Individual_Assignment_ML_final27

March 27, 2025

Predictive Modelling UK Department For Transport Road Safety Data dataset (2023)

0.1 Table of Contents

0.2 Importing Libraries and Preparing Environment

[114]:
```
# ==============================
# System & Utilities
# ==============================
import os
import json
import gzip
import warnings
from timeit import default_timer as timer
from datetime import timedelta
from datetime import datetime
import logging
logging.basicConfig()
logging.getLogger("SKLEARNEX").setLevel(logging.ERROR)
```

```python
# ==============================
# Data Handling, Preprocessing & Cleaning
# ==============================
import numpy as np
import pandas as pd
from scipy import stats
from sklearn.exceptions import ConvergenceWarning
from sklearn.experimental import enable_iterative_imputer  # noqa
from sklearn.impute import IterativeImputer, SimpleImputer
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from IPython.display import display

# ==============================
# Plotting
# ==============================
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import learning_curve
# ==============================
# Model Evaluation & Metrics
# ==============================
from sklearn.metrics import (
    classification_report,
    ConfusionMatrixDisplay,
    precision_recall_fscore_support,
    accuracy_score,
    precision_score,
    recall_score
)
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    cross_val_predict,
    StratifiedShuffleSplit,
    StratifiedKFold,
    GridSearchCV
)

# ==============================
# Transformers & Pipelines
# ==============================
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.pipeline import Pipeline
```

```
# ==============================
# Machine Learning Models
# ==============================
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
# ==============================
# Hyperparameter Optimisation
# ==============================
from skopt import BayesSearchCV
from sklearn.model_selection import GridSearchCV
# ==============================
# Imbalanced Learning Techniques
# ==============================
from imblearn.over_sampling import SMOTE

# ==============================
# Model Persistence
# ==============================
from joblib import dump

# ==============================
# Create folder for saved models
# ==============================
if not os.path.exists("models"):
    os.makedirs("models")

# ==============================
# Suppress Warnings
# ==============================
warnings.filterwarnings(action='ignore')
```

0.3 Model Evaulation & Learning Curve Plot functions

[101]:

```
# Evaluates a trained model by printing classification metrics on the provided␣
↪test data
def evaluate_model(model, ytest, Xtest):

    yhat = model.predict(Xtest)
    print(classification_report(ytest, yhat, zero_division=0))


# Displays a formatted summary table of cross-validation results from any␣
↪search object
def print_cv_results(search_obj, col_width=100, max_rows=15):
```

```
    results = pd.DataFrame(search_obj.cv_results_)[
        ['params', 'mean_train_score', 'mean_test_score']
    ]
    results["diff, %"] = 100 * (
        results["mean_train_score"] - results["mean_test_score"]
    ) / results["mean_train_score"]

    # Display formatting
    pd.set_option('display.max_colwidth', col_width)
    pd.set_option('display.min_rows', max_rows)
    pd.set_option('display.max_rows', max_rows)

    display(results.sort_values('mean_test_score', ascending=False))
```

[102]:
```
def plot_learning_curves(model, X, y, scoring='f1_macro', cv=10, train_sizes=np.
↪linspace(0.1, 1.0, 5), save_path=None):
    """
    Plots learning curves with F1 Macro score for a classification model.

    Parameters:
    - model: fitted estimator or pipeline
    - X, y: training data
    - scoring: metric to evaluate (default = 'f1_macro')
    - cv: number of cross-validation folds
    - train_sizes: fractions of training set to use
    - save_path: if provided, saves the plot to this path
    """
    # Auto-generate model name for title
    try:
        model_name = type(model.named_steps[list(model.named_steps.
↪keys())[-1]]).__name__
    except AttributeError:
        model_name = type(model).__name__

    train_sizes, train_scores, val_scores = learning_curve(
        model, X, y,
        train_sizes=train_sizes,
        scoring=scoring,
        cv=cv,
        n_jobs=-1,
        shuffle=True,
        random_state=42
    )

    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
```

```
    val_mean = np.mean(val_scores, axis=1)
    val_std = np.std(val_scores, axis=1)

    plt.figure(figsize=(8, 5))
    plt.plot(train_sizes, train_mean, 'o-', label="Training score")
    plt.plot(train_sizes, val_mean, 'o-', label="Cross-validation score")

    plt.fill_between(train_sizes, train_mean - train_std, train_mean +␣
↪train_std, alpha=0.1)
    plt.fill_between(train_sizes, val_mean - val_std, val_mean + val_std,␣
↪alpha=0.1)

    plt.title(f"Learning Curve - {model_name}")
    plt.xlabel("Training Set Size")
    plt.ylabel("F1 Macro Score")
    plt.legend(loc="best")
    plt.grid(True)
    plt.tight_layout()

    if save_path:
        plt.savefig(save_path)
    plt.show()
```

0.4 Business Context & Objective

Transport for London's Vision Zero initiative sets an ambitious goal: to eliminate all deaths and serious injuries from London's transport network by 2041 (Transport for London, 2023). Central to this vision is the use of data-driven strategies to enhance road safety and influence driver behaviour.

This project directly supports that mission by developing machine learning models for the Westminster, Croydon, Wandsworth, Southwark, and Lambeth local authorities, aimed at predicting the severity of traffic accidents based on driver behaviour, vehicle characteristics, and environmental conditions. By identifying high-risk scenarios and patterns, the models seek to promote safer driving practices and enable these boroughs to implement proactive, targeted safety measures.

The models are trained on validated 2023 accident data, as the 2024 dataset has not yet undergone full verification. Ultimately, this initiative contributes to a safer, smarter urban transport environment, in full alignment with Vision Zero's long-term objectives.

0.5 Model Consideration

This project will build the following models:

- SVM
- Random Forest
- Logistic Regression
- GBM

Subsequently there performance will be compared against the baseline model and the top 2 best performing and practically suitable models will chosen to be implement as per the business use case mentioned above.

0.6 Loading preprocessed Train and Test Data

[6]:
```
X_train = pd.read_csv('X_train 1.csv', index_col=0)
y_train = pd.read_csv('y_train 1.csv', index_col=0)
X_test = pd.read_csv('X_test 1.csv', index_col=0)
y_test = pd.read_csv('y_test 1.csv', index_col=0)
```

[7]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
X_train.head()
```

```
(7808, 165)
(7808, 1)
(1947, 165)
(1947, 1)
```

[7]:
```
      accident_index  number_of_vehicles  number_of_casualties  speed_limit  \
2142    2.023010e+12            1.098612              0.693147     3.433987
1930    2.023010e+12            0.693147              0.693147     3.044522
8757    2.023010e+12            1.098612              0.693147     3.433987
9327    2.023010e+12            1.098612              0.693147     3.044522
7125    2.023010e+12            1.098612              0.693147     3.044522

      age_of_driver  local_authority_ons_district_Lambeth  \
2142       3.295837                                   1.0
1930       3.714609                                   1.0
8757       3.258097                                   1.0
9327       4.174387                                   1.0
7125       4.110874                                   0.0

      local_authority_ons_district_Southwark  \
2142                                     0.0
1930                                     0.0
8757                                     0.0
9327                                     0.0
7125                                     0.0

      local_authority_ons_district_Wandsworth  \
2142                                      0.0
1930                                      0.0
8757                                      0.0
9327                                      0.0
7125                                      1.0
```

```
     local_authority_ons_district_Westminster  road_type_One way street  …  \
2142                                      0.0                       0.0  …
1930                                      0.0                       0.0  …
8757                                      0.0                       0.0  …
9327                                      0.0                       0.0  …
7125                                      0.0                       0.0  …

     casualty_type_Motorcycle over 125cc and up to 500cc rider or  passenger  \
2142                                                0.0
1930                                                0.0
8757                                                0.0
9327                                                0.0
7125                                                0.0

     casualty_type_Motorcycle over 500cc rider or passenger  \
2142                                                0.0
1930                                                0.0
8757                                                0.0
9327                                                0.0
7125                                                0.0

     casualty_type_Other vehicle occupant  casualty_type_Pedestrian  \
2142                                  0.0                       0.0
1930                                  0.0                       0.0
8757                                  0.0                       0.0
9327                                  0.0                       1.0
7125                                  0.0                       0.0

     casualty_type_Taxi/Private hire car occupant  \
2142                                          0.0
1930                                          0.0
8757                                          0.0
9327                                          0.0
7125                                          0.0

     casualty_type_Tram occupant  \
2142                         0.0
1930                         0.0
8757                         0.0
9327                         0.0
7125                         0.0

     casualty_type_Van / Goods vehicle (3.5 tonnes mgw or under) occupant  \
2142                                                0.0
1930                                                0.0
8757                                                0.0
```

```
9327                                                    0.0
7125                                                    0.0

      month  hour  day_of_week
2142      2    19            1
1930      3    15            5
8757     11    17            5
9327     12    17            5
7125      9    18            2

[5 rows x 165 columns]
```

0.7 Further Scaling & Feature Engineering

[8]:

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
numerical_columns = ['number_of_vehicles', 'number_of_casualties',␣
 ↪'speed_limit', 'age_of_driver']

scaled_vals = scaler.fit_transform(X_train[numerical_columns])
# Put the scaled values back into the original dataframe
X_train[numerical_columns] = scaled_vals


# Encode cyclical features
X_train['hour_sin'] = np.sin(2 * np.pi * X_train['hour'] / 24)
X_train['hour_cos'] = np.cos(2 * np.pi * X_train['hour'] / 24)
X_train['day_sin'] = np.sin(2 * np.pi * (X_train['day_of_week'] - 1) / 7)
X_train['day_cos'] = np.cos(2 * np.pi * (X_train['day_of_week'] - 1) / 7)
X_train['month_sin'] = np.sin(2 * np.pi * (X_train['month'] - 1) / 12)
X_train['month_cos'] = np.cos(2 * np.pi * (X_train['month'] - 1) / 12)
X_train = X_train.drop(columns=['hour', 'day_of_week', 'month'])

# inspect the data
X_train.head()
```

[8]:

```
      accident_index  number_of_vehicles  number_of_casualties  speed_limit  \
2142    2.023010e+12            0.116021             -0.312529     1.315487
1930    2.023010e+12           -1.903348             -0.312529    -0.710615
8757    2.023010e+12            0.116021             -0.312529     1.315487
9327    2.023010e+12            0.116021             -0.312529    -0.710615
7125    2.023010e+12            0.116021             -0.312529    -0.710615

      age_of_driver  local_authority_ons_district_Lambeth  \
2142      -1.139575                                   1.0
1930       0.228371                                   1.0
```

```
8757     -1.262856                                1.0
9327      1.730264                                1.0
7125      1.522794                                0.0

      local_authority_ons_district_Southwark  \
2142                                     0.0
1930                                     0.0
8757                                     0.0
9327                                     0.0
7125                                     0.0

      local_authority_ons_district_Wandsworth  \
2142                                      0.0
1930                                      0.0
8757                                      0.0
9327                                      0.0
7125                                      1.0

      local_authority_ons_district_Westminster  road_type_One way street  …  \
2142                                       0.0                       0.0  …
1930                                       0.0                       0.0  …
8757                                       0.0                       0.0  …
9327                                       0.0                       0.0  …
7125                                       0.0                       0.0  …

      casualty_type_Pedestrian  casualty_type_Taxi/Private hire car occupant  \
2142                       0.0                                           0.0
1930                       0.0                                           0.0
8757                       0.0                                           0.0
9327                       1.0                                           0.0
7125                       0.0                                           0.0

      casualty_type_Tram occupant  \
2142                          0.0
1930                          0.0
8757                          0.0
9327                          0.0
7125                          0.0

      casualty_type_Van / Goods vehicle (3.5 tonnes mgw or under) occupant  \
2142                                                0.0
1930                                                0.0
8757                                                0.0
9327                                                0.0
7125                                                0.0

      hour_sin      hour_cos   day_sin   day_cos  month_sin  month_cos
```

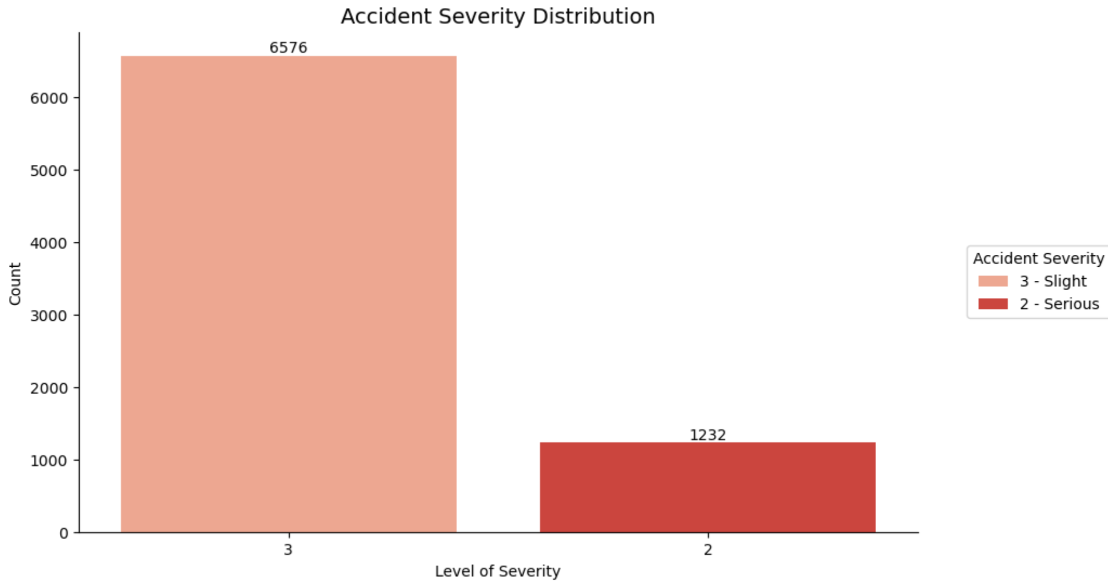
```
2142 -0.965926  2.588190e-01  0.000000  1.000000   0.500000   0.866025
1930 -0.707107 -7.071068e-01 -0.433884 -0.900969   0.866025   0.500000
8757 -0.965926 -2.588190e-01 -0.433884 -0.900969  -0.866025   0.500000
9327 -0.965926 -2.588190e-01 -0.433884 -0.900969  -0.500000   0.866025
7125 -1.000000 -1.836970e-16  0.781831  0.623490  -0.866025  -0.500000

[5 rows x 168 columns]
```

[9]:
```
from sklearn.preprocessing import StandardScaler

# Assume 'scaler' is already fitted on X_train
numerical_columns = ['number_of_vehicles', 'number_of_casualties',␣
↪'speed_limit', 'age_of_driver']

# Apply the same transformation to X_test
scaled_vals = scaler.transform(X_test[numerical_columns])
X_test[numerical_columns] = scaled_vals

# Encode cyclical features
X_test['hour_sin'] = np.sin(2 * np.pi * X_test['hour'] / 24)
X_test['hour_cos'] = np.cos(2 * np.pi * X_test['hour'] / 24)
X_test['day_sin'] = np.sin(2 * np.pi * (X_test['day_of_week'] - 1) / 7)
X_test['day_cos'] = np.cos(2 * np.pi * (X_test['day_of_week'] - 1) / 7)
X_test['month_sin'] = np.sin(2 * np.pi * (X_test['month'] - 1) / 12)
X_test['month_cos'] = np.cos(2 * np.pi * (X_test['month'] - 1) / 12)

# Drop original cyclical columns
X_test = X_test.drop(columns=['hour', 'day_of_week', 'month'])
```

0.8 Distribution of Target Variable

[10]:
```
#Create a bar chart for the target variable.
plt.figure(figsize=(10, 6))
ax = sns.countplot(x =y_train['accident_severity'],␣
↪order=y_train['accident_severity'].value_counts().index, palette='Reds')

#Add counts on bars
for container in ax.containers:
    ax.bar_label(container, fontsize=10)

#Label the plot and create a legend
plt.title('Accident Severity Distribution', fontsize=14)
plt.legend(title='Accident Severity', labels=['3 - Slight','2 - Serious'],␣
↪bbox_to_anchor=(1.05, 0.5), loc='center left')
plt.xlabel('Level of Severity')
plt.ylabel('Count')
plt.xticks(ha='center')
```

```
#Cleaner borders
sns.despine()
plt.show()
```



From the above graph it is evident there is a case of imbalanced data, that Accident Severity Level 3 (“Slight”) has significantly more instances (6,590) compared to Severity Level 2 (“Serious”) with only 1,218 occurrences. This can lead to Biased Model Predictions & Poor Performance on Minority Class, which will be addressed shortly.

[11]:
```
X_train.head()
```

[11]:
```
      accident_index  number_of_vehicles  number_of_casualties  speed_limit  \
2142    2.023010e+12            0.116021             -0.312529     1.315487
1930    2.023010e+12           -1.903348             -0.312529    -0.710615
8757    2.023010e+12            0.116021             -0.312529     1.315487
9327    2.023010e+12            0.116021             -0.312529    -0.710615
7125    2.023010e+12            0.116021             -0.312529    -0.710615

      age_of_driver  local_authority_ons_district_Lambeth  \
2142      -1.139575                                   1.0
1930       0.228371                                   1.0
8757      -1.262856                                   1.0
9327       1.730264                                   1.0
7125       1.522794                                   0.0

      local_authority_ons_district_Southwark  \
2142                                     0.0
```

```
1930                                    0.0
8757                                    0.0
9327                                    0.0
7125                                    0.0

      local_authority_ons_district_Wandsworth  \
2142                                      0.0
1930                                      0.0
8757                                      0.0
9327                                      0.0
7125                                      1.0

      local_authority_ons_district_Westminster  road_type_One way street  …  \
2142                                       0.0                       0.0  …
1930                                       0.0                       0.0  …
8757                                       0.0                       0.0  …
9327                                       0.0                       0.0  …
7125                                       0.0                       0.0  …

      casualty_type_Pedestrian  casualty_type_Taxi/Private hire car occupant  \
2142                       0.0                                           0.0
1930                       0.0                                           0.0
8757                       0.0                                           0.0
9327                       1.0                                           0.0
7125                       0.0                                           0.0

      casualty_type_Tram occupant  \
2142                          0.0
1930                          0.0
8757                          0.0
9327                          0.0
7125                          0.0

      casualty_type_Van / Goods vehicle (3.5 tonnes mgw or under) occupant  \
2142                                                0.0
1930                                                0.0
8757                                                0.0
9327                                                0.0
7125                                                0.0

      hour_sin      hour_cos   day_sin   day_cos  month_sin  month_cos
2142 -0.965926  2.588190e-01  0.000000  1.000000   0.500000   0.866025
1930 -0.707107 -7.071068e-01 -0.433884 -0.900969   0.866025   0.500000
8757 -0.965926 -2.588190e-01 -0.433884 -0.900969  -0.866025   0.500000
9327 -0.965926 -2.588190e-01 -0.433884 -0.900969  -0.500000   0.866025
7125 -1.000000 -1.836970e-16  0.781831  0.623490  -0.866025  -0.500000
```

```
[5 rows x 168 columns]
```

1 Model Building & Evaluation

1.1 Baseline

[15]:
```python
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train, y_train)
yhat_train = dummy_clf.predict(X_train)

evaluate_model(dummy_clf, y_train, X_train)
```

```
              precision    recall  f1-score   support

           2       0.00      0.00      0.00      1232
           3       0.84      1.00      0.91      6576

    accuracy                           0.84      7808
   macro avg       0.42      0.50      0.46      7808
weighted avg       0.71      0.84      0.77      7808
```

1.2 Random Forest (RF) Model

1.2.1 RF Model 1 – Adjusting Class Weights (using class_weight) to Handle Class Imbalance

[122]:
```python
start_rf = datetime.now()

param_grid = [
    {
        'n_estimators': [100, 200, 500],
        'max_depth': [5, 10, None],
        'class_weight': [None, 'balanced', {2: 3, 3: 1}]
    }
]

rf = RandomForestClassifier(random_state=7)

# Set up GridSearchCV
rf_grid_search = GridSearchCV(rf, param_grid,
                              cv=10,  # 10-fold cross-validation
                              scoring='f1_macro',
                              n_jobs=-1,  # Use all CPU cores
                              return_train_score=True)
```

```python
rf_grid_search.fit(X_train, y_train)
end_rf = datetime.now()
execution_time_rf = (end_rf - start_rf).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_rf)
```

```
Execution time (HH:MM:SS): 2.72670585
```

[110]:

```python
# Display cross-validation results for each parameter combination
print_cv_results(rf_grid_search, col_width=100)
```

```
                                                                   params  \
21  {'class_weight': {2: 3, 3: 1}, 'max_depth': 10, 'n_estimators': 100}
23  {'class_weight': {2: 3, 3: 1}, 'max_depth': 10, 'n_estimators': 500}
22  {'class_weight': {2: 3, 3: 1}, 'max_depth': 10, 'n_estimators': 200}
14    {'class_weight': 'balanced', 'max_depth': 10, 'n_estimators': 500}
12    {'class_weight': 'balanced', 'max_depth': 10, 'n_estimators': 100}
19   {'class_weight': {2: 3, 3: 1}, 'max_depth': 5, 'n_estimators': 200}
18   {'class_weight': {2: 3, 3: 1}, 'max_depth': 5, 'n_estimators': 100}
..                                                                  …
11     {'class_weight': 'balanced', 'max_depth': 5, 'n_estimators': 500}
3          {'class_weight': None, 'max_depth': 10, 'n_estimators': 100}
5          {'class_weight': None, 'max_depth': 10, 'n_estimators': 500}
4          {'class_weight': None, 'max_depth': 10, 'n_estimators': 200}
1           {'class_weight': None, 'max_depth': 5, 'n_estimators': 200}
2           {'class_weight': None, 'max_depth': 5, 'n_estimators': 500}
0           {'class_weight': None, 'max_depth': 5, 'n_estimators': 100}

    mean_train_score  mean_test_score    diff, %
21          0.796174         0.606708  23.797167
23          0.802678         0.604705  24.664119
22          0.799805         0.604108  24.468141
14          0.649449         0.592521   8.765553
12          0.647891         0.592262   8.586266
19          0.627311         0.590397   5.884429
18          0.625589         0.590067   5.678223
..               …                …          …
11          0.518476         0.516179   0.442910
3           0.487489         0.463023   5.018782
5           0.480901         0.462185   3.891908
4           0.483578         0.462185   4.423919
1           0.457175         0.457175   0.000007
2           0.457175         0.457175   0.000007
0           0.457175         0.457175   0.000007

[27 rows x 4 columns]
```

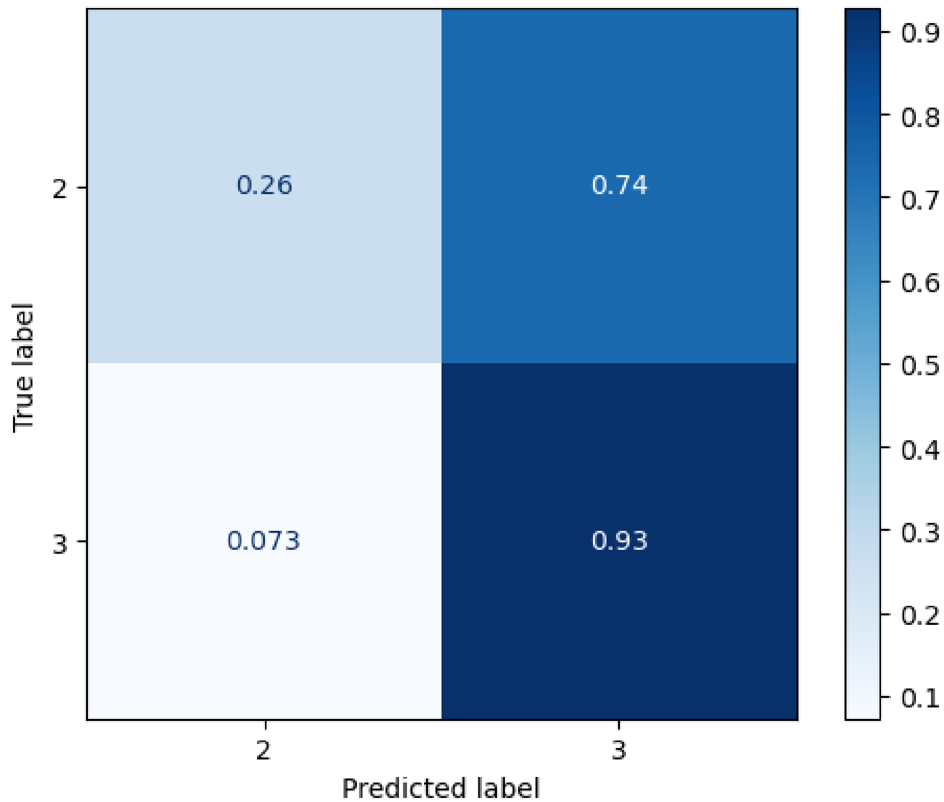
[111]:
```python
# Results
print("Best Parameters:", rf_grid_search.best_params_)
print("Best F1 Macro Score:", rf_grid_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(rf_grid_search.best_estimator_, X_train,␣
↪y_train, cv=10)

# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train, y_train_pred, zero_division=0))
```

```
Best Parameters: {'class_weight': {2: 3, 3: 1}, 'max_depth': 10, 'n_estimators':
100}
Best F1 Macro Score: 0.606707514265665

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           2       0.40      0.26      0.32      1232
           3       0.87      0.93      0.90      6576

    accuracy                           0.82      7808
   macro avg       0.64      0.59      0.61      7808
weighted avg       0.80      0.82      0.81      7808
```

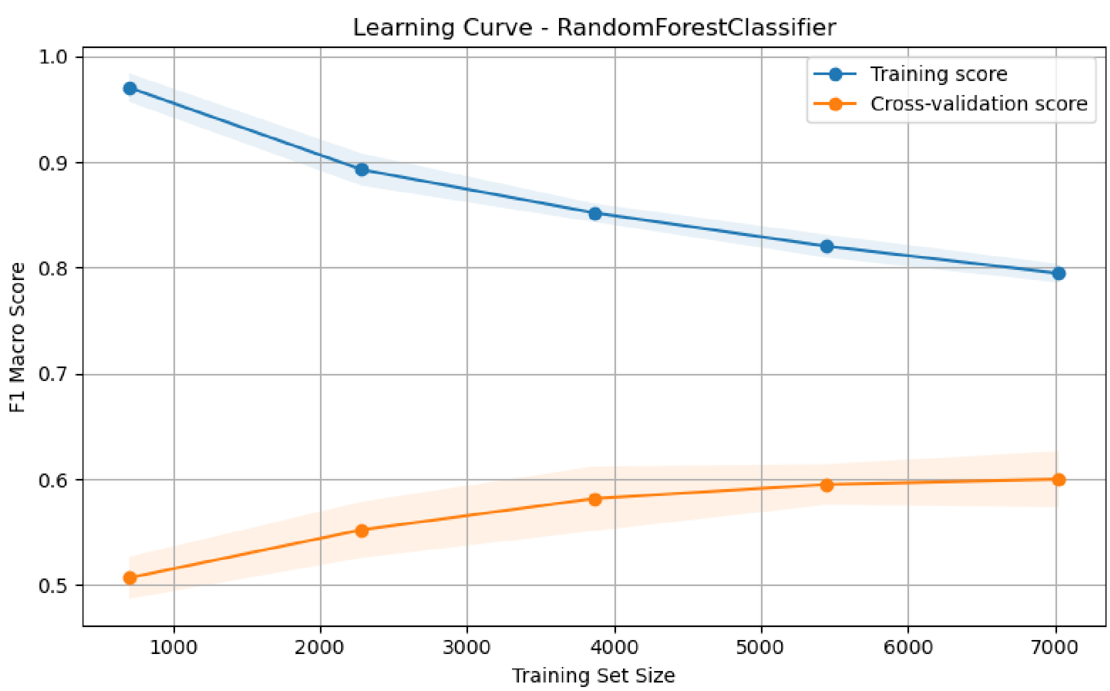
The best Random Forest model with the tuned hyperparameters above showed a clear improvement over the baseline. It achieved a macro F1-score of just over 0.60, compared to 0.46 for the baseline model.

[112]:
```python
# cross-validation confusion matrix, training data
yhat = cross_val_predict(rf_grid_search.best_estimator_, X_train, y_train,␣
↪cv=10)
ConfusionMatrixDisplay.from_predictions(y_train, yhat,
                                        labels=rf_grid_search.best_estimator_.
↪classes_,
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

[112]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a07b7df4a0>



[113]:
```
plot_learning_curves(rf_grid_search.best_estimator_, X_train, y_train)
```



1.2.2 RF Model 2 - Oversampling the Minority Class (using SMOTE) To Handle Class Imbalance

As Undersampling can result in risk of underfitting and lose of useful information that could contribute to model generalization, we will be Oversampling the Minority Class using SMOTE (Synthetic Minority Over-sampling Technique).

[115]:
```python
start_rf_smote = datetime.now()



# Define the pipeline
pipeline = Pipeline([
    ('smote', SMOTE(random_state=7)),
    ('rfc', RandomForestClassifier(random_state=7))
])

# Define the search space
param_dist = {
    'rfc__n_estimators': [100, 200, 500],
    'rfc__max_depth': [5, None],
    'smote__sampling_strategy': [0.5, 0.75, 1.0]
}

# Randomized search
rf_smote_random_search = RandomizedSearchCV(
    estimator=pipeline,
    param_distributions=param_dist,
    n_iter=6,                          # Number of random combinations to try
    cv=10,
    scoring='f1_macro',
    return_train_score=True,
    random_state=42,
    n_jobs=-1
)

# Fit the model
rf_smote_random_search.fit(X_train, y_train)
end_rf_smote = datetime.now()
execution_time_rf_smote = (end_rf_smote - start_rf).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_rf_smote)
```

```
Execution time (HH:MM:SS): 10.211479066666667
```

[34]:
```
# Display cross-validation results for each parameter combination
print_cv_results(rf_smote_random_search, col_width=100)
```

```
↪params  \
2      {'smote__sampling_strategy': 1.0, 'rfc__n_estimators': 500,
↪'rfc__max_depth': 5}
3      {'smote__sampling_strategy': 1.0, 'rfc__n_estimators': 200,
↪'rfc__max_depth': 5}
1     {'smote__sampling_strategy': 0.75, 'rfc__n_estimators': 100,
↪'rfc__max_depth': 5}
5  {'smote__sampling_strategy': 0.75, 'rfc__n_estimators': 200, 'rfc__max_depth':
↪ None}
0      {'smote__sampling_strategy': 0.5, 'rfc__n_estimators': 100,
↪'rfc__max_depth': 5}
4      {'smote__sampling_strategy': 0.5, 'rfc__n_estimators': 200,
↪'rfc__max_depth': 5}

   mean_train_score  mean_test_score    diff, %
2          0.610542         0.594310   2.658551
3          0.610469         0.592939   2.871518
1          0.585947         0.568636   2.954459
5          1.000000         0.543291  45.670873
0          0.499053         0.494496   0.913133
4          0.500848         0.490209   2.124149
```

[37]:
```
# Results
print("Best Parameters:", rf_smote_random_search.best_params_)
print("Best F1 Macro Score:", rf_smote_random_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(rf_smote_random_search.best_estimator_,
↪X_train, y_train, cv=10)

# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train, y_train_pred, zero_division=0))
```

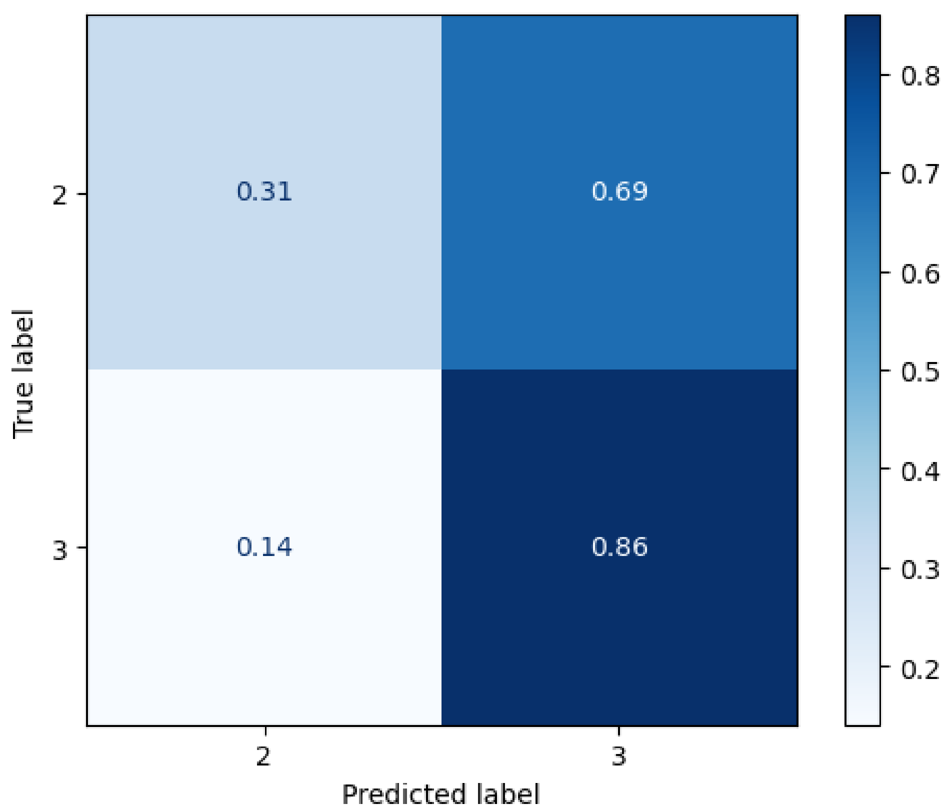
```
Best Parameters: {'smote__sampling_strategy': 1.0, 'rfc__n_estimators': 500,
'rfc__max_depth': 5}
Best F1 Macro Score: 0.5943099996659227

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           2       0.29      0.31      0.30      1232
           3       0.87      0.86      0.86      6576
```

```
    accuracy                           0.77      7808
   macro avg       0.58      0.58      0.58      7808
weighted avg       0.78      0.77      0.78      7808
```

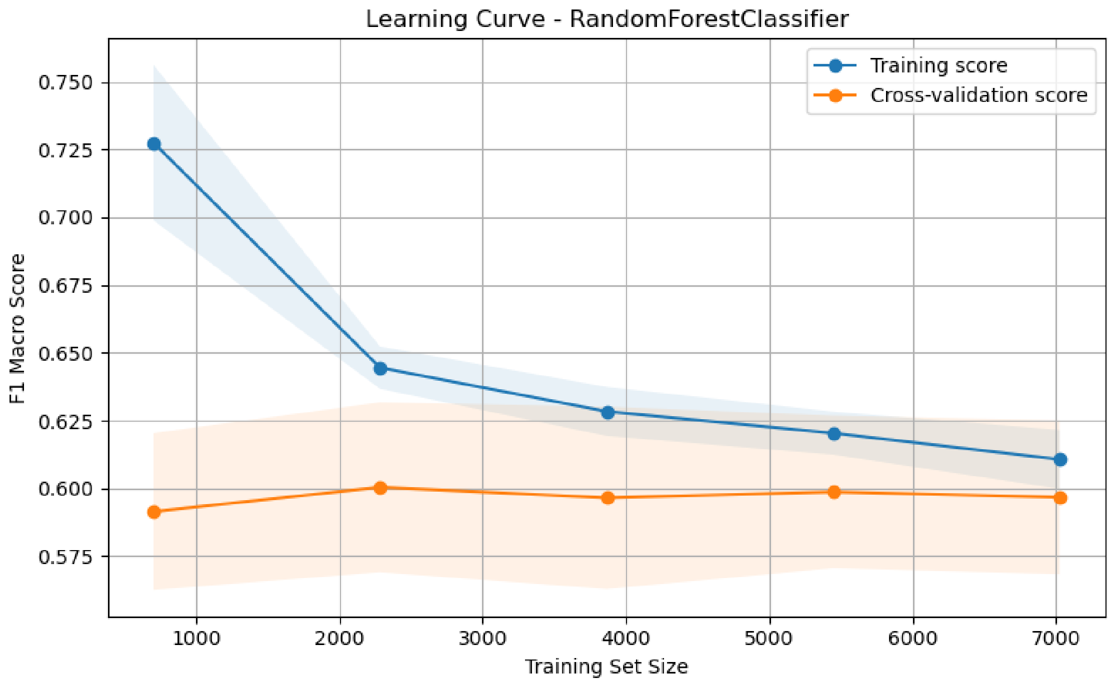
The best model identified through RandomizedSearchCV uses SMOTE with a sampling_strategy of 1.0. This means that during training, the minority class (label 2) was oversampled to have the same number of instances as the majority class (label 3).

[40]:
```
# cross-validation confusion matrix, training data
yhat = cross_val_predict(rf_smote_random_search.best_estimator_, X_train,␣
↪y_train, cv=10)
ConfusionMatrixDisplay.from_predictions(y_train, yhat,
                                        labels=rf_smote_random_search.
↪best_estimator_.classes_,
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

[40]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a0735d56d0>



[41]:
```
plot_learning_curves(rf_smote_random_search.best_estimator_, X_train, y_train)
```



[42]:
```
# Get metrics for both models
model1_metrics = get_model_metrics(rf_grid_search.best_estimator_, X_train,␣
↪y_train)
model2_metrics = get_model_metrics(rf_smote_random_search.best_estimator_,␣
↪X_train, y_train)

# Create DataFrame containing results from both RF models
df_compare = pd.DataFrame({
    "Random Forest without SMOTE": model1_metrics,
    "Random Forest with SMOTE": model2_metrics
})

# Display results
df_compare
```

[42]:

	Random Forest without SMOTE	Random Forest with SMOTE
F1 Macro	0.61	0.58
Class 2 F1	0.32	0.30
Accuracy	0.82	0.77

- Random Forest (RF) without SMOTE outperformed the SMOTE version on most metrics: Accuracy: 0.82 vs 0.77 & F1 Macro: 0.61 vs 0.59
- Class 2 F1-score was slightly better with SMOTE: 0.33 vs 0.31 (without SMOTE)
- SMOTE provided a small gain for minority class, but reduced overall accuracy.

Overall, The RF model with class weight handled class imbalance sufficiently well without oversampling, offering a better overall balance.

1.2.3 Learning Curve results:

RF with SMOTE: - Shows a smaller gap between training and validation scores, indicating less overfitting.

- Validation performance is slightly more stable and improves modestly with more data.

RF with class weight: - Maintains a higher training score, but with a larger train-validation gap, suggesting more overfitting.

- Validation score improves slightly but plateaus earlier, showing limited generalisation.

While RF with SMOTE generalised slightly better, RF with class_weight was chosen due to its due to its higher overall accuracy and macro F1-score.

[ ]:
```
# Save the best estimator (final trained model)
dump(rf_grid_search.best_estimator_, 'models/Random_Forest.joblib')
```

1.3 Decision Tree

1.3.1 DT Model 1 - with Class Weight

[116]:
```
start_dt = datetime.now()

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=7)

hp_grid = {
    'max_depth': [5, 10, 15, 20, 25, 30, 35, 40],
    'min_samples_split': [5, 10, 15, 20, 25, 30, 35],
    'class_weight': [{2: 3, 3: 1},{2: 2, 3: 1}, 'balanced', None],  # Using␣
 ↪class weights
}

dt_grid_search = GridSearchCV(dt, hp_grid, cv=10,
                              scoring='f1_macro',
                              return_train_score=True)

dt_grid_search.fit(X_train, y_train)

end_dt = datetime.now()
execution_time_dt = (end_dt - start_dt).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_dt)
```

```
Execution time (HH:MM:SS): 4.028156433333334
```

[45]:
```python
# Display cross-validation results for each parameter combination
print_cv_results(dt_grid_search, col_width=100)
```

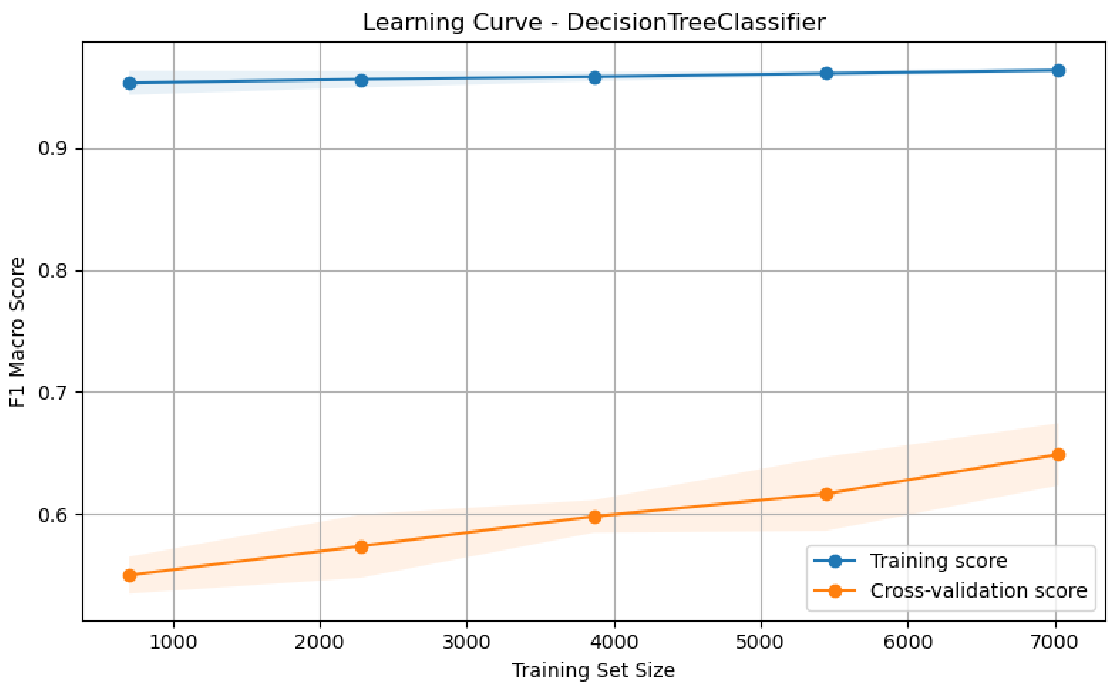
```
                                                                                  params  \
98   {'class_weight': {2: 2, 3: 1}, 'max_depth': 35, 'min_samples_split': 5}
91   {'class_weight': {2: 2, 3: 1}, 'max_depth': 30, 'min_samples_split': 5}
105  {'class_weight': {2: 2, 3: 1}, 'max_depth': 40, 'min_samples_split': 5}
84   {'class_weight': {2: 2, 3: 1}, 'max_depth': 25, 'min_samples_split': 5}
49   {'class_weight': {2: 3, 3: 1}, 'max_depth': 40, 'min_samples_split': 5}
28   {'class_weight': {2: 3, 3: 1}, 'max_depth': 25, 'min_samples_split': 5}
42   {'class_weight': {2: 3, 3: 1}, 'max_depth': 35, 'min_samples_split': 5}
..                                                                         …
172          {'class_weight': None, 'max_depth': 5, 'min_samples_split': 25}
168           {'class_weight': None, 'max_depth': 5, 'min_samples_split': 5}
169          {'class_weight': None, 'max_depth': 5, 'min_samples_split': 10}
171          {'class_weight': None, 'max_depth': 5, 'min_samples_split': 20}
170          {'class_weight': None, 'max_depth': 5, 'min_samples_split': 15}
173          {'class_weight': None, 'max_depth': 5, 'min_samples_split': 30}
174          {'class_weight': None, 'max_depth': 5, 'min_samples_split': 35}

     mean_train_score  mean_test_score    diff, %
98           0.963524         0.650433  32.494389
91           0.962295         0.650172  32.435271
105          0.963877         0.650120  32.551506
84           0.956255         0.650074  32.018805
49           0.959103         0.644398  32.812463
28           0.940100         0.643865  31.510962
42           0.958748         0.643860  32.843620
..                …              …          …
172          0.513336         0.488099   4.916238
168          0.516268         0.487549   5.562760
169          0.515565         0.487458   5.451693
171          0.514243         0.487415   5.217025
170          0.514579         0.486632   5.431087
173          0.511043         0.485052   5.085834
174          0.509443         0.485052   4.787798

[224 rows x 4 columns]
```

[46]:
```python
# Results
print("Best Parameters:",dt_grid_search.best_params_)
print("Best F1 Macro Score:", dt_grid_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(dt_grid_search.best_estimator_, X_train,␣
 ↪y_train, cv=10)
```

```python
# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train, y_train_pred, zero_division=0))
```

```
Best Parameters: {'class_weight': {2: 2, 3: 1}, 'max_depth': 35,
'min_samples_split': 5}
Best F1 Macro Score: 0.6504328050310609

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           2       0.40      0.44      0.42      1232
           3       0.89      0.87      0.88      6576

    accuracy                           0.81      7808
   macro avg       0.65      0.66      0.65      7808
weighted avg       0.81      0.81      0.81      7808
```

[47]:
```python
plot_learning_curves(dt_grid_search.best_estimator_, X_train, y_train)
```



Model shows signs of overfitting, though the training score is slightly lower and the cross-validation score improves steadily with more data.

1.3.2 DT Model 2 - with SMOTE

[123]:
```
# Build a pipeline that applies SMOTE for class balancing and fits a Decision␣
↪Tree model
start_dt_smote = datetime.now()
pipeline = Pipeline([
    ('smote', SMOTE(random_state=7)),
    ('model', DecisionTreeClassifier(random_state=7))
])

#Set hyperparameters
hp_grid = {
    'smote__sampling_strategy': [0.5, 0.75, 1.0],
    'model__max_depth': (1, 50),
    'model__min_samples_split': (2, 100),
    'model__min_impurity_decrease': (0.0, 0.1)
}

#Initialize Bayesian search
dt_smote_bayes_search = BayesSearchCV(pipeline,
                                      hp_grid,
                                      n_iter=50,
                                      random_state=7,
                                      scoring='f1_macro',
                                      return_train_score=True,
                                      cv=10,
                                      n_jobs=-1)

#Fit the model
dt_smote_bayes_search.fit(X_train, y_train)

end_dt_smote = datetime.now()
execution_time_dt_smote = (end_dt_smote - start_dt_smote).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_dt_smote)
```

```
Execution time (HH:MM:SS): 1.28635035
```

[49]:
```
# Display cross-validation results for each parameter combination
print_cv_results(dt_smote_bayes_search, col_width=100)
```

```
                                                                              ␣
↪                  params  \
49  {'model__max_depth': 50, 'model__min_impurity_decrease': 0.0,␣
↪'model__min_samples_split': 2, 'sm…
46  {'model__max_depth': 49, 'model__min_impurity_decrease': 0.
↪00012549424663307331, 'model__min_sam…
42  {'model__max_depth': 27, 'model__min_impurity_decrease': 0.0,␣
↪'model__min_samples_split': 2, 'sm…
```

```
43  {'model__max_depth': 26, 'model__min_impurity_decrease': 0.0,␣
 ↪'model__min_samples_split': 2, 'sm…
40  {'model__max_depth': 28, 'model__min_impurity_decrease': 0.0,␣
 ↪'model__min_samples_split': 2, 'sm…
33  {'model__max_depth': 50, 'model__min_impurity_decrease': 0.0,␣
 ↪'model__min_samples_split': 2, 'sm…
47  {'model__max_depth': 29, 'model__min_impurity_decrease': 0.0,␣
 ↪'model__min_samples_split': 2, 'sm…
..                                                                              ␣
 ↪                   …
30  {'model__max_depth': 31, 'model__min_impurity_decrease': 0.
 ↪011033854977239224, 'model__min_sampl…
22  {'model__max_depth': 1, 'model__min_impurity_decrease': 0.
 ↪050595008993641534, 'model__min_sample…
18  {'model__max_depth': 1, 'model__min_impurity_decrease': 0.1,␣
 ↪'model__min_samples_split': 98, 'sm…
17  {'model__max_depth': 1, 'model__min_impurity_decrease': 0.0,␣
 ↪'model__min_samples_split': 95, 'sm…
8   {'model__max_depth': 43, 'model__min_impurity_decrease': 0.
 ↪06216711201133072, 'model__min_sample…
5   {'model__max_depth': 50, 'model__min_impurity_decrease': 0.
 ↪08795381484152082, 'model__min_sample…
0   {'model__max_depth': 49, 'model__min_impurity_decrease': 0.
 ↪08586193860926859, 'model__min_sample…

    mean_train_score  mean_test_score    diff, %
49          1.000000         0.633265  36.673495
46          0.873097         0.624476  28.475747
42          0.995109         0.620538  37.641211
43          0.991559         0.616725  37.802527
40          0.996197         0.616616  38.102979
33          1.000000         0.616523  38.347674
47          0.997601         0.615796  38.272287
..               …                …          …
30          0.462302         0.462671  -0.079743
22          0.457175         0.457175   0.000007
18          0.457175         0.457175   0.000007
17          0.457175         0.457175   0.000007
8           0.457175         0.457175   0.000007
5           0.457175         0.457175   0.000007
0           0.457175         0.457175   0.000007

[50 rows x 4 columns]
```

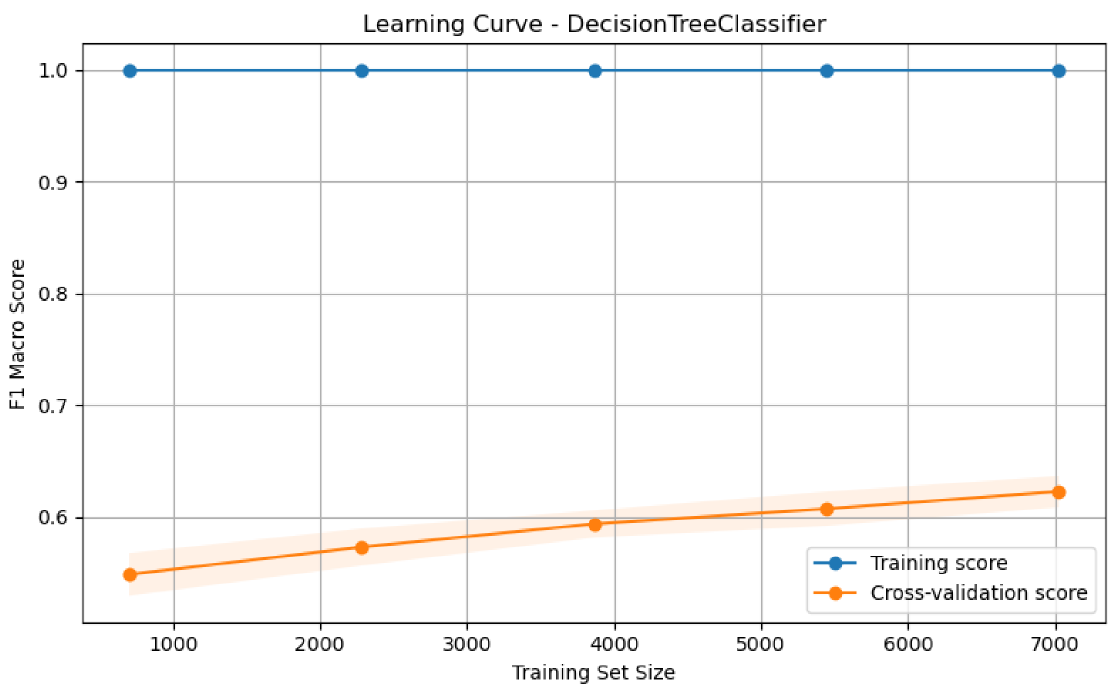
[50]:
```
# Results
print("Best Parameters:", dt_smote_bayes_search.best_params_)
```

```
print("Best F1 Macro Score:", dt_smote_bayes_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(dt_smote_bayes_search.best_estimator_,␣
↪X_train, y_train, cv=10)

# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train, y_train_pred, zero_division=0))
```

```
Best Parameters: OrderedDict({'model__max_depth': 50,
'model__min_impurity_decrease': 0.0, 'model__min_samples_split': 2,
'smote__sampling_strategy': 0.5})
Best F1 Macro Score: 0.6332650525955346

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           2       0.37      0.39      0.38      1232
           3       0.88      0.88      0.88      6576

    accuracy                           0.80      7808
   macro avg       0.63      0.63      0.63      7808
weighted avg       0.80      0.80      0.80      7808
```

[51]:

```
plot_learning_curves(dt_smote_bayes_search.best_estimator_, X_train, y_train)
```



This model is overfitting, as it achieves perfect training scores while cross-validation scores remain significantly lower and gradually improve with more data.

[52]:
```
# Get metrics for both models
model1_metrics = get_model_metrics(dt_grid_search.best_estimator_, X_train,␣
↪y_train)
model2_metrics = get_model_metrics(dt_smote_bayes_search.best_estimator_,␣
↪X_train, y_train)

# Create DataFrame
df_compare = pd.DataFrame({
    "Decision Tree with Class weight": model1_metrics,
    "Decision Tree with SMOTE": model2_metrics
})

# Display transposed table (metrics as rows)
df_compare
```

[52]:

	Decision Tree with Class weight	Decision Tree with SMOTE
F1 Macro	0.65	0.63
Class 2 F1	0.42	0.38
Accuracy	0.81	0.80

- The Decision Tree (DT) model with class_weight='balanced' outperformed the SMOTE-based model across all key metrics.
- It achieved a higher overall accuracy (0.81 vs 0.80) and a notably stronger F1-score for the minority class, Class 2 (0.42 vs 0.38).
- While the SMOTE-based model did improve Class 2 performance significantly compared to the baseline (F1-score from 0.00 to 0.38), it still underperformed relative to the class_weight approach.
- DT with class weighting, effectively handled the class imbalance without the need for oversampling.
- Overall, the DT model with class_weight='balanced' provided the best balance between accuracy and minority class sensitivity.

[55]:
```
# Save the best estimator (final trained model) from RandomizedSearchCV
dump(dt_grid_search.best_estimator_, 'models/Decision_Tree.joblib')
```

[55]: ['models/Decision_Tree.joblib']

1.4 Logistics Regression (LR)

1.4.1 LR Model 1 - with Class Weight

[124]:
```
start_lr = datetime.now()

#Create pipeline for logistic regression and apply class weights to combat␣
↪class imbalance
pipeline = Pipeline([
    ('model', LogisticRegression(
        random_state=7,
        max_iter=1000,  #Ensure convergence
        solver='liblinear',
        class_weight={2: 3, 3: 1}
    ))
])

#Set hyperparameters
param_grid = {
    'model__penalty': ['l1', 'l2']
}

#Initialize logistic regression
logreg_search = RandomizedSearchCV(pipeline,
                                   param_grid,
                                   n_iter=50,
                                   cv=10,
                                   scoring='f1_macro',
                                   random_state=7,
                                   n_jobs=-1,
                                   return_train_score=True
)

#Fit the model
logreg_search.fit(X_train, y_train)

end_lr = datetime.now()
execution_time_lr = (end_lr - start_lr).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_lr)
```

```
Execution time (HH:MM:SS): 1.2164680666666667
```

[ ]:
```
print("Start time:", start_lr)
print("End time:", end_lr)
print("Total time (mins):", round(execution_time_lr, 2))
```

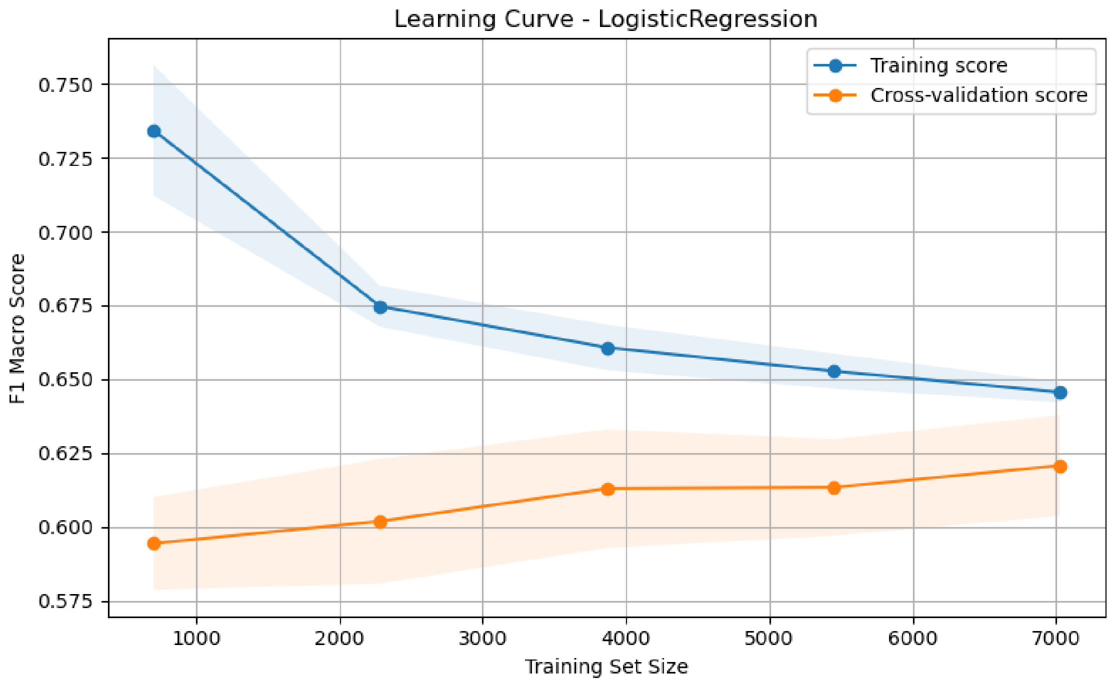
[57]:
```
# Display cross-validation results for each parameter combination
print_cv_results(logreg_search, col_width=100)
```

```
                       params  mean_train_score  mean_test_score   diff, %
0  {'model__penalty': 'l1'}          0.645633         0.620808  3.845010
1  {'model__penalty': 'l2'}          0.457175         0.457175  0.000007
```

[58]:
```
# Results
print("Best Parameters:", logreg_search.best_params_)
print("Best F1 Macro Score:", logreg_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(logreg_search.best_estimator_, X_train,
 ↪y_train, cv=10)

# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train, y_train_pred, zero_division=0))
```

```
Best Parameters: {'model__penalty': 'l1'}
Best F1 Macro Score: 0.6208084982652516

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           2       0.33      0.48      0.39      1232
           3       0.89      0.82      0.85      6576

    accuracy                           0.76      7808
   macro avg       0.61      0.65      0.62      7808
weighted avg       0.80      0.76      0.78      7808
```

[59]:
```
plot_learning_curves(logreg_search.best_estimator_, X_train, y_train)
```



This model with L1 regularisation displays some overfitting, but the consistent rise in cross-validation performance indicates improving generalisation as training data increases.

1.4.2 Model 2 LR - with SMOTE

[128]:
```
start_lr_smote = datetime.now()

# Define pipeline
pipeline = Pipeline([
    ('smote', SMOTE(random_state=42)),
    ('logreg', LogisticRegression(max_iter=1000, random_state=42))
])

# Define hyperparameter grid
param_grid = {
    'logreg__C': [1, 1000],             # Removed very small C (0.01)
    'logreg__solver': ['liblinear', 'lbfgs']
}

# Stratified cross-validation
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Grid search with pipeline
logreg_smote_search = GridSearchCV(pipeline, param_grid=param_grid,␣
↪scoring='f1_macro', return_train_score=True, cv=cv)
```

```python
logreg_smote_search.fit(X_train, y_train)

end_lr_smote = datetime.now()
execution_time_lr_smote = (end_lr_smote - start_lr_smote).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_lr_smote)
```

```
Execution time (HH:MM:SS): 0.08690131666666666
```

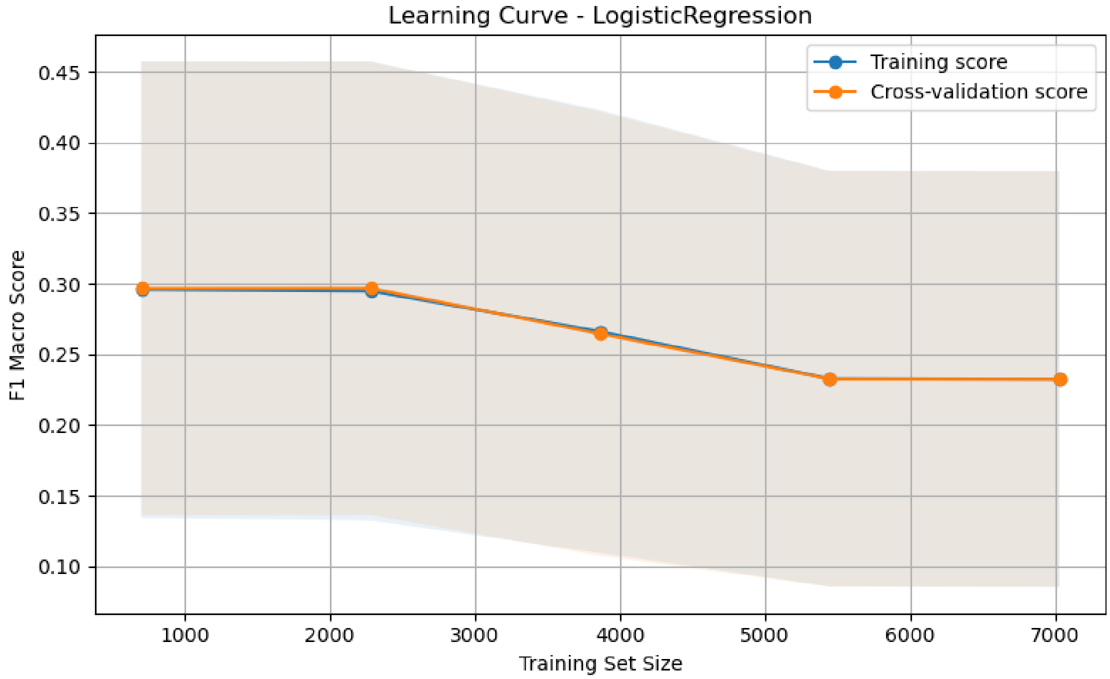
[61]:
```python
# Display cross-validation results for each parameter combination
print_cv_results(logreg_smote_search, col_width=100)
```

```
                                                   params  mean_train_score  \
1          {'logreg__C': 1, 'logreg__solver': 'lbfgs'}          0.425074
3       {'logreg__C': 1000, 'logreg__solver': 'lbfgs'}          0.425074
0      {'logreg__C': 1, 'logreg__solver': 'liblinear'}          0.328802
2   {'logreg__C': 1000, 'logreg__solver': 'liblinear'}          0.264626

   mean_test_score   diff, %
1         0.425188 -0.026747
3         0.425188 -0.026747
0         0.328961 -0.048311
2         0.264763 -0.051762
```

[62]:
```python
# Results
print("Best Parameters:", logreg_smote_search.best_params_)
print("Best F1 Macro Score:", logreg_smote_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(logreg_smote_search.best_estimator_, X_train,␣
 ↪y_train, cv=10)

# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train, y_train_pred, zero_division=0))
```

```
Best Parameters: {'logreg__C': 1, 'logreg__solver': 'lbfgs'}
Best F1 Macro Score: 0.4251878071863445

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           2       0.16      0.10      0.12      1232
           3       0.84      0.90      0.87      6576

    accuracy                           0.77      7808
   macro avg       0.50      0.50      0.50      7808
weighted avg       0.73      0.77      0.75      7808
```

[63]:
```
plot_learning_curves(logreg_smote_search.best_estimator_, X_train, y_train)
```



This model exhibits underfitting, as both training and cross-validation scores remain low and decline with more data, indicating limited learning capacity and poor generalisation

[64]:
```
# Get metrics for both models
model1_metrics = get_model_metrics(logreg_search.best_estimator_, X_train,␣
 ↪y_train)
model2_metrics = get_model_metrics(logreg_smote_search.best_estimator_,␣
 ↪X_train, y_train)

# Create DataFrame
df_compare = pd.DataFrame({
    "Logistic Regression with Class weight": model1_metrics,
    "Logistic Regression with SMOTE": model2_metrics
})

# Display transposed table (metrics as rows)
df_compare
```

[64]:
```
            Logistic Regression with Class weight  \
F1 Macro                                     0.62
Class 2 F1                                   0.39
Accuracy                                     0.76
```

```
              Logistic Regression with SMOTE
F1 Macro                                0.50
Class 2 F1                              0.12
Accuracy                                0.77
```

- Logistic Regression with class_weight='balanced' outperforms SMOTE across all key metrics apart from accuracy.
- However the model with SMOTE performs worse on the minority class (Class 2 F1 = 0.12) compared to the model with class weight='balanced'.

[65]:
```
# Save the best estimator (final trained model) from RandomizedSearchCV
dump(logreg_search.best_estimator_, 'models/logistic_Regression.joblib')
```

[65]:
```
['models/logistic_Regression.joblib']
```

1.5 SVM

[120]:
```
start_svm = datetime.now()

# Define the SVM model
svm = SVC(random_state=7, kernel='rbf')

# Hyperparameter grid
hp_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'class_weight': [
        {2: 2, 3: 1}
    ]
}



# Perform grid search with 10-fold cross-validation
svm_grid_search = GridSearchCV(
    svm,
    hp_grid,
    cv=10,
    scoring='f1_macro',
    return_train_score=True
)

# Fit the model
svm_grid_search.fit(X_train, y_train)

end_svm = datetime.now()
execution_time_svm = (end_svm - start_svm).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_svm)
```

```
Execution time (HH:MM:SS): 5.234349616666666
```

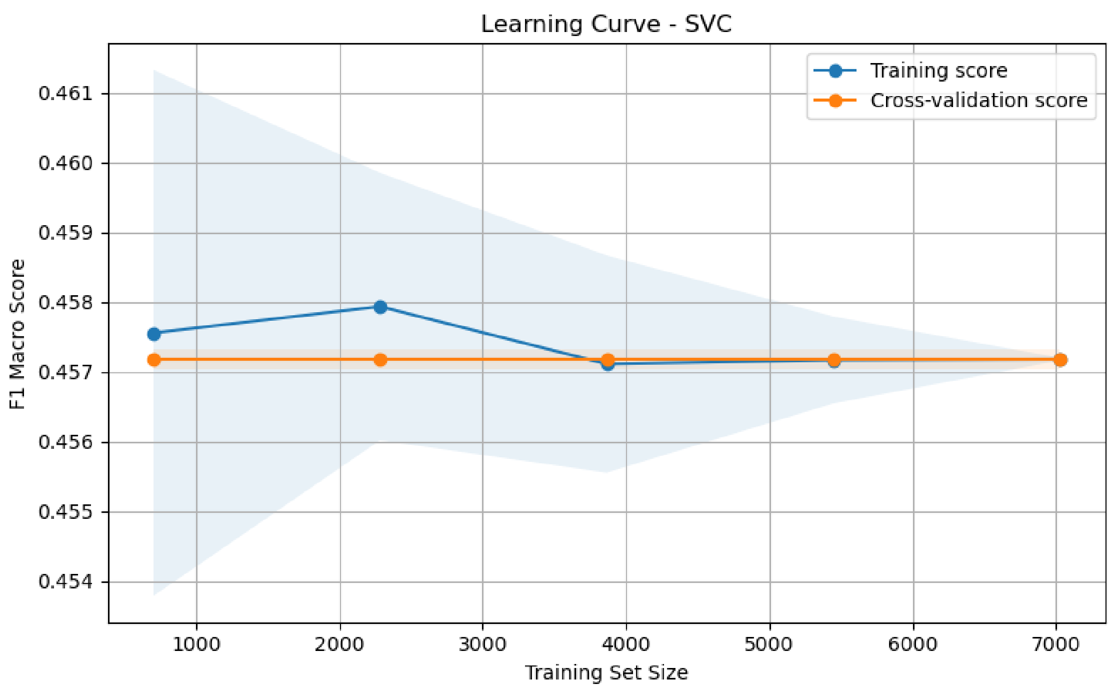
[67]:
```python
# Display cross-validation results for each parameter combination
print_cv_results(svm_grid_search, col_width=100)
```

```
                                         params  mean_train_score  \
0  {'C': 0.01, 'class_weight': {2: 2, 3: 1}}          0.457175
1   {'C': 0.1, 'class_weight': {2: 2, 3: 1}}          0.457175
2     {'C': 1, 'class_weight': {2: 2, 3: 1}}          0.457175
3    {'C': 10, 'class_weight': {2: 2, 3: 1}}          0.457175
4   {'C': 100, 'class_weight': {2: 2, 3: 1}}          0.457175

   mean_test_score   diff, %
0         0.457175  0.000007
1         0.457175  0.000007
2         0.457175  0.000007
3         0.457175  0.000007
4         0.457175  0.000007
```

[68]:
```python
# Results
print("Best Parameters:", svm_grid_search.best_params_)
print("Best F1 Macro Score:", svm_grid_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(svm_grid_search.best_estimator_, X_train,␣
 ↪y_train, cv=10)

# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train, y_train_pred, zero_division=0))
```

```
Best Parameters: {'C': 0.01, 'class_weight': {2: 2, 3: 1}}
Best F1 Macro Score: 0.45717460591693254

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           2       0.00      0.00      0.00      1232
           3       0.84      1.00      0.91      6576

    accuracy                           0.84      7808
   macro avg       0.42      0.50      0.46      7808
weighted avg       0.71      0.84      0.77      7808
```

[69]:
```python
plot_learning_curves(svm_grid_search.best_estimator_, X_train, y_train)
```



- The model fails completely to detect Class 2, assigning nearly all predictions to Class 3 despite using class_weight to try and address imbalance.
- The F1-score for Class 2 is 0.00, indicating zero precision and recall.
- All tested values of C (0.01 to 100) produced identical scores, implying the model is insensitive to regularisation changes.

The learning curve shows: - Very flat performance with increasing data. - No overfitting (train and CV scores are nearly identical), but also no learning — suggesting the model is underfitting.

[70]:
```
# Save the best estimator (final trained model) from GridSearchCV
dump(svm_grid_search.best_estimator_, 'models/svm.joblib')
```

[70]: ['models/svm.joblib']

1.6 XGBoost

[71]:
```
# Encode target labels as consecutive integers starting from 0 (e.g., 0, 1) to
↪ensure correct class indexing in XGBoost
le = LabelEncoder()

# Apply label encoding to training and test target data
y_train_encoded = le.fit_transform(y_train)
y_test_encoded = le.transform(y_test)
```

[126]:
```python
start_xgb = datetime.now()

# Define a pipeline with XGBoost classifier
pipeline = Pipeline([
    ('xgb', XGBClassifier(
        random_state=7,
        n_jobs=-1,
        eval_metric='logloss',
        use_label_encoder=False
    ))
])

# Set up the hyperparameter search space
param_grid = {
    'xgb__n_estimators': [100, 200, 300],
    'xgb__max_depth': [3, 5, 7],
    'xgb__learning_rate': [0.01, 0.05, 0.1],
    'xgb__subsample': [0.8, 0.9, 1.0],
    'xgb__colsample_bytree': [0.7, 0.9, 1.0],
    'xgb__gamma': [0, 1, 5],
    'xgb__reg_lambda': [0.1, 1, 10],  # L2 regularisation
    'xgb__scale_pos_weight': [1, 5, 10]  # Handle class imbalance
}

# Initialise the randomised hyperparameter search with cross-validation
xgb_search = RandomizedSearchCV(
    pipeline,
    param_grid,
    n_iter=50,
    cv=10,
    scoring='f1_macro',
    random_state=7,
    return_train_score=True,
    n_jobs=-1
)

# Fit the model to the training data
xgb_search.fit(X_train, y_train_encoded)

end_xgb = datetime.now()
execution_time_xgb = (end_xgb - start_xgb).total_seconds() / 60

print("Execution time (HH:MM:SS):", execution_time_xgb)
```

```
Execution time (HH:MM:SS): 1.6531634833333333
```

[73]:
```python
# Display cross-validation results for each parameter combination
print_cv_results(xgb_search, col_width=100)
```

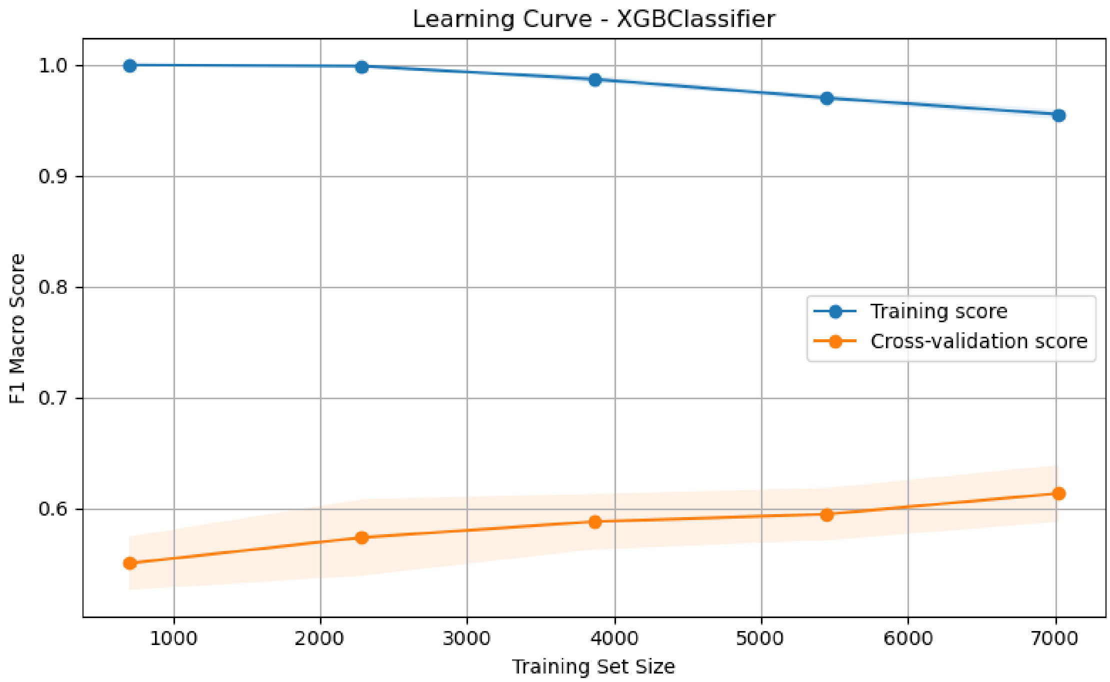
```
                                                                                ␣
 ↪                  params  \
23  {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 1, 'xgb__reg_lambda': 1,␣
 ↪'xgb__n_estimators': 2…
40  {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 1, 'xgb__reg_lambda': 0.1,␣
 ↪'xgb__n_estimators':…
42  {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 1, 'xgb__reg_lambda': 10,␣
 ↪'xgb__n_estimators': …
10  {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 1, 'xgb__reg_lambda': 10,␣
 ↪'xgb__n_estimators': …
15  {'xgb__subsample': 1.0, 'xgb__scale_pos_weight': 5, 'xgb__reg_lambda': 0.1,␣
 ↪'xgb__n_estimators':…
4   {'xgb__subsample': 1.0, 'xgb__scale_pos_weight': 1, 'xgb__reg_lambda': 0.1,␣
 ↪'xgb__n_estimators':…
7   {'xgb__subsample': 1.0, 'xgb__scale_pos_weight': 1, 'xgb__reg_lambda': 1,␣
 ↪'xgb__n_estimators': 3…
..                                                                              ␣
 ↪                     …
12  {'xgb__subsample': 0.9, 'xgb__scale_pos_weight': 5, 'xgb__reg_lambda': 1,␣
 ↪'xgb__n_estimators': 3…
32  {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 1, 'xgb__reg_lambda': 0.1,␣
 ↪'xgb__n_estimators':…
39  {'xgb__subsample': 0.9, 'xgb__scale_pos_weight': 10, 'xgb__reg_lambda': 1,␣
 ↪'xgb__n_estimators': …
27  {'xgb__subsample': 0.9, 'xgb__scale_pos_weight': 5, 'xgb__reg_lambda': 10,␣
 ↪'xgb__n_estimators': …
35  {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 5, 'xgb__reg_lambda': 1,␣
 ↪'xgb__n_estimators': 1…
20  {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 5, 'xgb__reg_lambda': 10,␣
 ↪'xgb__n_estimators': …
49  {'xgb__subsample': 1.0, 'xgb__scale_pos_weight': 10, 'xgb__reg_lambda': 0.1,␣
 ↪'xgb__n_estimators'…

    mean_train_score  mean_test_score    diff, %
23          0.952854         0.604144  36.596383
40          0.804190         0.564820  29.765339
42          0.715695         0.554600  22.508805
10          0.646855         0.533580  17.511661
15          0.690478         0.524456  24.044489
4           0.614038         0.518911  15.491958
7           0.595346         0.514758  13.536372
..               …                …        …
12          0.457362         0.457175   0.040947
32          0.459323         0.457175   0.467649
39          0.457362         0.457175   0.040947
27          0.457175         0.457175   0.000007
```

```
35           0.457175          0.457175   0.000007
20           0.457175          0.457175   0.000007
49           0.457362          0.457175   0.040947

[50 rows x 4 columns]
```

[74]:
```
# Results
print("Best Parameters:", xgb_search.best_params_)
print("Best F1 Macro Score:", xgb_search.best_score_)

# Cross-validated predictions
y_train_pred = cross_val_predict(xgb_search.best_estimator_, X_train,␣
 ↪y_train_encoded, cv=10)

# Evaluation
print("\nCross-Validated Classification Report:")
print(classification_report(y_train_encoded, y_train_pred, zero_division=0))
```

```
Best Parameters: {'xgb__subsample': 0.8, 'xgb__scale_pos_weight': 1,
'xgb__reg_lambda': 1, 'xgb__n_estimators': 200, 'xgb__max_depth': 7,
'xgb__learning_rate': 0.1, 'xgb__gamma': 0, 'xgb__colsample_bytree': 0.9}
Best F1 Macro Score: 0.6041437379704678

Cross-Validated Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.20      0.30      1232
           1       0.87      0.98      0.92      6576

    accuracy                           0.85      7808
   macro avg       0.74      0.59      0.61      7808
weighted avg       0.83      0.85      0.82      7808
```

[75]:
```
plot_learning_curves(xgb_search.best_estimator_, X_train, y_train_encoded)
```



- Performance on the minority class (Class 0) is weak: F1-score = 0.30.
- Achieved strong overall performance with accuracy = 0.85 and F1 Macro = 0.61 & performed very well on the majority class (Class 1): F1-score = 0.92, recall = 0.98

The Learning curve shows: - Validation scores improve slightly with more data but plateau around 0.61. - Large and persistent train–test gap, indicating that the model memorises the training data and struggles to generalise.

[76]:
```python
# Save the best estimator (final trained model) from RandomizedSearchCV
dump(xgb_search.best_estimator_, 'models/xgb.joblib')
```

[76]: ['models/xgb.joblib']

2 Overall Model Comparison

[139]:
```python
# List of models to evaluate, including a baseline model.
models = [
    ("Baseline", DummyClassifier(strategy="most_frequent", random_state=7)),
    ("Random Forest", rf_grid_search.best_estimator_),
    ("Decision Tree", dt_grid_search.best_estimator_),
    ("Logistic Regression", logreg_search.best_estimator_),
    ("SVM", svm_grid_search.best_estimator_),
    ("XGBoost", xgb_search.best_estimator_)
]
```

```
# Execution times for each model (in minutes)
execution_times = {
    "Random Forest": execution_time_rf,
    "Decision Tree": execution_time_dt,
    "Logistic Regression": execution_time_lr,
    "SVM": execution_time_svm,
    "XGBoost": execution_time_xgb
}

# Create a function to calculate various performance metrics for a model
def get_metrics(model, X_train, y_train, y_train_encoded, model_name):
    pos_label = 2 if model_name != "XGBoost" else 0
    return {
        "Best Score": cross_val_score(model, X_train, y_train, cv=10,␣
↪scoring='f1_macro').mean(),
        "F1 Macro Score": cross_val_score(model, X_train, y_train, cv=10,␣
↪scoring='f1_macro').mean(),
        "Accuracy": cross_val_score(model, X_train, y_train, cv=10,␣
↪scoring='accuracy').mean(),
        "Precision of Serious Accident class": precision_score(
            y_train, cross_val_predict(model, X_train, y_train, cv=10),␣
↪pos_label=pos_label, zero_division=0),
        "Recall of Serious Accident class": recall_score(
            y_train, cross_val_predict(model, X_train, y_train, cv=10),␣
↪pos_label=pos_label, zero_division=0)
    }

# Initialise results dictionary
results = {
    "Model": [model[0] for model in models],
    "Best Score": [],
    "F1 Macro Score": [],
    "Accuracy": [],
    "Precision of Serious Accident class": [],
    "Recall of Serious Accident class": []
}

# Populate the results dictionary
for model_name, model in models:
    y_train_data = y_train_encoded if model_name == "XGBoost" else y_train
    metrics = get_metrics(model, X_train, y_train_data, y_train_encoded,␣
↪model_name)
    for metric, value in metrics.items():
        results[metric].append(value)

# Convert to DataFrame
```

```
results = pd.DataFrame(results).round(2)

# Sort by Recall
results = results.sort_values("Recall of Serious Accident class",␣
↪ascending=False)

# Transpose
results = results.set_index("Model").T

# Add execution time as last row
results.loc["Execution Time (minutes)"] = results.columns.map(execution_times)

# Add Baseline with None to avoid NaN but keep type compatibility
execution_times["Baseline"] = None

# Add execution time row
results.loc["Execution Time (minutes)"] = results.columns.map(execution_times)

# Replace NaN (from Baseline) with '-'
results.loc["Execution Time (minutes)"] = results.loc["Execution Time␣
↪(minutes)"].fillna("-")

# Move Baseline column to the end
cols = [col for col in results.columns if col != "Baseline"] + ["Baseline"]
results = results[cols]
# Convert to DataFrame
results = pd.DataFrame(results).round(2)
# Display final table
results
```

[139]:

Model	Logistic Regression	Decision Tree
Best Score	0.62	0.65
F1 Macro Score	0.62	0.65
Accuracy	0.76	0.81
Precision of Serious Accident class	0.33	0.40
Recall of Serious Accident class	0.48	0.44
Execution Time (minutes)	1.22	4.03

Model	Random Forest	XGBoost	SVM	Baseline
Best Score	0.61	0.61	0.46	0.46
F1 Macro Score	0.61	0.61	0.46	0.46
Accuracy	0.82	0.85	0.84	0.84
Precision of Serious Accident class	0.40	0.61	0.00	0.0
Recall of Serious Accident class	0.26	0.20	0.00	0.0
Execution Time (minutes)	2.73	1.65	1.82	-

3 Model Selection

3.0.1 Best 2 Models:

Based on the results, Decision Tree and Logistic Regression offer the best trade-off between accuracy, recall, and balanced performance, especially for the Severe class.

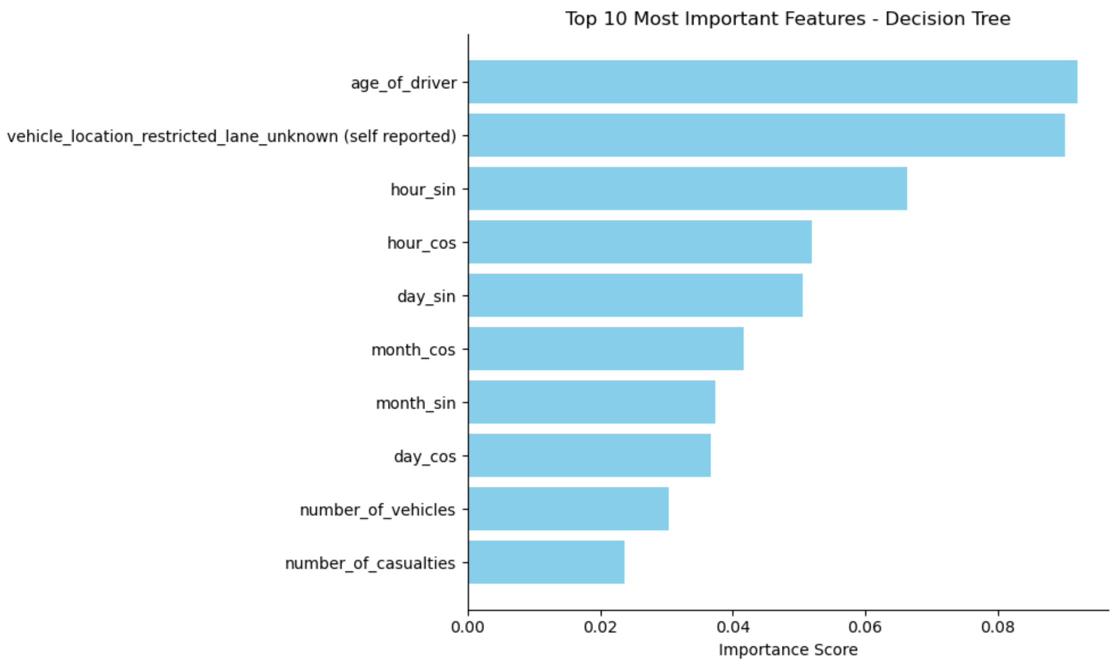
Decision Tree - Highest Accuracy (0.81). - Highest F1 Macro Score (0.65) — best overall class balance. - Second Best recall for the Serious accident class (0.44) — captures significant amount of serious accidents. - Execution time: 4.03 minutes — although the highest it is acceptable for the performance gain. - Interpretable model, balancing trade off between targeted initiative to reduce serious & slight accidents which making it useful for policy decisions.

Logistic Regression - High accuracy (0.76) and strong (second-highest) F1 Macro (0.62) — reliable overall performance. - Achieves best recall (0.48) for serious accidents, far superior to baseline (0.00). - Near identical F1 Macro as Random Forest (0.61), but with the best recall (0.48) for Serious cases. - Fastest Execution time: 1.22 minutes shows model is very efficient, offering fast and reliable results. - More balanced performance between classes compared to models like SVM or XGBoost.

3.1 Feature Importance

Decision Tree

[92]:
```python
# # Get the trained Decision Tree model
tree_model = dt_grid_search.best_estimator_

# Extract feature importance values
importances = tree_model.feature_importances_

# Get indices of the top 10 important features
top_indices = np.argsort(importances)[-10:][::-1]

# Plotting the top 10 features
plt.figure(figsize=(10, 6))
plt.barh(range(10), importances[top_indices], color='skyblue')
plt.yticks(ticks=range(10), labels=X_train.columns[top_indices])
plt.xlabel("Importance Score")
plt.title("Top 10 Most Important Features - Decision Tree")
plt.gca().invert_yaxis()  # Highest on top
# Remove top and right spines, keep x and y axes
sns.despine()

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



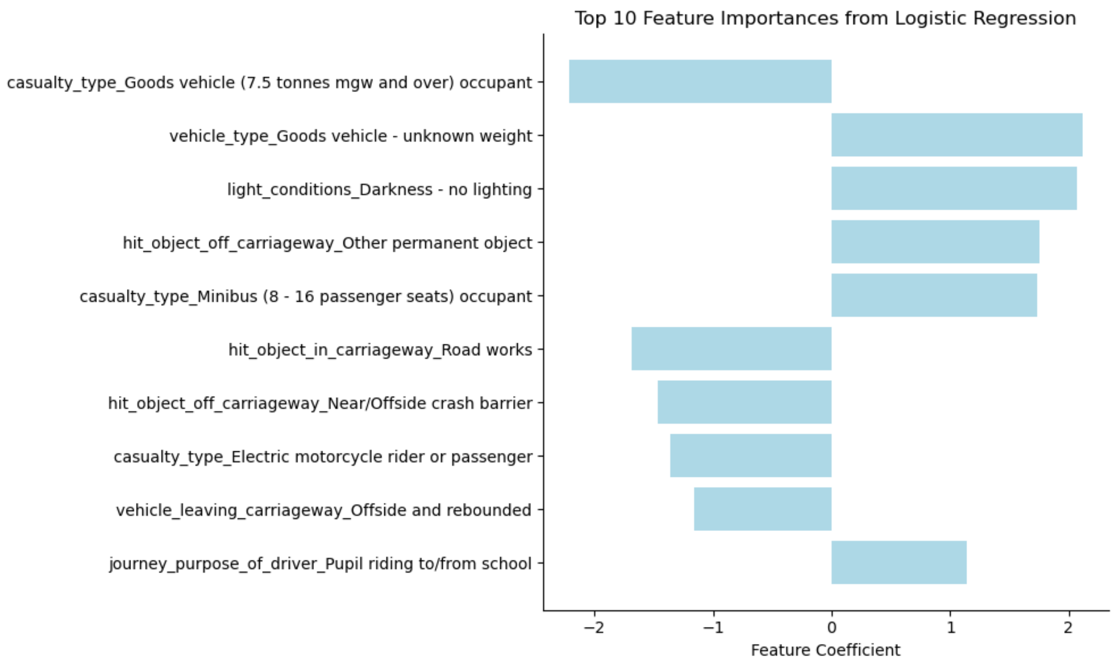
Most influential factors in determining whether a road accident is serious or slight:

- age_of_driver: Is the Most predictive feature. Driver age is has a strong determining factor on the accident severity. May indicate that certain age groups (e.g. very young or elderly) are more involved in a certain accident types.
- vehicle_location_restricted_lane_unknown (self reported): Theres a greater distingiush between severity of the accidents in restricted lanes and non-rstricted lanes.Suggests that enforcement or redesign of such lanes could reduce severity.
- Time of day influences severity: Which can supports time-based interventions.

Logistic Regression

[93]:
```
# Extract the trained pipeline and Logistic Regression model
logreg_model = logreg_search.best_estimator_
logreg_model = logreg_model.named_steps['model']

# Get the coefficients from the Logistic Regression model
coefficients = logreg_model.coef_[0]

# Sort top 10 features by absolute value
sorted_idx = np.argsort(np.abs(coefficients))[::-1][:10]

# Plot the top 10 feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(10), coefficients[sorted_idx], align='center', color='lightblue')
plt.yticks(range(10), np.array(X_train.columns)[sorted_idx])
```

```python
plt.xlabel('Feature Coefficient')
plt.title('Top 10 Feature Importances from Logistic Regression')
plt.gca().invert_yaxis()
plt.tight_layout()
# Clean up borders
sns.despine()
plt.show()
```



Feature importance plot for this Logistic Regression model:

- Target class 2 = serious
- Target class 3 = slight

The model uses class 3 (slight) as the positive class by default (since 3 > 2 in binary classification).

- casualty_type_Goods vehicle (7.5 tonnes mgw and over) occupant: This feature has the strongest influence. Model associates this feature with a higher likelihood of a serious accident.
- light_conditions_Darkness - no lighting: light_conditions_Darkness - no lighting: contributes strongly to the model's ability to classify slight accidents.

3.2 Evaluating Model with Test Set

3.2.1 Decision Tree

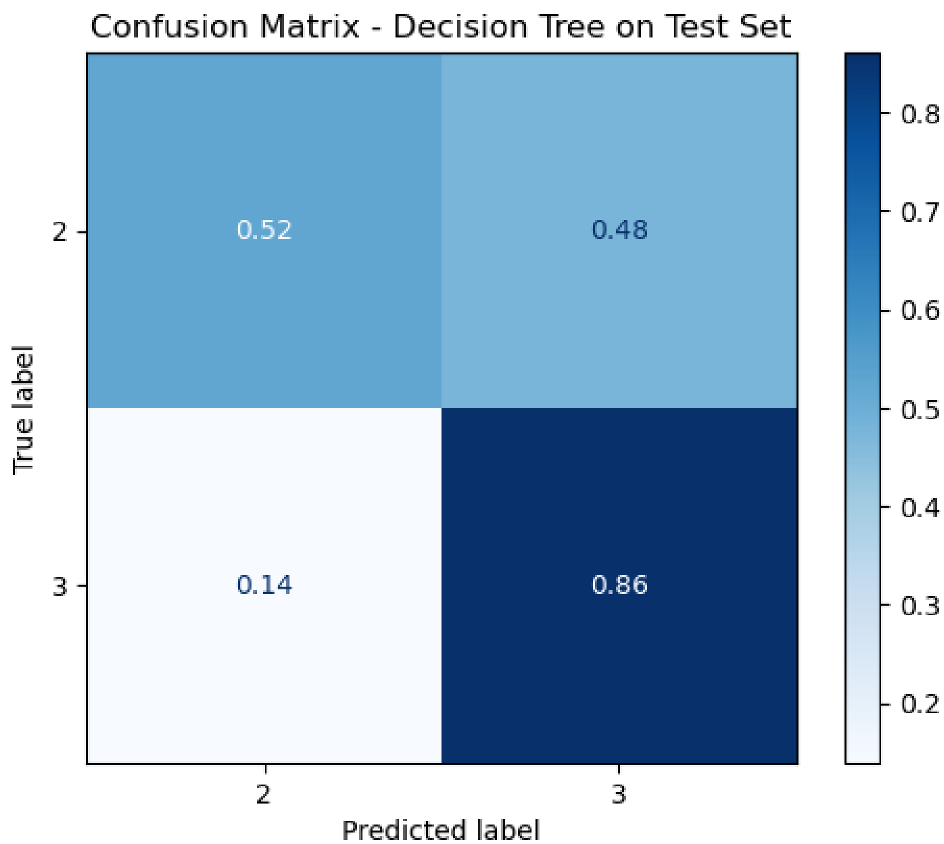
[91]:
```python
evaluate_model(dt_grid_search.best_estimator_, y_test, X_test)
```

```
              precision    recall  f1-score   support
```

```
           2       0.42      0.52      0.46       310
           3       0.90      0.86      0.88      1637

    accuracy                           0.81      1947
   macro avg       0.66      0.69      0.67      1947
weighted avg       0.83      0.81      0.82      1947
```

[95]:

```
from sklearn.metrics import ConfusionMatrixDisplay

# Generate predictions on the test set
yhat_test = dt_grid_search.best_estimator_.predict(X_test)

# Plot normalised confusion matrix
ConfusionMatrixDisplay.from_predictions(
    y_test,
    yhat_test,
    labels=dt_grid_search.best_estimator_.classes_,
    normalize="true",
    cmap=plt.cm.Blues
)

plt.title("Confusion Matrix - Decision Tree on Test Set")
plt.show()
```



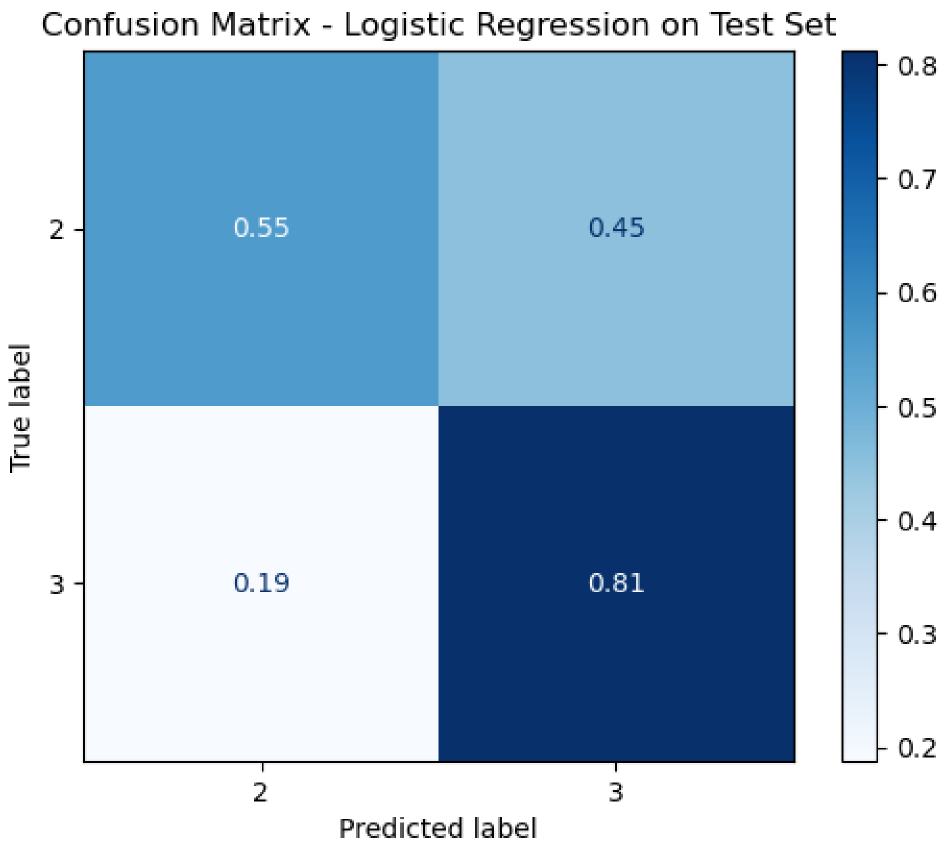
3.2.2 Logistic Regression

[96]:
```
evaluate_model(logreg_search.best_estimator_, y_test, X_test)
```

```
              precision    recall  f1-score   support

           2       0.36      0.55      0.43       310
           3       0.91      0.81      0.86      1637

    accuracy                           0.77      1947
   macro avg       0.63      0.68      0.64      1947
weighted avg       0.82      0.77      0.79      1947
```

[130]:
```
from sklearn.metrics import ConfusionMatrixDisplay

# Generate predictions on the test set
yhat_test = logreg_search.best_estimator_.predict(X_test)

# Plot normalised confusion matrix
```

```
ConfusionMatrixDisplay.from_predictions(
    y_test,
    yhat_test,
    labels=logreg_search.best_estimator_.classes_,
    normalize="true",
    cmap=plt.cm.Blues
)

plt.title("Confusion Matrix - Logistic Regression on Test Set")
plt.show()
```



Recommended Model: Decision Tree: - Offers better overall balance between accuracy, macro average, and class 3 performance. - Has lower recall for class 2, but makes up for it with higher precision and f1-score for this class.

4 Conclusion

The Decision Tree model was selected as the most appropriate due to its strong balance between accuracy (0.81), class-wise performance, and interpretability. It effectively distinguishes between serious and slight accidents.

For the stakeholders, this model provides a transparent and actionable tool to support safety-related decisions — such as allocating resources to high-risk scenarios, adjusting road safety strategies, or informing policy changes.

To further improve performance, future efforts could focus on increasing the volume of data available especially for the minority classs (serious accidents) to the model to enhance generalisation. While class weighting has already been applied to address class imbalance, additional improvements could come from combining these with advanced resampling techniques such as SMOTEENN or ADASYN. These approaches may help further improve recall for the minority class without significantly compromising overall accuracy.

REFERENCES

Transport For London (2023) Vision zero action plan - London, Vision Zero action plan Taking forward the Mayor's Transport Strategy. Available at: https://content.tfl.gov.uk/vision-zero-action-plan.pdf

Pekar, V. (2024). Big Data for Decision Making. Lecture examples and exercises. (Version 1.0.0). URL: https://github.com/vpekar/bd4dm