Module 6 exercises: Classification

The problem

- Loan data for every customer who borrowed £1000 for 12 months.
- Examples/cases = row = single customer/loan.
- Features = columns = fields = characteristics of customer/loan.
- Target: the characteristic/column you're trying to predict/understand.
    - For this problem, it is **default**, which takes a value of 0 if the applicant paid back their loan, and 1 if they did not.
- Problem: what model best describes the relationship of the features to the target?

Step 1: Learn pattern (model) from data which describes features relationship to target.

Step 2: Use pattern to guess unknown target from known features.

Exercise 1

1. Import the following packages:
    a. pandas
    b. matplotlib
    c. numpy
    d. seaborn
2. Load the following data file into a `DataFrame`:
    a. `loan_data.csv`

Exercise 2 – EDA

1. Familiarise yourself with the dataset. Seek to understand:
    - the types of data.

- distributions of features.
- obvious patterns.
- errors, nulls, outliers.

Exercise 3 - Data preparation

Prepare the dataset for modelling by performing the following steps:

1. Remove or impute null values (with justification).
2. Encode the target variable using `LabelEncoder`.
3. Encode categorical features using either `OneHotEncoder` or `OrdinalEncoder` (with justification).
4. Split the data into training and testing sets. Recall, **default** is our target.
5. Decide whether to scale (normalise/standardise) numeric features.

Exercise 4 – Building your first model

1. After preparing the dataset, we can fit our first classification model. Import the `LogisticRegression` model from scikit-learn's `linear_model` module.
2. Create a `LogisticRegression` model, then use the `fit` method to train the model on `X_train` and `y_train`.
3. Generate predictions from the model for `X_test` using the `predict` method, storing them in the variable `y_pred`.

Exercise 5 – Building a Decision Tree

1. Using the same approach as above, build a `DecisionTreeClassifier` model with the following parameter settings:

   - max_depth=2
   - min_samples_leaf=20
   - random_state = 42

   The model can be obtained from `sklearn.tree`.

2. Use the `plot_tree` function from `sklearn.tree` to visualise the decision tree you have built. You can copy the sample code below if required:

```
fig = plt.figure(figsize=(25,15))

_ = tree.plot_tree(dt,
              feature_names = X_train[cols].columns,
               class_names = ['Settles','Defaults'],
               filled=True
              )
```

3. If you have time, examine the `feature_importances_` attribute of the model. The name of each feature listend can be obtained from `feature_names_in_`.

Exercise 6 – Building a Random Forest

1. Using the same approach as above, build a `RandomForestClassifier` model with the following parameter settings:
2. random_state = 42
3. The model can be obtained from `sklearn.ensemble`.
4. If you have time, examine the `feature_importances_` attribute of the model. The name of each feature listend can be obtained from `feature_names_in_`.

Exercise 7 – Evaluating your models

1. Display sklearn's `classification_report` for your models, obtained from the `metrics` module.
2. Interpret the following elements of it:
   - accuracy
   - precision
   - recall
   - f1-score
3. Construct a confusion matrix using the following code:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

predictions = lr.predict(X_test[cols])
```

```
cm = confusion_matrix(y_test,
                      predictions)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=('Settle','Default'))

disp.plot();
```

How well does each model perform? Would you use it?

```
# Displaying precision and recall figures
print(classification_report(y_test, predictions, target_names=["Sett
les", "Defaults"]))

# PLotting the confusion matrix
predictions = rf.predict(X_test[cols])

cm = confusion_matrix(y_test,
                      predictions,
                      labels=rf.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=('Settle','Default')
                             )

disp.plot();
```

Exercise 8 – Feature Engineering and Hyperparameter Tuning

1. Experiment with features and hyperparameters to see whether you can improve the models you have built.

Extension (to be completed after Module 7: Model Selection and Evaluation)

Exercise 9 – ROC Curve

1. Using the below code, compare each model using an ROC curve.

Note: The code will only work if you create a list called models which contains each model you have built.

```
from sklearn.metrics import RocCurveDisplay
for i, model in enumerate(models):
    if i == 0:
        plot = RocCurveDisplay.from_estimator(model,
```

```
                                   X_test,
                                   y_test,
                                   plot_chance_level=True)
    axes = plot.ax_
else:
    RocCurveDisplay.from_estimator(model,
                                   X_test,
                                   y_test,
                                   plot_chance_level=False,
                                   ax=axes)
```

Which model performs best?