataset covers the period from April 15, 2014, to April 15, 2024. This extensive time span provides a comprehensive view of the EUR/JPY currency pair's historical performance, capturing various market conditions and trends.

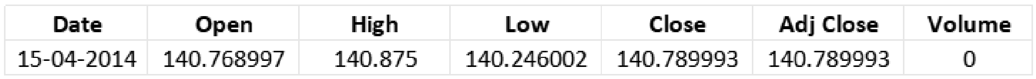
3.3 Data Attributes

The dataset includes the following attributes for each trading day:

- Date: The specific date of the trading data.
- Open: The price at which the EUR/JPY currency pair started trading on a given day.
- High: The highest price reached by the EUR/JPY currency pair during the trading day.
- Low: The lowest price reached by the EUR/JPY currency pair during the trading day.
- Close: The price at which the EUR/JPY currency pair closed at the end of the trading day.

- Adj Close: The adjusted closing price that accounts for events like dividends, stock splits, and new stock issuance, ensuring continuity of data.
- Volume: The total number of units traded during the day.

Example Data Entry :



Date	Open	High	Low	Close	Adj Close	Volume
15-04-2014	140.768997	140.875	140.246002	140.789993	140.789993	0

4 Data Preprocessing

4.1 Scaling:

It is necessary for the effective training of the SVM model. The MinMaxScaler from scikit-learn is used to normalize the features in the X array to a common range, usually between 0 and 1, to prevent features with large ranges from dominating the model.

4.2 Feature Engineering:

The Volume feature is added to the X array. The differences of the Open, High, Low, and Close prices are calculated and added to the X array as new features. This is a common technique in time series analysis to capture changes in the data.

4.3 Handling Missing Values:

The SimpleImputer from scikit-learn is used to replace missing values in the X array with the median of the respective feature.

4.4 Target Variable

The target variable 'y' was derived by calculating the sign of the difference between the current day's closing price and the previous day's closing price. The target values were 1, -1, and 0, representing an upward, downward, and no change in movement, respectively.

5 Feature Selection

The selected features for the model are :

- Open
- High
- Low
- Close

- Volume

6 Model Building

6.1 Model Selection:

Support Vector Machine (SVM) was chosen as the predictive model due to its effectiveness in binary classification tasks and its ability to handle high-dimensional feature spaces.

6.2 Splitting Data:

The train_test_split function from scikit-learn is used to split the preprocessed data into training and testing sets. The test size is set to 0.2, which means 20 % of the data will be used for testing and the remaining 80 % will be used for training.

6.3 Hyperparameter Tuning:

A grid search is performed to find the best hyperparameters for an SVM classifier. To optimize the SVM model, hyperparameter tuning was conducted using GridSearchCV, a method that systematically works through multiple combinations of parameter values to determine the best model configuration. The key hyperparameters tuned included:

- C: The regularization parameter, which controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights.
- Gamma: The kernel coefficient, which defines the influence of a single training example; low values indicate 'far' influence and high values indicate 'close' influence.
- Kernel: The function used to map the input features into higher-dimensional space; both the RBF (Radial Basis Function) and Linear kernels were considered.

The GridSearchCV function from scikit-learn is used to perform the grid search. The cv parameter is set to 5, which means the grid search will perform 5-fold cross-validation to evaluate the performance of each hyperparameter combination. The scoring parameter is set to 'accuracy', which means the grid search will use accuracy as the metric to evaluate the performance of each hyperparameter combination. The error_score parameter is set to 'raise', which means the grid search will raise an error if it encounters an invalid score.

7 Model Evaluation

7.1 Performance Metrics:

- **Accuracy**: The model achieved a high accuracy of approximately 99.4% on the test data.
- **Classification Report:** The classification report showed that the model performed exceptionally well for both the upward and downward movement classes, with precision, recall, and F1-score values close to 1.0. The model was less effective in predicting no change in movement, which could be due to the imbalanced nature of this class.

7.2 Cross-Validation:

Cross-validation was performed to ensure the model's robustness. The mean cross-validation score was approximately 99.08%, with a standard deviation of 0.33%, indicating consistent performance across different subsets of data.

8 Results & Visualizations

8.1 Hyperparameter Tuning Results:

- **Heatmap of Test Accuracy for Different C and Gamma Values:** The heatmap revealed that the model's accuracy improved with higher values of C and Gamma, emphasizing the importance of these parameters in controlling the decision boundary's complexity and the overall model's flexibility. The highest mean test accuracy of 99% was achieved with C = 100.0 and Gamma = 0.1 or Gamma = 1.0.
- **Bar Plot of Mean Test Accuracy for Different Kernel Types:** A comparison of kernel types demonstrated that the Linear kernel marginally outperformed the RBF kernel, which is consistent with the optimal parameters chosen. This finding indicates that the data has a linear separability component, and that the linear kernel was sufficient for this classification task.

8.2 Learning Curve Analysis:

The learning curve plotted shows that the model had very low training and testing error across different sizes of training data, indicating that the model was neither underfitting nor overfitting, suggesting a good balance between bias and variance, thus was well-generalized.

9 Conclusion

This project successfully developed a highly accurate Support Vector Machines for predicting the directional movement of the EUR/JPY currency pair using historical data. The model's high accuracy, combined with thorough hyperparameter tuning and evaluation, underscores the potential of machine learning in Forex market analysis. The proposed future enhancements offer pathways to further improve and generalize the model's predictive capabilities across broader market contexts.

10 Future Works

While the project achieved significant success in predicting Forex directional movements, there are several areas for further improvement:

- **Exploration of Advanced Models:** Future work could involve experimenting with more sophisticated machine learning models, such as ensemble methods (e.g., Random Forest, Gradient Boosting) or deep learning approaches, to potentially enhance predictive performance.

- **Feature Enrichment:** Incorporating additional features like macroeconomic indicators, sentiment analysis, and technical indicators could provide a richer context for the model, thereby improving its accuracy and robustness.
- **Addressing Class Imbalance:** The imbalance in the target classes could be addressed using techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or adjusting class weights to improve the model's performance on the less frequent classes.
- **Generalization to Other Currency Pairs:** Extending the analysis to include other Forex pairs would allow for a more comprehensive understanding and application of the model in different market conditions.

Code Implementation

[1]:
```
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
data = pd.read_csv("C:\\Users\\AHAN ROY\\Forex Movement Project\\EURJPY=X.csv")
data
```

[1]:
```
            Date       Open       High        Low      Close  Adj Close  \
0     2014-05-14  59.310001  59.509998  59.310001  59.310001  59.310001
1     2014-05-15  59.478001  59.590000  59.108002  59.400002  59.400002
2     2014-05-16  59.330002  59.353001  58.520000  59.330002  59.330002
3     2014-05-19  58.577999  58.588001  58.330002  58.560001  58.560001
4     2014-05-20  58.368000  58.799999  58.368000  58.368000  58.368000
...          ...        ...        ...        ...        ...        ...
2605  2024-05-08  83.469902  83.535896  83.455002  83.469902  83.469902
2606  2024-05-09  83.464302  83.511101  83.438004  83.464302  83.464302
2607  2024-05-10  83.459396  83.573898  83.448997  83.459396  83.459396
2608  2024-05-13  83.524803  83.608803  83.482002  83.524803  83.524803
2609  2024-05-14  83.486000  83.519997  83.436996  83.448997  83.448997

      Volume
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
...      ...
2605     0.0
2606     0.0
2607     0.0
2608     0.0
2609     0.0

[2610 rows x 7 columns]
```

[2]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2610 entries, 0 to 2609
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       2610 non-null   object
 1   Open       2607 non-null   float64
 2   High       2607 non-null   float64
 3   Low        2607 non-null   float64
 4   Close      2607 non-null   float64
 5   Adj Close  2607 non-null   float64
 6   Volume     2607 non-null   float64
dtypes: float64(6), object(1)
memory usage: 142.9+ KB
```

[3]:
```
from sklearn.impute import SimpleImputer
# Prepare data
X = data[['Open', 'High', 'Low', 'Close','Volume']].values
y = np.sign(data['Close'].diff())
y
```

[3]:
```
0       NaN
1       1.0
2      -1.0
3      -1.0
4      -1.0
       ...
2605   -1.0
2606   -1.0
2607   -1.0
2608    1.0
2609   -1.0
Name: Close, Length: 2610, dtype: float64
```

[4]:
```
from sklearn.impute import SimpleImputer

imp = SimpleImputer(strategy='most_frequent')
y = imp.fit_transform(y.values.reshape(-1, 1))
```

[5]:
```
from sklearn.svm import SVC
import numpy as np

# Preprocess data
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

```
# Feature engineering
X = np.append(X, np.array(data['Volume']).reshape(-1, 1), axis=1)
X = np.append(X, np.array(data['Open'].diff()).reshape(-1, 1), axis=1)
X = np.append(X, np.array(data['High'].diff()).reshape(-1, 1), axis=1)
X = np.append(X, np.array(data['Low'].diff()).reshape(-1, 1), axis=1)
X = np.append(X, np.array(data['Close'].diff()).reshape(-1, 1), axis=1)

from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='median')
X = imp.fit_transform(X)

# Reshape y
y = y.ravel()

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Hyperparameter tuning
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel':␣
 ↪['rbf', 'linear']}
grid_search = GridSearchCV(svm.SVC(), param_grid, cv=5,␣
 ↪scoring='accuracy',error_score='raise')
grid_search.fit(X_train, y_train)
```

```
[5]: GridSearchCV(cv=5, error_score='raise', estimator=SVC(),
                  param_grid={'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001],
                              'kernel': ['rbf', 'linear']},
                  scoring='accuracy')
```

```
[6]: from sklearn.ensemble import BaggingClassifier

# Train SVM classifier with best hyperparameters
clf = svm.SVC(C=grid_search.best_params_['C'], gamma=grid_search.
 ↪best_params_['gamma'], kernel=grid_search.best_params_['kernel'])

# Use bagging to reduce overfitting
bagging = BaggingClassifier(clf, n_estimators=100, max_samples=0.8,␣
 ↪max_features=1.0)
bagging.fit(X_train, y_train)

# Predict directional movement
y_pred = bagging.predict(X_test)

# Calculate accuracy and print classification report
```

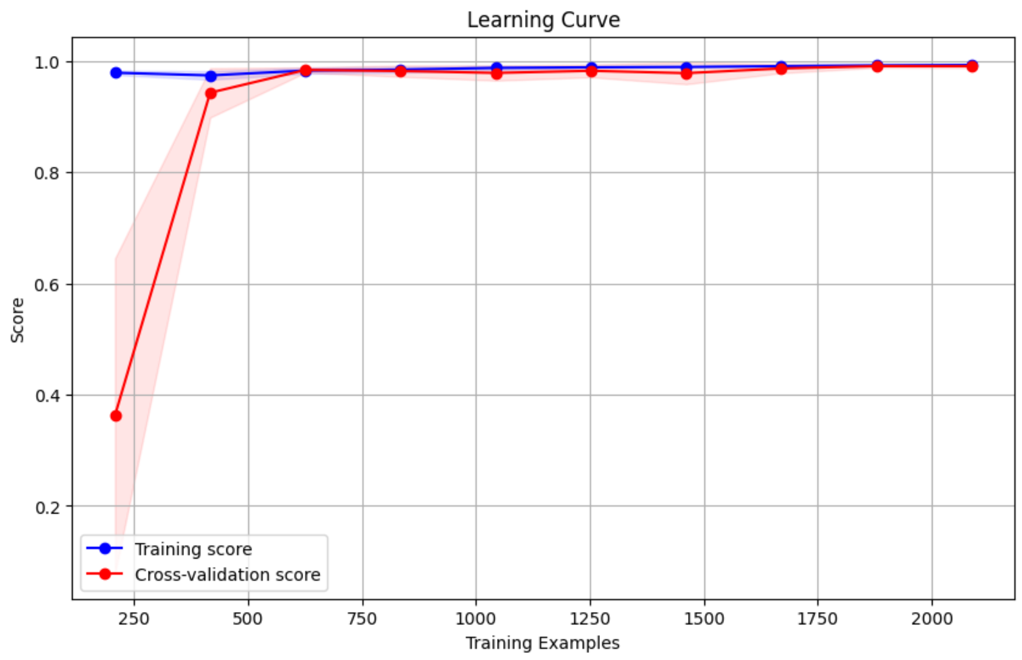
```
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
print('\nClassification Report:\n', classification_report(y_test, y_pred,␣
 ↪zero_division=1))

# Print the best hyperparameters and model
print("Best Hyperparameters:")
print(grid_search.best_params_)
print("\nBest Model:")
print(clf)
```

```
Accuracy: 0.9942528735632183

Classification Report:
               precision    recall  f1-score   support

        -1.0       1.00      0.99      0.99       269
         0.0       1.00      0.00      0.00         1
         1.0       0.99      1.00      1.00       252

    accuracy                           0.99       522
   macro avg       1.00      0.66      0.66       522
weighted avg       0.99      0.99      0.99       522

Best Hyperparameters:
{'C': 100, 'gamma': 1, 'kernel': 'linear'}

Best Model:
SVC(C=100, gamma=1, kernel='linear')
```

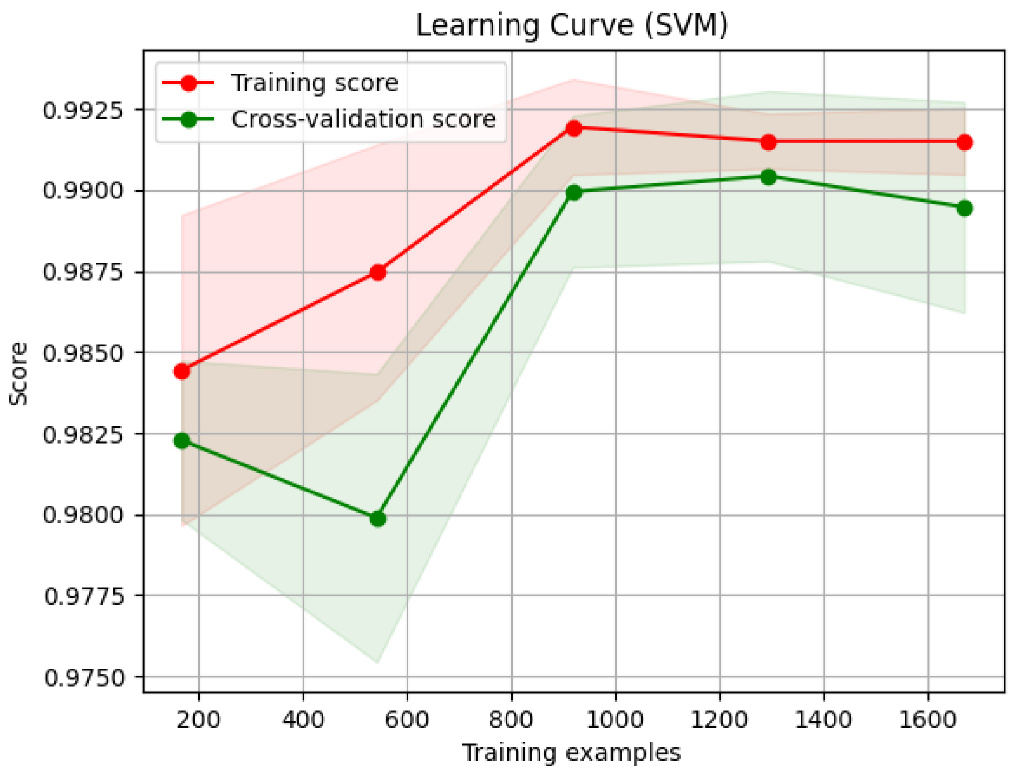
[7]:
```
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

# Compute the cross-validation score
scores = cross_val_score(clf, X, y, cv=5)

# Print the mean and standard deviation of the scores
print("Cross-validation scores:", scores)
print("Mean score:", np.mean(scores))
print("Standard deviation:", np.std(scores))
```

```
Cross-validation scores: [0.99616858 0.98659004 0.99233716 0.98850575
0.99042146]
Mean score: 0.9908045977011494
Standard deviation: 0.003295910064000998
```

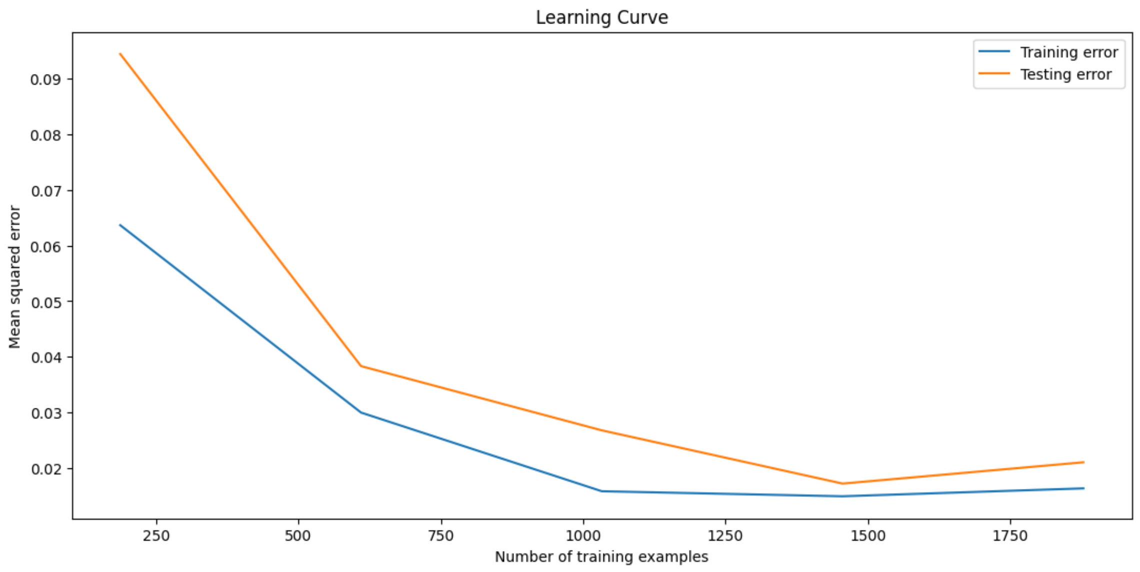
```
[8]: import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.model_selection import learning_curve

     # Define the learning curve parameters
     train_sizes, train_scores, test_scores = learning_curve(clf, X, y, cv=5,␣
      ↪train_sizes=np.linspace(0.1, 1.0, 10))

     # Calculate mean and standard deviation of training and test scores
     train_mean = np.mean(train_scores, axis=1)
     train_std = np.std(train_scores, axis=1)
     test_mean = np.mean(test_scores, axis=1)
     test_std = np.std(test_scores, axis=1)

     # Plot the learning curve
     plt.figure(figsize=(10, 6))
     plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,␣
      ↪alpha=0.1, color="b")
     plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,␣
      ↪alpha=0.1, color="r")
     plt.plot(train_sizes, train_mean, 'o-', color="b", label="Training score")
     plt.plot(train_sizes, test_mean, 'o-', color="r", label="Cross-validation␣
      ↪score")
     plt.title("Learning Curve")
     plt.xlabel("Training Examples")
     plt.ylabel("Score")
     plt.legend(loc="best")
     plt.grid(True)
     plt.show()
```



[9]:
```python
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.svm import SVC

# Define a function to plot the learning curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    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)
    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")
    return plt

# Train SVM classifier with best hyperparameters
clf = SVC(C=grid_search.best_params_['C'], gamma=grid_search.
 ↪best_params_['gamma'], kernel='linear')
clf.fit(X_train, y_train)

# Plot learning curve
title = "Learning Curve (SVM)"
plot_learning_curve(clf, title, X_train, y_train, cv=None, n_jobs=-1)
plt.show()
```

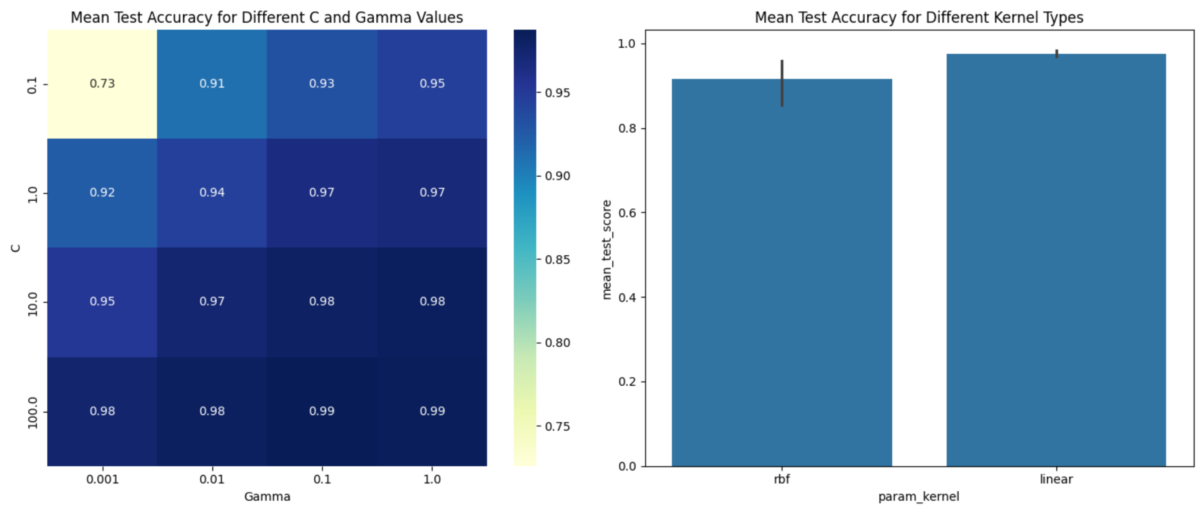


[10]:
```
# Calculate learning curve
train_sizes, train_scores, test_scores = learning_curve(clf, X_train, y_train,␣
 ↪cv=10, scoring='neg_mean_squared_error')

# Calculate mean training and testing error
train_errors = -train_scores.mean(axis=1)
test_errors = -test_scores.mean(axis=1)

# Plot learning curve
plt.figure(figsize=(13, 6))
plt.plot(train_sizes, train_errors, label='Training error')
plt.plot(train_sizes, test_errors, label='Testing error')
plt.xlabel('Number of training examples')
plt.ylabel('Mean squared error')
plt.title('Learning Curve')
plt.legend()
plt.grid(False)
plt.show()
```



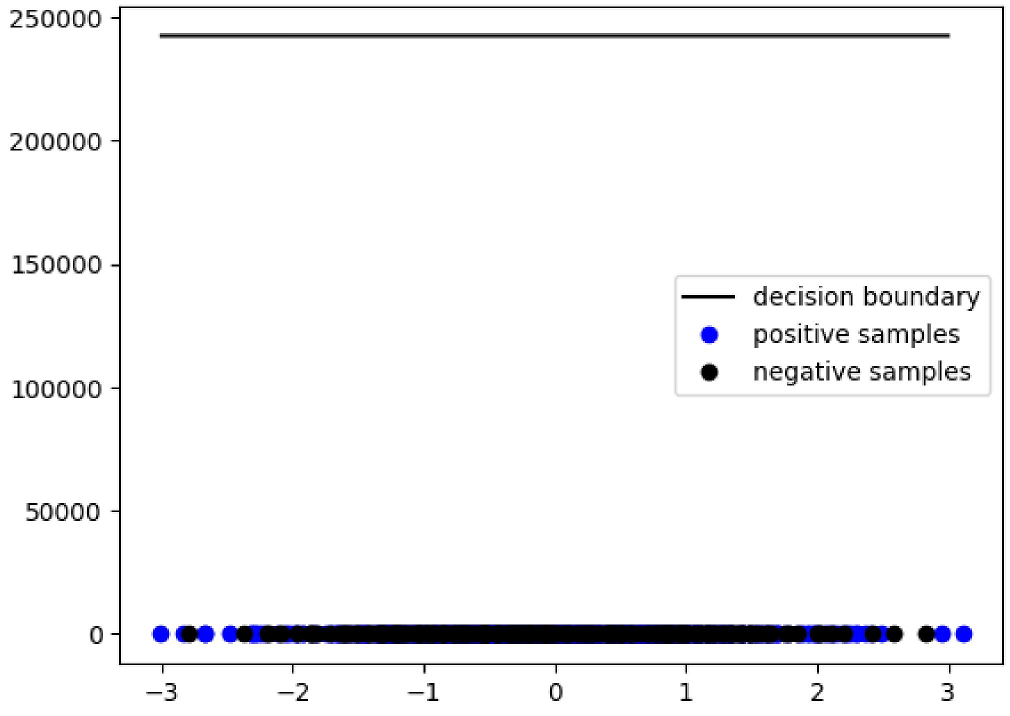
[11]:
```
# Extract grid search results
results = pd.DataFrame(grid_search.cv_results_)

# Plot grid search results for different hyperparameters
fig, axs = plt.subplots(1, 2, figsize=(14, 6))

# Plot mean test scores for different values of C and gamma
```

```
sns.heatmap(results.pivot_table(index='param_C', columns='param_gamma',␣
↪values='mean_test_score'), annot=True, cmap="YlGnBu", ax=axs[0])
axs[0].set_title('Mean Test Accuracy for Different C and Gamma Values')
axs[0].set_xlabel('Gamma')
axs[0].set_ylabel('C')

# Plot mean test scores for different kernel types
sns.barplot(x='param_kernel', y='mean_test_score', data=results, ax=axs[1])
axs[1].set_title('Mean Test Accuracy for Different Kernel Types')

plt.tight_layout()
plt.show()
```



[12]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm

# Generate some data
X = np.random.randn(1000, 2)
y = np.sign(np.random.randn(1000) + 0.5)

# Train an SVM classifier
clf = svm.SVC(kernel='linear')
clf.fit(X, y)

# Plot the decision boundary
x1 = np.linspace(-3, 3, 100)
x2 = -(clf.coef_[0][0] * x1 + clf.intercept_[0]) / clf.coef_[0][1]
```

```
plt.plot(x1, x2, 'k-', label='decision boundary')
plt.plot(X[:, 0][y==1], X[:, 1][y==1], 'bo', label='positive samples')
plt.plot(X[:, 0][y==-1], X[:, 1][y==-1], 'ko', label='negative samples')
plt.legend()
plt.show()
```



References

[1] Scikit-learn documentation for *SVM* and *GridSearchCV*

[2] Historical Foreign Exchange data from YAHOO FINANCE. Available online: https://finance.yahoo.com/quote/EURJPY%3DX/history?period1=1397520000&period2=1713139200&guccounter=1.

[3] G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning with Applications in Python* , Springer Texts in Statistics, 2023