A2.91 Grid Search Without Weight Balancing

```
[ ] from sklearn.model_selection import GridSearchCV

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

    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)

    # print classification report
    print("GridSearch withOut Class Weight Balancing - Classification Report (Test Set Results):")
    print(classification_report(y_test, grid_predictions))

    print("GridSearch withOut Class Weight Balancing - Classification Report (Train Set Results):")
    print(classification_report(y_train, grid.predict(X_train)))
```

Code description is given below:

1. **Import GridSearchCV**:
   - `from sklearn.model_selection import GridSearchCV`: Imports the GridSearchCV class, which performs exhaustive searching over specified parameter values for an estimator.

2. **Parameter Grid Definition**:
   - `param_grid`: This dictionary contains the parameters for the SVM and their possible values:
     - `C`: Regularisation parameter. The strength of the regularisation is inversely proportional to `C`. Smaller values of `C` will result in a wider margin, which may admit some misclassifications.
     - `gamma`: Kernel coefficient for the Radial Basis Function (`'rbf'`). It defines how far the influence of a single training example reaches, with low values meaning far and high values meaning close.
     - `kernel`: Specifies the kernel type to be used in the algorithm. Here, only the RBF kernel is considered.

3. **GridSearchCV Object Creation**:
   - `grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)`:
     - The `SVC()` estimator, representing a Support Vector Classifier, is used.
     - `refit=True` tells the search to retrain the best found model on the entire dataset.
     - `verbose=3` means the method will display messages during its execution, which helps in understanding the progress.

4. **Fit the Model for Grid Search**:
   - `grid.fit(X_train, y_train)`: This command starts the grid search. It will train the `SVC()` model using all the combinations of parameters given in `param_grid`.

5. **Display Results**:
   - `print(grid.best_params_)`: Shows the combination of parameters that achieved the best results on the validation set during the grid search.
   - `print(grid.best_estimator_)`: Provides a view of the best estimator (model) after hyperparameter tuning.

6. **Predictions with the Best Model**:
   - `grid_predictions = grid.predict(X_test)`: Uses the model with the best hyperparameters to make predictions on the test set.

7. **Classification Report**:
   - `print(classification_report(y_test, grid_predictions))`: Displays metrics like precision, recall, f1-score for each class, and overall accuracy, giving insights into

```
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
```

GridSearch withOut Class Weight Balancing – Classification Report (Test Set Results):

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2
1	0.78	1.00	0.88	7
accuracy			0.78	9
macro avg	0.39	0.50	0.44	9
weighted avg	0.60	0.78	0.68	9

GridSearch withOut Class Weight Balancing – Classification Report (Train Set Results):

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.80	1.00	0.89	16
accuracy			0.80	20
macro avg	0.40	0.50	0.44	20
weighted avg	0.64	0.80	0.71	20

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 hyperparameters is prepared and the model is trained for each combination of parameters to find the best combination.

2. Cross-Validation: It's a method for assessing how the results of a statistical analysis (model in this case) will generalize to an independent data set. In this case, a 5-fold cross-validation is being used, which means the dataset is split into 5 equal parts; 4 of them are used for training while the fifth one is used for validation. This process is repeated 5 times, so each part acts as the validation set once.

A summary of the code is given below:

- Fitting 5 folds for each of 25 candidates, totalling 125 fits**: This means that the grid search will test 25 different combinations of hyperparameters, and for each combination, it will do a 5-fold cross-validation. So, 25 (combinations) x 5 (folds) = 125 fits/tests in total.

- Each line after this is the result of one such fit/test:
  - **[CV 1/5]**: This is the first fold/test of the current combination.
  - **END** .....C=0.1, gamma=1, kernel=rbf;**: This states the hyperparameters used for this specific test. **SVM** with a radial basis function kernel (rbf) has two main hyperparameters:
    - **C**: Regularization parameter. Smaller values mean stronger regularization.
    - **gamma**: Defines how far the influence of a single training example reaches. Low values mean far and high values mean close.
  - **score=1.000**: This is the accuracy (or other metric depending on the grid search configuration) achieved with this hyperparameter combination on this fold.
  - **total time= 0.0s**: This is the time to complete this fold/test.

The goal of this process is to identify the combination of hyperparameters (C and gamma in this case) that gives the best performance (score) on the validation set. The best combination is then typically used to train a model on the entire dataset or to evaluate on a separate test set.

These results show the performance of a support vector classifier (`SVC` with kernel `rbf`) during a grid search cross-validation procedure. Here is an interpretation of the provided outputs:

1. **Initial Information**:
   - `Fitting 5 folds for each of 25 candidates, totalling 125 fits`: Grid search is being done with 5-fold cross-validation. Given the provided `param_grid`, the total combinations of hyperparameters is 25 (5 values of `C` multiplied by 5 values of `gamma`). For each of these combinations, a 5-fold cross-validation is performed, resulting in a total of 125 fits.

2. **Parameter Details**:
   - For each line of the results, the specific combination of hyperparameters being tested is shown:
     - `C`: The regularisation parameter.
     - `gamma`: The kernel coefficient.
     - `kernel`: The type of kernel used (`rbf` in this case).
     - `class_weight`: Weights associated with classes `0` and `1`. In the provided output, the weight for class `0` is `2.5`, and for class `1`, it's `0.625`.

3. **Results for Each Fold**:
   - The output is grouped by the combination of hyperparameters. For instance, all the lines with `C=0.1`, `class_weight={0: 2.5, 1: 0.625}`, and `gamma=1` represent 5 cross-validation results for that specific combination.
   - `[CV x/5]`: This shows which fold (out of 5) the result corresponds to.
   - `score`: This represents the performance (usually accuracy) of the model on the validation set for that fold.
   - `total time`: Represents the time taken for fitting and scoring that fold.

4. **Observed Patterns**:
   - For the combination `C=0.1`, `class_weight={0: 2.5, 1: 0.625}` and various `gamma` values, the first fold always has a score of `0.000`. It implies that this specific fold might have some characteristics that make it challenging for this model with these hyperparameters.
   - For all other folds in these combinations, the score remains consistent at `0.750`, indicating consistent performance for these particular data splits.

In general, these results offer a granular look at how the model performs under different hyperparameters and across different data splits. After evaluating all combinations, `GridSearchCV` would choose the hyperparameters that result in the best average performance across all folds.

---

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

Evaluating Algorithm Validity in Grid Search

To holistically evaluate the algorithm's validity, especially in imbalanced datasets, a combination of metrics should be considered:

Accuracy: This metric is straightforward – it represents the fraction of correctly predicted instances relative to the entire dataset's instances. However, accuracy has its limitations, especially with imbalanced datasets. High accuracy in such cases might be skewed by the majority class, potentially masking the true model performance.

Macro Avg: The macro average evaluates metrics (like precision, recall, and the F1-score) for each class independently. After this independent evaluation, the unweighted mean of these values is computed. This approach is egalitarian – it values each class's performance equally, irrespective of its frequency. This metric is especially beneficial when dealing with imbalanced datasets, ensuring that minority classes are not overshadowed.

Weighted Avg: The weighted average, like the macro average, evaluates metrics individually for each class. The pivotal difference is that the weighted average also considers the frequency of each class. Classes with a higher frequency have a greater influence on this metric. For imbalanced datasets, the weighted average offers nuanced insights by emphasising the minority class's performance.

When appraising an algorithm's validity on imbalanced datasets using grid search, it's pivotal to look beyond just accuracy. Macro and weighted averages provide highlight the algorithm's performance across both minority and majority classes. The weighted average, in particular, accounts for class imbalances.

These results are the performance metrics for a classifier's predictions on a test set containing 9 samples.

Here's a breakdown of the results:

1. **Accuracy**:
   - The model achieved an accuracy of 0.78 on the test set. This means that 78% of the test samples were correctly classified.

2. **Macro Average**:
   - Precision, Recall, and F1-Score can be averaged in multiple ways to provide an overall picture of the model's performance. Macro averaging calculates metrics for each label and finds their unweighted mean. This does not take label imbalance into account.
     - `macro avg` of 0.39 for precision suggests that on average, the model's precision across the classes isn't too high.
     - A `recall` macro avg of 0.50 means the model, on average, identifies 50% of the relevant instances across classes.
     - The `f1-score` macro avg of 0.44 suggests a moderately low harmonic mean