Module 7: Advanced Predictive Performance Evaluation

Learning outcomes

1. Apply sampling and validation techniques to address class imbalance and improve model performance.
2. Analyse the real-world applications of oversampling in ML/AI.
3. Use the k-fold cross-validation algorithm to assess predictive performance.
4. Determine suitable data or preprocessing strategies for ML.
5. Apply predictive modelling techniques to assess model performance.

Class imbalance

- In many real-world ML applications, the important outcomes (e.g. fraud or system failure) are rare.
- This imbalance can cause models to perform poorly on the minority class, which can be the most critical to detect.
- The goal is to adjust the model or data set so that the rare class is not overlooked.

Type I and type II errors

- **Type I error (false positive):** the model wrongly predicts the minority class when the instance actually belongs to the majority class (e.g. flagging a genuine transaction as fraud).
- **Type II error (false negative):** the model fails to detect the minority class (e.g. missing a fraudulent transaction).
- In imbalanced data sets, reducing type II errors is often more important, but this can increase the risk of type I errors.

Techniques to address class imbalance

Oversampling

- Oversampling increases the number of minority class samples in the training data.
- Oversampling helps the model pay more attention to rare but important cases.
- Methods include:
  - Random oversampling: duplicating existing minority class samples
  - Synthetic Minority Oversampling Technique (SMOTE): generating synthetic samples based on feature space similarities
- Advantages:
  - Reduces false negatives (type II errors)
  - Enhances recall and sensitivity

- Risks:
  - Overfitting, especially if the data is simply duplicated
  - Synthetic examples may introduce unrealistic patterns
- Implementing oversampling in Python:
  - Apply oversampling within each fold of cross-validation – not before splitting the data – to avoid data leakage and ensure fair evaluation
  - Monitor metrics such as recall, precision and false negative rate for the minority class.
  - Use `scikit-learn` for sample implementation.

Undersampling

- Undersampling reduces the number of majority class samples to balance the data.
- Undersampling is suitable when there's plenty of data.
- Advantages:
  - Forces the model to treat both classes with equal importance
- Risks:
  - Discards potentially useful information from the majority class
  - May harm model performance if too much data is removed

Stratified sampling

- Stratified sampling ensures that each training and validation split preserves the original class distribution.
- Stratified sampling prevents bias during model training and evaluation.
- Stratified sampling helps to create realistic and representative training and validation sets.

k-fold cross-validation

- k-fold cross-validation is a reliable method for evaluating model performance, especially when data is limited.
- The data set is split into `k` equally sized folds:
  - In each iteration, `k-1` folds are used for training and 1 for validation.
  - The process is repeated `k` times.
- Final performance is calculated as the average across all `k` runs.
- Benefits:
  - Reduces randomness in performance results
  - Makes efficient use of limited data
  - Helps avoid overfitting by ensuring generalisable results
- Limitations:

  - Computationally intensive, as the model must be trained `k` times
  - Can still suffer if class balance isn't maintained in each fold
  - Requires careful handling of preprocessing to prevent data leakage
- Implementing cross-validation with oversampling in Python:
  - The most common library is `scikit-learn`.
  - Always shuffle data before splitting (unless using time series).
  - Use `StratifiedKFold` for classification to preserve class proportions in each fold.
  - Combine with pipelines to prevent **data leakage** during preprocessing.