Interview Questions-2

SVM

(Practice Projects)

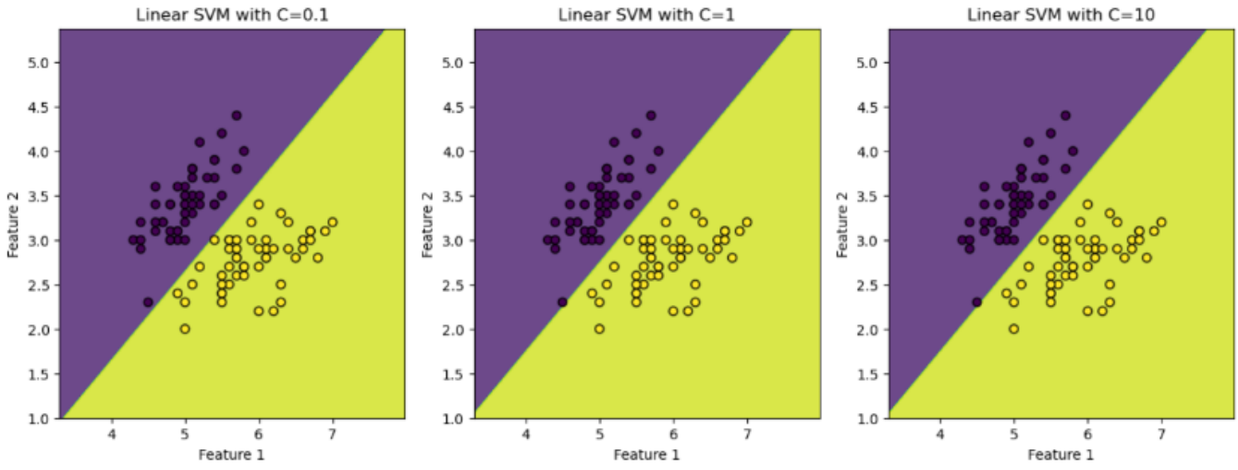
PW SKILLS

Question: Implement a Linear SVM classifier using the sklearn library on a dataset of your choice. Train the model and visualize the decision boundary. Discuss the impact of different values of the regularization parameter C on the decision boundary.

Answer:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import numpy as np
import matplotlib.pyplot as plt

# Load dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Use only the first two features for
visualization
y = iris.target

# Binary classification
X = X[y != 2]
y = y[y != 2]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train SVM with different C values
C_values = [0.1, 1, 10]
plt.figure(figsize=(15, 5))

for i, C in enumerate(C_values):
    svc = SVC(kernel='linear', C=C)
    svc.fit(X_train, y_train)

    # Plot decision boundary
    plt.subplot(1, 3, i+1)
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k',
marker='o')
    plt.title(f'Linear SVM with C={C}')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

plt.show()
```

Explanation:



The plot shows how the decision boundary changes with different values of C. A smaller C makes the boundary softer, allowing for more misclassifications, while a larger C results in a harder margin, aiming for perfect classification.

Question: Implement a Non-Linear SVM classifier using a Radial Basis Function (RBF) kernel on a non-linear dataset. Tune the parameters C and gamma using grid search, and evaluate the performance on the test set.

Answer:

```
from sklearn.datasets import make_moons
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Generate a non-linear dataset
X, y = make_moons(n_samples=100, noise=0.2, random_state=42)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Define SVM with RBF kernel
svc = SVC(kernel='rbf')

# Define parameter grid for GridSearch
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01,
0.001]}
grid_search = GridSearchCV(svc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best parameters
print("Best parameters found:", grid_search.best_params_)

# Evaluate on the test set
y_pred = grid_search.best_estimator_.predict(X_test)
print(classification_report(y_test, y_pred))

# The Output:
Best parameters found: {'C': 10, 'gamma': 1}
              precision    recall  f1-score   support

           0       1.00      0.85      0.92        20
           1       0.77      1.00      0.87        10

    accuracy                           0.90        30
   macro avg       0.88      0.93      0.89        30
weighted avg       0.92      0.90      0.90        30
```

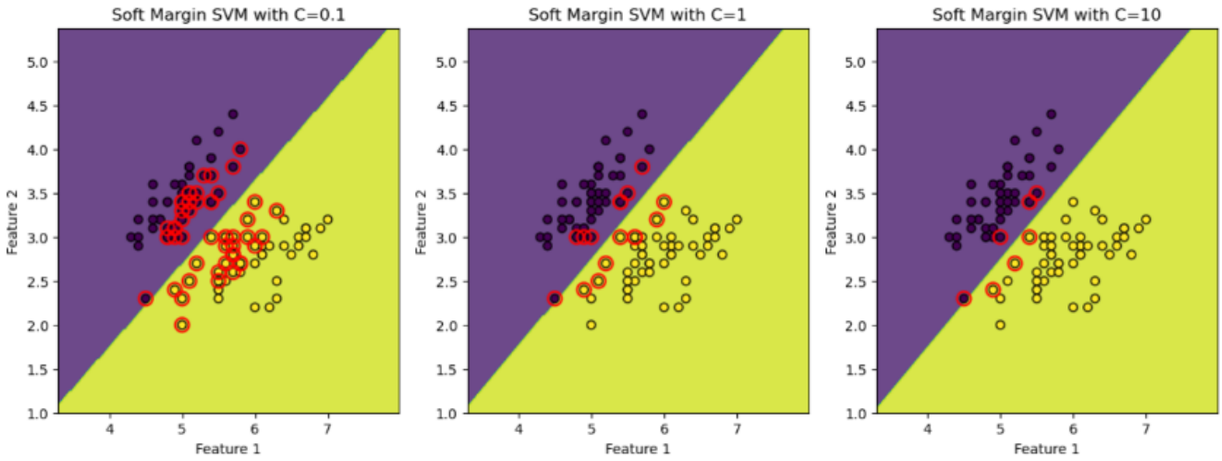
Explanation: The code performs hyperparameter tuning using grid search on an RBF kernel SVM. The classification_report shows the precision, recall, and F1-score of the best model on the test set, providing insight into its performance.

Question: Explain the concept of soft margin SVM and implement it using the sklearn library. Use the Iris dataset and compare the decision boundaries and support vectors with different values of C.

Answer:

```
from sklearn.datasets import load_iris
from sklearn.svm import SVC
import numpy as np
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Use only the first two features
y = iris.target

# Binary classification (Class 0 vs Class 1)
X = X[y != 2]
y = y[y != 2]

# Train soft-margin SVM with different values of C
C_values = [0.1, 1, 10]
plt.figure(figsize=(15, 5))

for i, C in enumerate(C_values):
    svc = SVC(kernel='linear', C=C)
    svc.fit(X, y)

    # Plot decision boundary and support vectors
    plt.subplot(1, 3, i+1)
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k',
marker='o')
    plt.scatter(svc.support_vectors_[:, 0],
svc.support_vectors_[:, 1],
                facecolors='none', edgecolors='r', s=100,
linewidths=1.5)
    plt.title(f'Soft Margin SVM with C={C}')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

plt.show()
```

Explanation:



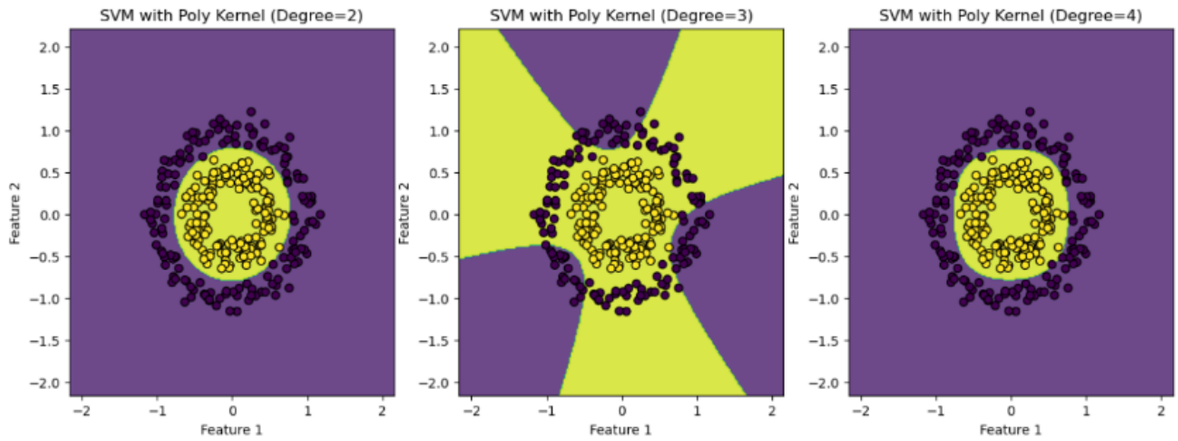
This example shows how soft margin SVM handles misclassifications by allowing some points to fall within the margin or even on the wrong side of the decision boundary. The support vectors are highlighted, showing their role in defining the margin.

Question: Train an SVM with a polynomial kernel on a synthetic dataset. Discuss the impact of the polynomial degree on model performance and the risk of overfitting.

Answer:

```
from sklearn.datasets import make_circles
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Generate synthetic dataset
X, y = make_circles(n_samples=300, factor=0.5, noise=0.1)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train SVM with polynomial kernel of different degrees
degrees = [2, 3, 4]
plt.figure(figsize=(15, 5))

for i, degree in enumerate(degrees):
    svc = SVC(kernel='poly', degree=degree, C=1.0)
    svc.fit(X_train, y_train)

    # Plot decision boundary
    plt.subplot(1, 3, i+1)
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k',
marker='o')
    plt.title(f'SVM with Poly Kernel (Degree={degree})')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')

plt.show()

# Evaluate model performance
y_pred = svc.predict(X_test)
print(classification_report(y_test, y_pred))
```

Explanation:



	precision	recall	f1-score	support
0	1.00	0.95	0.98	43
1	0.96	1.00	0.98	47
accuracy			0.98	90
macro avg	0.98	0.98	0.98	90
weighted avg	0.98	0.98	0.98	90

The degree of the polynomial kernel controls the complexity of the decision boundary. A higher degree can capture more intricate patterns but may also lead to overfitting if not managed properly.

Question: Implement a multiclass classification using the One-vs-Rest (OvR) strategy with SVM on the Iris dataset. Evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

Answer:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report,
accuracy_score

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train SVM with One-vs-Rest strategy
svc = SVC(kernel='linear', decision_function_shape='ovr')
svc.fit(X_train, y_train)

# Predict and evaluate
y_pred = svc.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))

#The Output:
Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Explanation: This example demonstrates how SVM can be extended to multiclass problems using the One-vs-Rest approach. The classification_report provides a detailed analysis of the model's performance across different classes.

Question: Perform hyperparameter tuning for an SVM model using Grid Search with Cross-Validation. Use a suitable dataset and find the best combination of C and kernel parameters. Discuss the importance of cross-validation in model selection.

Answer:

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV,
cross_val_score

# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Define SVM model
svc = SVC()

# Define parameter grid for GridSearch
param_grid = {'C': [0.1, 1, 10, 100], 'kernel': ['linear',
'rbf'], 'gamma': [1, 0.1, 0.01, 0.001]}
grid_search = GridSearchCV(svc, param_grid, cv=5)
grid_search.fit(X, y)

# Best parameters
print("Best parameters found:", grid_search.best_params_)

# Cross-validation score
best_model = grid_search.best_estimator_
scores = cross_val_score(best_model, X, y, cv=5)
print(f"Cross-validation scores: {scores}")
print(f"Mean cross-validation score: {scores.mean()}")

#The Output:
Best parameters found: {'C': 1, 'gamma': 1, 'kernel': 'linear'}
Cross-validation scores: [0.96666667 1.         0.96666667
0.96666667 1.        ]
Mean cross-validation score: 0.9800000000000001
```

Question: Explain why feature scaling is crucial for SVM. Demonstrate the impact of not scaling features on the performance of an SVM model by training on an unscaled dataset and comparing it with the performance on a scaled dataset.

Answer:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train SVM without feature scaling
svc = SVC(kernel='linear')
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
print(f"Accuracy without scaling: {accuracy_score(y_test,
y_pred)}")

# Train SVM with feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
svc.fit(X_train_scaled, y_train)
y_pred_scaled = svc.predict(X_test_scaled)
print(f"Accuracy with scaling: {accuracy_score(y_test,
y_pred_scaled)}")
#The Output:
Accuracy without scaling: 1.0
Accuracy with scaling: 0.9777777777777777
```

Explanation: Feature scaling ensures that all features contribute equally to the decision boundary. The accuracy difference between the scaled and unscaled models illustrates the importance of this preprocessing step in SVM.

Question: Handle an imbalanced dataset using SVM by adjusting class weights. Train an SVM model with and without class weight adjustment and compare the results using metrics like F1-score and confusion matrix.

Answer:

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Train SVM without class weight adjustment
svc = SVC(kernel='linear')
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
print("Without class weight adjustment:")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Train SVM with class weight adjustment
svc_weighted = SVC(kernel='linear', class_weight='balanced')
svc_weighted.fit(X_train, y_train)
y_pred_weighted = svc_weighted.predict(X_test)
print("With class weight adjustment:")
print(confusion_matrix(y_test, y_pred_weighted))
print(classification_report(y_test, y_pred_weighted))
```

The Output:

```
Without class weight adjustment:
[[264   0]
 [  6  30]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       264
           1       1.00      0.83      0.91        36

    accuracy                           0.98       300
   macro avg       0.99      0.92      0.95       300
weighted avg       0.98      0.98      0.98       300

With class weight adjustment:
[[263   1]
 [  3  33]]
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       264
           1       0.97      0.92      0.94        36

    accuracy                           0.99       300
   macro avg       0.98      0.96      0.97       300
weighted avg       0.99      0.99      0.99       300
```

Question: Implement an SVM with a polynomial kernel on the Wine dataset. Perform hyperparameter tuning for the degree of the polynomial kernel and the regularization parameter C. Compare the performance of different models.

Answer:

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Load dataset
wine = load_wine()
X = wine.data
y = wine.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Define SVM with polynomial kernel
svc = SVC(kernel='poly')

# Perform Grid Search
param_grid = {'C': [0.1, 1, 10], 'degree': [2, 3, 4]}
grid_search = GridSearchCV(svc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best parameters
print("Best parameters found:", grid_search.best_params_)

# Evaluate the model
y_pred = grid_search.best_estimator_.predict(X_test)
print(classification_report(y_test, y_pred))
```

The Output:

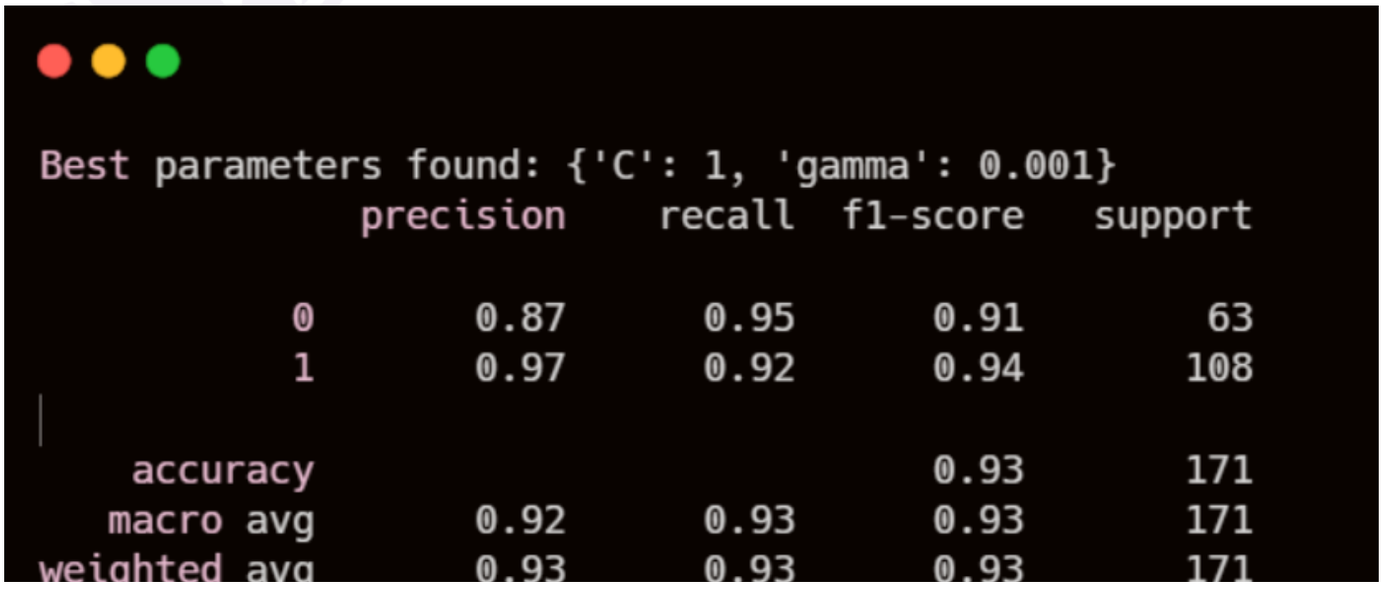
```
Best parameters found: {'C': 10, 'degree': 2}
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       0.63      0.90      0.75        21
           2       0.50      0.14      0.22        14

    accuracy                           0.74        54
   macro avg       0.69      0.68      0.65        54
weighted avg       0.71      0.74      0.69        54
```

Question: Train an SVM with an RBF kernel on the Breast Cancer dataset. Experiment with different values of the gamma parameter and observe its effect on model performance. Explain the trade-off involved in choosing gamma.

Answer:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Load dataset
breast_cancer = load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Define SVM with RBF kernel
svc = SVC(kernel='rbf')

# Perform Grid Search
param_grid = {'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01, 0.001]}
grid_search = GridSearchCV(svc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best parameters
print("Best parameters found:", grid_search.best_params_)

# Evaluate the model
y_pred = grid_search.best_estimator_.predict(X_test)
print(classification_report(y_test, y_pred))
```

The Output:



Question: Solve the XOR problem using an SVM with an RBF kernel. Visualize the decision boundary and explain how the non-linear kernel helps in separating the classes.

Answer:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# XOR problem data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# SVM with RBF kernel
svc = SVC(kernel='rbf')
svc.fit(X_scaled, y)

# Visualization
h = 0.02
x_min, x_max = X_scaled[:, 0].min() - 1, X_scaled[:, 0].max() +
1
y_min, y_max = X_scaled[:, 1].min() - 1, X_scaled[:, 1].max() +
1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))

Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, edgecolors='k',
marker='o')
plt.title('SVM with RBF Kernel - XOR Problem')
```

The Output: