A2.92 Grid Search With Weight Balancing

```
[ ]  from sklearn.model_selection import GridSearchCV
     from sklearn.svm import SVC
     from sklearn.metrics import classification_report

     # Define the class weights
     class_weight = {0: len(y_train) / (2 * sum(y_train == 0)), 1: len(y_train) / (2 * sum(y_train == 1))}

     # defining parameter range
     param_grid = {'C': [0.1, 1, 10, 100, 1000],
                   'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                   'kernel': ['rbf'],
                   'class_weight': [class_weight]}  # Add class_weight parameter to the param_grid

     grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)

     # fitting the model for grid search
     grid.fit(X_train, y_train)

     # print best parameter after tuning
     print(grid.best_params_)

     # print how our model looks after hyper-parameter tuning
     print(grid.best_estimator_)

     grid_predictions = grid.predict(X_test)
```

This code is making use of the `GridSearchCV` from Scikit-learn to perform hyperparameter tuning for a Support Vector Machine (SVM) classifier as described below.

1. **Importing Necessary Libraries**
   ```python
   from sklearn.model_selection import GridSearchCV
   from sklearn.svm import SVC
   from sklearn.metrics import classification_report
   ```
   - `GridSearchCV`: This function from Scikit-learn is used for exhaustive search over a specified parameter grid. It can help determine the best parameters to use for a given model.
   - `SVC`: Support Vector Classification, a popular machine learning classification algorithm.
   - `classification_report`: This function gives a detailed report on classification, including metrics such as precision, recall, and F1-score.

2. **Define the Class Weights**
   ```python
   class_weight = {0: len(y_train) / (2 * sum(y_train == 0)), 1: len(y_train) / (2 * sum(y_train == 1))}
   ```
   This step deals with class imbalance. If one class is represented more than the other in your training data, it might skew the model's predictions. To tackle this, the `class_weight` is defined in such a way that it inversely scales the number of samples for each class. For instance, if there are more samples of class 0 than class 1, then class 0 will have a lower weight, ensuring both classes have balanced importance during model training.

3. **Defining Parameter Grid**
   ```python
   param_grid = {'C': [0.1, 1, 10, 100, 1000],
                 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                 'kernel': ['rbf'],
                 'class_weight': [class_weight]}
   ```
   This dictionary defines the parameters we'd like to search over. `C`, `gamma`, and `kernel` are hyperparameters for the SVM.
   - `C`: Regularisation parameter. A smaller value of C creates a wider margin, which may result in more training errors but better generalisation to the test data.
   - `gamma`: Kernel coefficient. Determines the influence of individual training samples. A low value means 'far' points have an influence, while a high value means only 'close' points have any influence.
   - `kernel`: Specifies the kernel type to be used in the algorithm. `rbf` stands for Radial Basis Function.
   - `class_weight`: This parameter allows for weighting classes differently, which is especially useful for imbalanced datasets.

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=10, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=100, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
```

```
{'C': 1, 'class_weight': {0: 2.5, 1: 0.625}, 'gamma': 1, 'kernel': 'rbf'}
SVC(C=1, class_weight={0: 2.5, 1: 0.625}, gamma=1)
                  precision    recall  f1-score   support

           0          0.00         0.00      0.00        2                 [Table 5.3]
           1          0.78         1.00      0.88        7                 [Table 5.3]

    accuracy                        0.78        9
   macro avg      0.39       0.50      0.44        9
weighted avg       0.60      0.78      0.68        9
```

The idea behind this exhaustive grid search approach is to train and validate a model on every possible combination of hyperparameters to find the best combination. Once the best combination of hyperparameters is found (based on the average validation score across all folds), the model can be retrained on the entire dataset using those hyperparameters.

From the provided information, it seems that the model performance is quite consistent across different hyperparameters, with many scenarios producing a score of 0.750. However, you would typically want to analyze the results further, perhaps by calculating average scores across all folds for each set of hyperparameters, to determine the optimal set.

Decoding the Grid Search Output

Consider the following example from a grid search output:

```
[CV 1/5] END C=0.1, class_weight= {0: 2.5,1: 0.625}, gamma=1, kernel= rbf,
score =0.000 total time = 0.0s
```

[CV 1/5]: This segment of the output denotes the first iteration of the 5-fold cross-validation. Cross-validation, in this instance, divides the dataset into five segments or 'folds'. During the grid search process, the model undergoes training and evaluation five times. Each fold acts as the test dataset once, ensuring a comprehensive assessment.

Hyperparameters (END C=0.1, class_weight= {0: 2.5,1: 0.625}, gamma=1, kernel= rbf): This segment reveals the specific hyperparameters chosen for this run of the cross-validation. For the Support Vector Classifier (SVC) here, the selected parameters are C=0.1, class_weight with values {0: 2.5, 1: 0.625}, gamma set to 1, and the kernel specified as 'rbf'.

score = 0.000: This value represents the evaluation score for the fold under the specified hyperparameters. Depending on the criteria established during the GridSearchCV setup, this score could correspond to metrics like accuracy, F1-score, and more. The value of 0.000 in this fold suggests that with the chosen hyperparameters, the model's performance is not optimal.

total time = 0.0s: As the name suggests, this metric indicates the duration taken to execute this specific iteration of the cross-validation.

This output pertains to the results of a hyperparameter tuning process that appears to be using cross-validation for a Support Vector Machine (SVM) with the Radial Basis Function (RBF) kernel. The machine learning process in context is grid search, a popular method for optimising hyperparameters. Let's break down the given information:

1. **"Fitting 5 folds for each of 25 candidates, totalling 125 fits"**: This tells us that a 5-fold cross-validation process is being used, and there are 25 unique combinations of hyperparameters (i.e., 25 candidates) that are being tried out. In 5-fold cross-validation, the dataset is divided into 5 equal parts, and in each iteration, 4 parts are used for training while the remaining part is used for validation. The total number of fits (125) is the product of the number of candidates (25) and the number of folds (5).

2. **Each [CV X/5] line**: Each line provides the results for one of the fits:

   - **CV X/5**: This indicates which fold out of the 5 is currently being validated.
   - **END**: Indicates the end of the fit for the given set of parameters.
   - **C, gamma, kernel, and class_weight**: These are hyperparameters for the SVM.
      - **C**: Regularization parameter. A smaller value of C creates a wider margin, which may allow more misclassifications. A larger value of C creates a narrower margin and will try to classify all the training samples correctly.
      - **gamma**: Defines how far the influence of a single training set reaches. A low gamma value means ‘far’ and high value means ‘close’. It can be thought of as the inverse of the radius of influence of samples selected by the model as support vectors.
      - **kernel**: Specifies the kernel type to be used in the algorithm. In this case, the RBF kernel is used.
      - **class_weight**: Used for unbalanced classes. If not given, all classes are supposed to have weight one. For example, `class_weight={0: 2.5, 1: 0.625}` means that for the purpose of classification, every instance of class 0 is treated as 2.5 instances and every instance of class 1 is treated as 0.625 instances.
   - **score**: This indicates the accuracy (or some other scoring metric, depending on the configuration) of the model on the validation set for the given fold and hyperparameters.
   - **total time**: This is the time taken to fit the model and validate it on the fold for the given set of hyperparameters.

Truncated output generated from the above code

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.1, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.01, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 5/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 1/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.000 total time=   0.0s
[CV 2/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 3/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
[CV 4/5] END C=0.1, class_weight={0: 2.5, 1: 0.625}, gamma=0.0001, kernel=rbf;, score=0.750 total time=   0.0s
...
    accuracy                           0.78         9
   macro avg       0.39      0.50      0.44         9
weighted avg       0.60      0.78      0.68         9
```

The provided results describe the process and outcome of hyperparameter tuning using cross-validation for a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel. Let's break down the information:

1. **Hyperparameter Tuning**:
   - **Folds**: The process uses 5-fold cross-validation. In 5-fold CV, the dataset is divided into 5 parts; 4 parts are used for training and 1 part for testing. This process is repeated 5 times, using a different part for testing each time.
   - **Candidates**: There are 25 different combinations of hyperparameters being tested.
   - **Total Fits**: A total of 125 fits were performed, which means each of the 25 combinations was tested using 5-fold CV.

   - From the results, various combinations of the hyperparameters (`C`, `gamma`, and `class_weight`) are being tested. For example, for `C=0.1`, `class_weight={0: 2.5, 1: 0.625}`, we see that different values of `gamma` (1, 0.1, 0.01, 0.001, and 0.0001) were tried.

2. **Cross-Validation Scores**:
   - Each combination of hyperparameters is associated with 5 scores, one for each fold of the CV.
   - For the given hyperparameters, the score is consistently `0.750` for the 2nd, 3rd, 4th, and 5th folds, but `0.000` for the 1st fold. This suggests that the model might not be performing consistently across different subsets of the data.

3. **Classification Report**:
   - **Accuracy**: 0.78 indicates that the model correctly predicted the class 78% of the time on a dataset of 9 samples.
   - **Precision**: For class 0, it's undefined (which might be due to no positive predictions for this class), and for class 1, it's 0.78. Precision indicates how many of the positive predictions were actually correct.
   - **Recall**: For class 0, it's 0.00 (which means the model didn't correctly predict any true positives for this class), and for class 1, it's 1.00. Recall indicates how many of the actual positives were predicted correctly.
   - **F1-score**: Harmonic mean of precision and recall. For class 0, it's undefined (due to the 0 precision and recall), and for class 1, it's 0.88. F1-score is useful when the class distribution is imbalanced.
   - **Support**: Number of actual occurrences of the class in the specified dataset. 2 occurrences for class 0 and 7 occurrences for class 1.
   - **Macro avg**: Arithmetic average of the metric scores for each class. Here, the average is taken between the two classes.
   - **Weighted avg**: Average of the metric scores for each class, weighted by the number of samples in each class.

This output is the result of performing grid search cross-validation as detailed below:

1. Grid Search: It's a method for hyperparameter tuning where a 'grid' of