**DS Grad Programme 6 (Machine Learning – Part Two) – Classification**

Classification

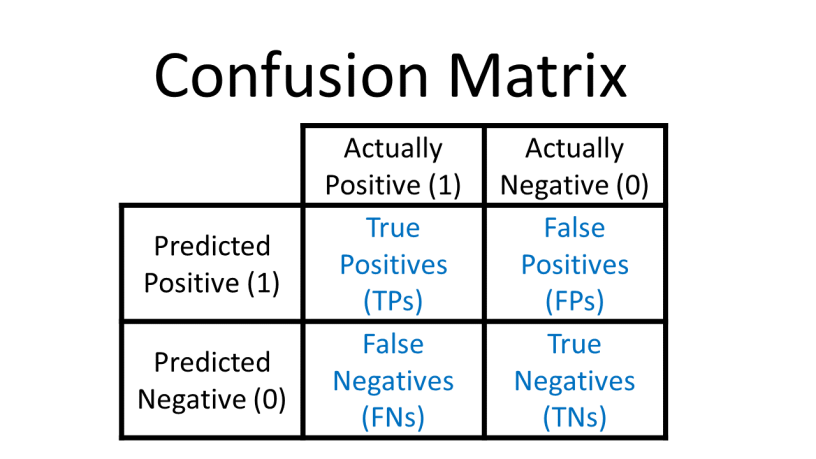
Classification is a supervised machine learning method where the model tries to predict the correct label of a given input data.

Examples

* Logistic Regression
* Support Vector Machine
* Decision Trees
* K-Nearest Neighbour

Measuring classification performance

* Useful tools
  + Confusion Matrix
    - This counts the key values needed to compute our metrics:
      * True Positive – Predicted positive and actually positive.
      * False Positive – Predicted positive but actually negative.
      * True Negative – Predicted negative and actually negative.
      * False Negative – Predicted negative but actually positive.



* + - Generates key metrics
      * Accuracy
        + How well the model performed in general.
        + Accuracy = (𝑇𝑃+𝑇𝑁)/(𝑇𝑃+𝐹𝑃+𝑇𝑁+𝐹𝑁)
      * Precision
        + How well the model predicted the positive classes.
        + The precision of a model is how good the models predictions are taking into account the false positives it produces.
        + This is the proportion of true positive results compared the total predicted positive results.
        + Precision = 𝑇𝑃/(𝑇𝑃+𝐹𝑃)
      * Recall
        + How well the model was able to identify positive classes during prediction.
        + Recall = 𝑇𝑃/(𝑇𝑃+𝐹𝑁)
      * F1 Score
        + Weighted average of precision and recall to account for both, though there are two optional metrics that we can use

Macro average

The macro average takes the score of each class then averages them.

The macro average tells us how our model performs equally across the classes

Weighted average

The weighted average takes the F1 score of each class and averages them with the weight of the class’s true value proportion.

The weighted average shows performance based on the abundance of each class.

* + - * + F1 Score = 2∗(𝑅𝑒𝑐𝑎𝑙𝑙 ∗𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛)/(𝑅𝑒𝑐𝑎𝑙𝑙+𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛)
      * Use the classification\_report() function to generate the results of each of these different tests
      * ROC/AUC curve
        + We can produce a plot that shows how our true positive rate varies with our false positive rate over a range of thresholds. This is called the Receiver Operating Characteristic (ROC) curve.
        + The ROC curve can help us understand the overlap between our predictions of different classes.
        + This shows us our performance compared to a random prediction, which is represented by a straight diagonal line. The further above this diagonal line our model is, the better it is performing.
        + The Area Under Curve (AUC) measures the area under the ROC curve, which will range from 0-1. An AUC value of 0.5 corresponds to the random model straight diagonal line, we want a value as close to 1 as we can get!
    - Deciding on which metrics to use
      * Accuracy not very useful as most datasets have a class imbalance
      * Precision and Recall help but each miss either false positives and false negatives, requiring a decision.
        + Are false positives worse than false positives? Depends on the context e.g. health.
      * F1 Score attempts to capture both patterns with a weighted average.
  + Decision trees also allow for feature selection by using the Information Gain metric
    - Ask questions to identify those options that will deliver most information gain (split the sample most effectively)
    - This is derived from measuring Entropy
      * Measure of on average, how many questions we would need to ask to predict a result, multiplied by the probability of each specific result occurring.
      * If a decision tree aims to have low entropy, then it aims to ask less questions to solve the problem
      * If a decision tree has higher entropy, it will be asking more questions, which will lead to overfitting
    - Information Gain determines how useful a feature is at determining the class label.
      * This can help you choose between features when narrowing down to the simplest model you can.
  + Hyperparameter optimisation
    - Can use GridSearchCV() class in sklearn.
      * The method uses K-fold cross validation as discussed in the previous chapter in order to ensure the performance is properly measured. This method for optimising performance can be done for both regression and classification problems, or any problem with hyperparameter. (see html for implementation)
    - RandomizedSearchCV()
      * Takes as an input: the chosen model, the distribution of values to sample from and the number of samples we want to take.
      * It samples from the ranges of hyperparameter values which allows us to search as much as we want, then we can try to improve on the best parameters given.
      * This is a non-brute force option, to give us clues re hyperparameter optimisation

Decision Trees

* When creating ensemble models with decision trees we can choose whether to employ bagging or boosting methods
* Bagging (eg. Random Forests)
  + T sets of the data are made, where each contains randomly selected samples with replacement, known as bootstrapping.
    - The size of each of these samples is the same as the size of the original dataset, but they will differ, due to being compiled using replacement
  + These sets are then trained in parallel with weak or base learners (with no tuning).
  + An average or majority is then taken to compute a more accurate estimate overall.
    - Classification – Majority vote is taken
    - Regression – Average is calculated
  + Evaluation
    - Easy to implement.
    - Helps reduce variance in a machine learning algorithm.
    - Difficult to interpret
    - Expensive as datasets grow.
  + Random Forests
    - Apply randomness in two different ways
      * Bootstrap aggregating (bagging)
        + Each decision tree in a Random Forest is trained on a random subset of the data points/rows.
        + Specifically, this subset is usually chosen with replacement, meaning a single sample can be used multiple times in the creation of a tree.
      * Feature randomness
        + The use of a random subset of the variables (or features) when determining the best split at each node of the trees.
        + When growing the individual trees, at each split in the construction process, the Random Forest algorithm considers only a random subset of the features rather than all features.
        + This approach ensures that the trees in the forest are de-correlated, which makes the forest more diverse and leads to lower correlation of errors and, consequently, a more accurate overall prediction.
      * In addition to each trial using a subset of the data points/rows, it also operates on a subset of the variables, at each split.
    - Output
      * Over thousands of iterations, with each of them voting, several things are achieved:
        + More difficult trends identified within the data.
        + Overfitting avoided as the model is more random and generalisable
        + Doesn’t heavily bias against a small group.
* Boosting (eg. XG Boost)
  + Boosting is similar to bagging, but allows models to learn sequentially, as opposed to in parallel.
    - Series of models made and with each iteration, misclassified data is penalised more.
      * This helps combat noise in data that leads to high bias.
      * Misclassification done using regularisation
      * Not sure how it identifies the misclassification
    - Examples include Adaboost, XGBoost and GradientBoost.
  + Evaluation
    - Incredibly powerful but can accentuate overfitting, as it learns the “quirks” of the data.
    - Sometimes seen as overkill for less complicated prediction problems.
    - Harder to deploy and maintain.

Class Distributions

* If there is an uneven class distribution in our target variable (which there usually is), then we need to approach our classification problem with care.
* Different options for addressing this problem include:
  + Stratification
    - When doing K-fold cross validation, we can ensure that the distribution is maintained in each of the K-folds by using StratifiedKFold().
  + Rebalance the dataset
    - For large inequality in class distributions we may need to rebalance the data set so that one class is not too much more significant than the others. This can be achieved by “re-weighting” the different classes. There are two options when doing this:
      * Change the class\_weight argument within the model chosen to “balanced”, the estimator will give a weight to each sample such that the the model training will treat all the samples of each class equally.
      * The data itself can be re-sampled in order to either produce more of the smaller class, or decrease the amount of the larger class. These methods are called over and undersampling respectively. This can either be implemented manually or done using a library such as sklearn.resample.
  + Oversampling
    - Use SMOTE or similar
  + Undersampling
    - Remove some of the more dominant class

Logistic Regression

* A logistic regression model calculates the probability that a given data point has a certain label
* If the probability of having a certain label is above a threshold (given by the distribution of the data) then the model gives the data that label.
* Method
  + Instead of fitting a linear model using regression, a logistic function is fitted to the data.
  + The value of the function corresponds to the probability that a data point is in a certain class.
  + This is done using a method called Maximum Likelihood Estimation, which calculates the probability of the data being what it is given a certain model.
  + It then tries different models and selects the one with the highest probability of producing the training data.