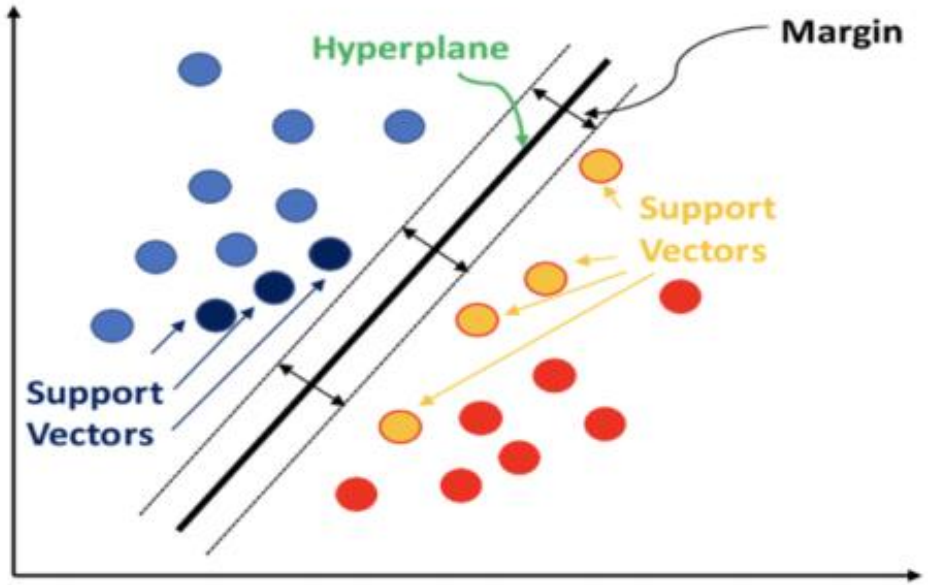
IMPLEMENTATION OF SUPPORT VECTOR MACHINES FOR AD-CLICK PREDICTION AND WINE-TYPE PREDICTION



DATASET OVERVIEW

1. Ad-click Prediction:

- Source: Kaggle
- Number of observations: 400
- Number of features: 5
- Number of classes in target feature: 2

2. Wine-type prediction:

- Source: UCI Machine Learning Repository
- Number of observations: 178
- Number of features: 17
- Number of classes in target feature: 3

DATA IMBALANCE AND BASELINE ACCURACY

EDA AND DATA CLEANING:

- The dataset did not have NULL values.
- The categorical features were encoded.

DATA IMBALANCE:

- The target feature had twice as much as not_clicked class as compared to clicked class.
- Used class weights method to handle data imbalance in the target feature.
- Sklearn's compute_class_weight is used to set class weights. Then the class weights are multiplied with the computed class weights so that the value that results is the same across all the classes.

BASELINE ACCURACY:

- Next, baseline accuracy of the model is computed. The DummyClassifier predicting the 'most_frequent' class is trained to determine the baseline accuracy. The baseline accuracy is identified as 64.2%. Our SVM must perform better than this.

Dataset Imbalance

```
In [15]: y.Purchased.value_counts()
Out[15]: Purchased
         not_clicked    257
         clicked        143
         Name: count, dtype: int64

In [16]: y.Purchased.value_counts(normalize = True)
Out[16]: Purchased
         not_clicked    0.6425
         clicked        0.3575
         Name: proportion, dtype: float64

In [17]: class_weights = compute_class_weight('balanced', classes = list(y.Purchased.cat.categories), y = y.Purchased)
         class_weights
Out[17]: array([1.3986014 , 0.77821012])

In [18]: class_weight_dict = dict(zip(list(y.Purchased.cat.categories), class_weights))
         class_weight_dict
Out[18]: {'clicked': 1.3986013986013985, 'not_clicked': 0.7782101167315175}

In [19]: for v in y.Purchased.unique():
           print(f"Class '{v}': {y.Purchased.value_counts()[v]} (counts) x {class_weight_dict[v]:.3f} (weight) = {y.Purchased.value_counts
         Class 'not_clicked': 257 (counts) x 0.778 (weight) = 200.000
         Class 'clicked': 143 (counts) x 1.399 (weight) = 200.000
```

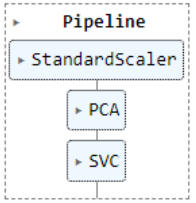
Baseline Accuracy

```
In [20]: dc = DummyClassifier(strategy = 'most_frequent')
         dc.fit(X, y.Purchased)

         acc = accuracy_score(y.Purchased, dc.predict(X))

         print(f'Baseline accuracy = {acc:.3f}')

         Baseline accuracy = 0.642
```

HYPERPARAMETER TUNING WITH DIFFERENT KERNELS AND NON-NESTED CROSS VALIDATION

- Feature Standardisation and PCA are added to ML Pipeline
- SVMs are trained on the dataset
- Hyperparameter tuning is performed on C, gamma and pca values
- The hyperparameters are tested on different SVM kernels (rbf, poly and linear) and the best model parameters are chosen using StratifiedKFold GridSearch CV.
- RBF Kernel is identified to provide the best performance. The best hyperparameters are also identified.

```
ml_pipeline = make_pipeline( StandardScaler(), PCA(), SVC() )
ml_pipeline
```




```
In [32]: param_C_values = [1, 10, 100, 1000]
         param_gamma_values = [0.1, 0.01, 0.001, 0.0001]
         param_pca_values = [0.7, 0.8, 0.85, 0.90, 0.95]

         param_grid = [
                       { 'svc__kernel' : ['linear'],
                         'svc__C' : param_C_values,
                         'svc__class_weight' : [None, class_weight_dict],
                         'pca__n_components' : param_pca_values
                       },
                       { 'svc__kernel' : ['poly'],
                         'svc__C' : param_C_values,
                         'svc__degree' : [2, 3, 4, 5, 6],
                         'svc__gamma' : param_gamma_values,
                         'svc__class_weight' : [None, class_weight_dict],
                         'pca__n_components' : param_pca_values
                       },
                       { 'svc__kernel' : ['rbf'],
                         'svc__C' : param_C_values,
                         'svc__gamma' : param_gamma_values,
                         'svc__class_weight' : [None, class_weight_dict],
                         'pca__n_components' : param_pca_values
                       }
         ]
```

```
best_model_params = gs.best_params_
best_model_params
```

```
{'pca__n_components': 0.8,
 'svc__C': 100,
 'svc__class_weight': {'clicked': 1.3986013986013985,
  'not_clicked': 0.7782101167315175},
 'svc__gamma': 0.1,
 'svc__kernel': 'rbf'}
```

```
cv_folds = StratifiedKFold(n_splits = 5, shuffle = True)

gs = GridSearchCV(estimator = ml_pipeline,
                  param_grid = param_grid,
                  #cv = 5,
                  cv = cv_folds,
                  scoring = 'accuracy',
                  return_train_score = True,
                  verbose = 5)
```

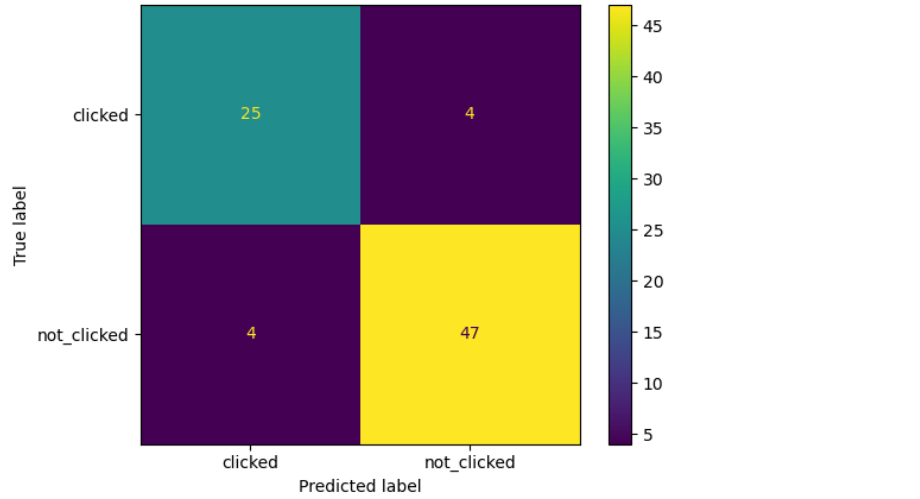
MAKING PREDICTIONS ON TEST SET

- Once the best parameters are identified, the best model is re-trained on the full training set.
- The best model is used to make predictions on the test set
- The overall accuracy of the model is 90%, which outperforms our baseline model.
- The model's performance is strong, with high precision, recall, and F1-scores for both classes, indicating that it effectively handles the classification task.

```
In [41]: from sklearn.metrics import classification_report, ConfusionMatrixDisplay

         print(classification_report(y_test, best_model.predict(X_test)))
```

	precision	recall	f1-score	support
clicked	0.86	0.86	0.86	29
not_clicked	0.92	0.92	0.92	51
accuracy			0.90	80
macro avg	0.89	0.89	0.89	80
weighted avg	0.90	0.90	0.90	80

```
ConfusionMatrixDisplay.from_predictions(y_test, best_model.predict(X_test))
plt.show()
```



NESTED CROSS VALIDATION APPROACH

- Nested cross-validation is performed to get an unbiased estimate of our SVM model's accuracy.
- The mean score using nested cross-validation is found to be 0.900 +/- 0.018
- Then we re-train the model on the full dataset.
- The model gives an accuracy of 91%.

```
from sklearn.model_selection import cross_val_score

inner_cv_folds = StratifiedKFold(n_splits = 5, shuffle = True)
outer_cv_folds = StratifiedKFold(n_splits = 5, shuffle = True)

# inner loop for parameter search
gs = GridSearchCV(estimator = ml_pipeline,
                  param_grid = param_grid,
                  cv = inner_cv_folds,
                  scoring = 'accuracy',
                  verbose = 0)

# outer loop for accuracy scoring
scores = cross_val_score(gs, X = X, y = y.Purchased, cv = outer_cv_folds, verbose = 99)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV] START ....................................................................
[CV] END .............................. score: (test=0.925) total time=  47.8s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:   47.8s remaining:    0.0s
[CV] START ....................................................................
[CV] END .............................. score: (test=0.912) total time=  47.7s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:  1.6min remaining:    0.0s
[CV] START ....................................................................
[CV] END .............................. score: (test=0.900) total time=  46.3s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.4min remaining:    0.0s
[CV] START ....................................................................
[CV] END .............................. score: (test=0.875) total time=  44.1s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:  3.1min remaining:    0.0s
[CV] START ....................................................................
[CV] END .............................. score: (test=0.887) total time=  46.9s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  3.9min remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  3.9min finished
```

```
print(f'Mean score using nested cross-validation: {scores.mean():.3f} +/- {scores.std():.3f}')
```

```
Mean score using nested cross-validation: 0.900 +/- 0.018
```

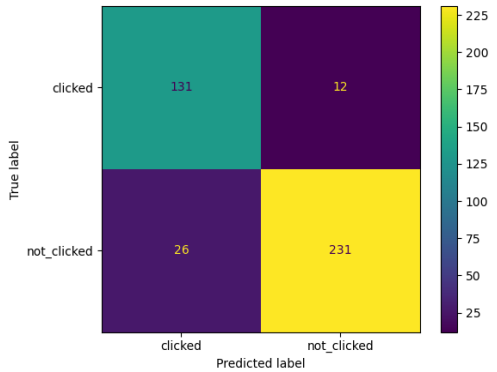
```
gs.best_params_
```

```
{'pca__n_components': 0.8,
 'svc__C': 1000,
 'svc__class_weight': {'clicked': 1.3986013986013985,
  'not_clicked': 0.7782101167315175},
 'svc__gamma': 0.01,
 'svc__kernel': 'rbf'}
```

```
print(classification_report(y, best_model.predict(X)))
```

```
              precision    recall  f1-score   support

     clicked       0.83      0.92      0.87       143
 not_clicked       0.95      0.90      0.92       257

    accuracy                           0.91       400
   macro avg       0.89      0.91      0.90       400
weighted avg       0.91      0.91      0.91       400
```

```
ConfusionMatrixDisplay.from_predictions(y, best_model.predict(X))
plt.show()
```



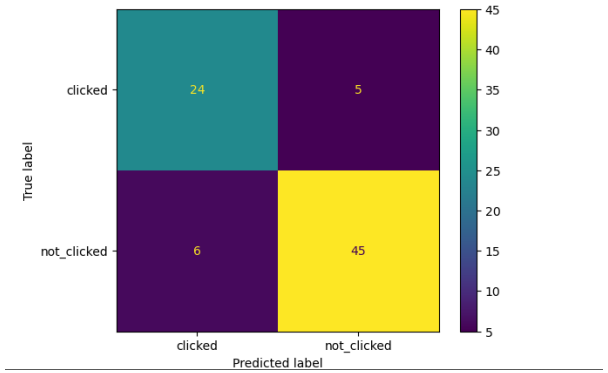
HANDLING CLASS IMBALANCES WITH DIFFERENT APPROACHES

Without Handling Data Imbalance

```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay

print(classification_report(y_test, best_model.predict(X_test)))
```

```
              precision    recall  f1-score   support

     clicked       0.80      0.83      0.81        29
 not_clicked       0.90      0.88      0.89        51

    accuracy                           0.86        80
   macro avg       0.85      0.85      0.85        80
weighted avg       0.86      0.86      0.86        80
```

```
ConfusionMatrixDisplay.from_predictions(y_test, best_model.predict(X_test))
plt.show()
```

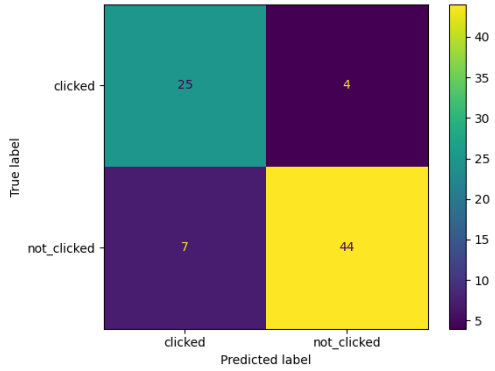


Synthetic Minority Over-sampling Technique (SMOTE) Approach:

```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay

print(classification_report(y_test, best_model.predict(X_test)))
```

```
              precision    recall  f1-score   support

     clicked       0.78      0.86      0.82        29
 not_clicked       0.92      0.86      0.89        51

    accuracy                           0.86        80
   macro avg       0.85      0.86      0.85        80
weighted avg       0.87      0.86      0.86        80
```

```
ConfusionMatrixDisplay.from_predictions(y_test, best_model.predict(X_test))
plt.show()
```

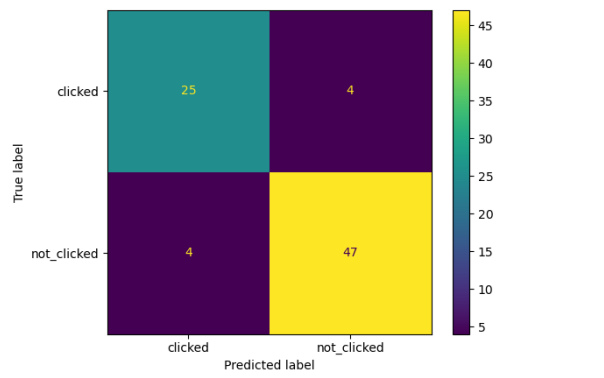


Random Under Sampler Approach:

```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay

print(classification_report(y_test, best_model.predict(X_test)))
```

```
              precision    recall  f1-score   support

     clicked       0.86      0.86      0.86        29
 not_clicked       0.92      0.92      0.92        51

    accuracy                           0.90        80
   macro avg       0.89      0.89      0.89        80
weighted avg       0.90      0.90      0.90        80
```

```
ConfusionMatrixDisplay.from_predictions(y_test, best_model.predict(X_test))
plt.show()
```



HANDLING CLASS IMBALANCES WITH DIFFERENT APPROACHES

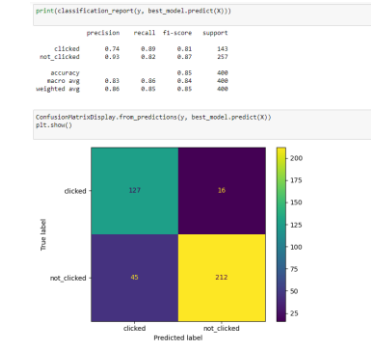
INTERPRETATION OF RESULTS:

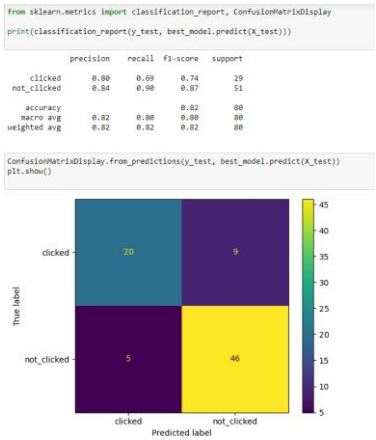
Although all the approaches produced an accuracy of 91% on the full dataset, the accuracy changed in the test set for different approaches.

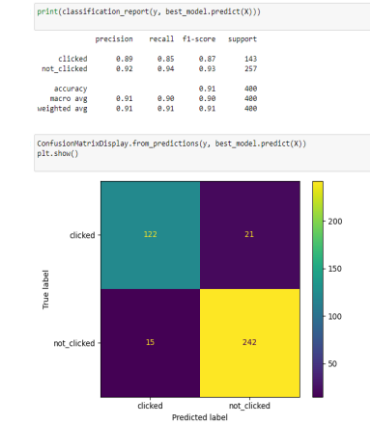
- CLASS WEIGHTS APPROACH:
  - Balanced performance, suggesting effective handling of class imbalance.
- RandomUnderSampler APPROACH:
  - Similar performance to class weights, indicating effective handling of class imbalance.
- SMOTE APPROACH:
  - Balanced performance, but not as high as class weights or random under-sampling.
- NO APPROACH:
  - Balanced performance but slightly lower than when using imbalance handling techniques.

IMPLEMENTATION OF SVM WITH LINEAR AND POLYNOMIAL KERNELS

- Next, we implemented the SVM model with linear and polynomial kernels, to study if there was a difference in performance.
- RBF Kernel achieves the highest accuracy both on the test data (90%) and the full dataset (91%).
- Linear Kernel has an accuracy of 88% on the test data and 85% on the full dataset.
- Polynomial Kernel has the lowest accuracy on the test data (82%) but performs well on the full dataset (91%).
- It is confirmed that **the RBF kernel outperforms Linear and Polynomial Kernels.**





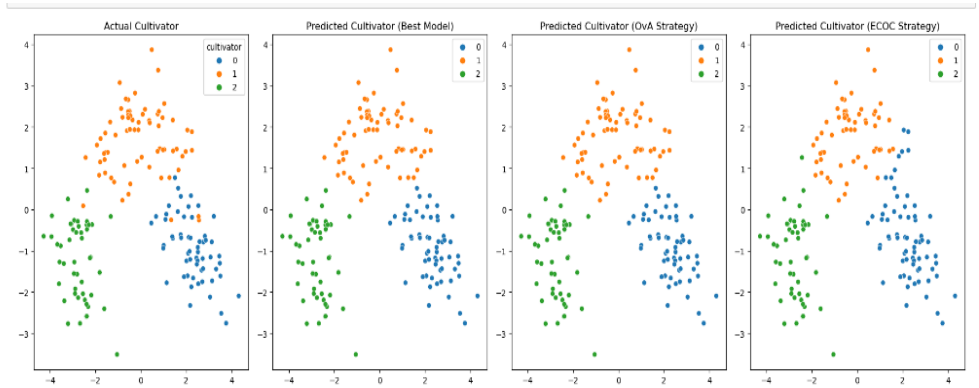


SVM FOR MULTI-CLASS CLASSIFICATION AND EVALUATION

- Wine dataset was used to study how SVM models performed on a multi-class problem
- Evaluation of the Best Model:
  - The best model obtained from GridSearchCV is evaluated on the test data.
  - Accuracy and classification report metrics are computed.
- One-vs-All (OvA) Strategy:
  - The best model is wrapped in a OneVsRestClassifier to implement the OvA strategy.
  - The model is trained and evaluated using OvA.
- Error Correcting Output Codes (ECOC) Strategy:
  - The best model is used to implement the ECOC strategy.
  - The model is trained and evaluated using ECOC.

It is understood that the best model obtained from GridSearchCV and One-vs-All outperformed Error Correcting Output Codes.

- This could be attributed to a variety of reasons:
  - ➢ While ECOC also benefits from hyperparameter tuning, the complexity of its coding matrix might make it harder to find the optimal combination of hyperparameters compared to a simpler OvA strategy.
  - ➢ The coding and decoding process in ECOC can be less intuitive, making it harder to diagnose and improve specific parts of the model.
  - ➢ The superior performance of the model obtained from GridSearchCV with the One-vs-All strategy over the Error Correcting Output Codes method can be due to better hyperparameter optimization, simpler and more efficient training, and alignment with the characteristics of the dataset.



Classification Report for Best Model:

	precision	recall	f1-score	support
0	0.95	0.98	0.97	59
1	0.99	0.94	0.96	71
2	0.98	1.00	0.99	48
accuracy			0.97	178
macro avg	0.97	0.98	0.97	178
weighted avg	0.97	0.97	0.97	178

Classification Report for OvA Strategy:

	precision	recall	f1-score	support
0	0.95	0.98	0.97	59
1	0.99	0.94	0.96	71
2	0.98	1.00	0.99	48
accuracy			0.97	178
macro avg	0.97	0.98	0.97	178
weighted avg	0.97	0.97	0.97	178

Classification Report for ECOC Strategy:

	precision	recall	f1-score	support
0	0.86	1.00	0.92	59
1	1.00	0.83	0.91	71
2	0.96	1.00	0.98	48
accuracy			0.93	178
macro avg	0.94	0.94	0.94	178
weighted avg	0.94	0.93	0.93	178