Imbalanced Classification

How to Effectively Work with Imbalanced Data

Delivered by: Noureddin Sadawi, PhD

About the Speaker

Name: Dr Noureddin Sadawi

Qualification: PhD Computer Science

Experience: Several areas including but not limited to Docker, Machine Learning and Data Science, Python and much more

Class Imbalance

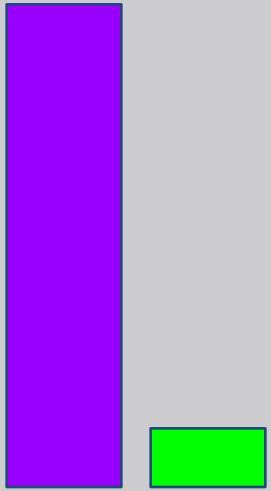
- Class imbalance refers to the problem when a classification dataset contains more than one class and number of instances in each class is not approximately the same
- For example, there might be a two-class classification dataset that contains 100 instances where the number of instances for one of the classes is 90 and for the other is 10
- This dataset is said to be imbalanced as the ratio of first class to second class instances is 90:10 (or 9:1)

What is Imbalanced Classification

- One might train a model that yields 90% accuracy but in reality it could be that the model is predicting the same class for the vast majority of testing data
- Imbalanced classification happens when the class distribution in the training dataset is unequal (Skewed Class Distribution)
  - Imbalance in the class distribution might be negligible
  - A severe imbalance is more challenging to model and may require specialized techniques
- Example real-world classification problems: rare disease data, fraud detection, spam detection, and churn prediction

Intuition for Imbalanced Classification

- One of the challenges for beginners working with imbalanced classification problems is what a specific skewed class distribution means
- For example, what is the difference and implication for a 1:10 vs. a 1:100 class ratio?
- The bars on the right show an example, class 0 (in purple) is much larger than class 1 (in green)



Code Example 1

Imbalanced Classification is a Challenge

- The main problem is the skewed class distribution (This is often exemplified by a binary (two-class) classification task where most of the examples belong to class 0 with only a few examples in class 1
- Because the class distribution is not balanced, most machine learning algorithms will perform poorly and require modification to avoid simply predicting the majority class in all cases
- Alternate methods for evaluating predictions on imbalanced examples are required

Code Example 2

Imbalanced Classification is a Challenge

- An additional level of complexity comes from the problem domain from which the examples were drawn
- It is common for the majority class to represent a normal case in the domain, whereas the minority class represents an abnormal case, such as a fault, fraud, outlier, anomaly, disease state, and so on
  - As such, the interpretation of misclassification errors may differ across the classes
- The abnormal, or minority, case is usually the class of interest (i.e. class 1)

Imbalanced Classification is a Challenge

- For example, misclassifying an example from the majority class as an example from the minority class (called a false-positive) is often not desired, but less critical than classifying an example from the minority class as belonging to the majority class, a so-called false negative
- This is referred to as the cost sensitivity of misclassification errors and is a second foundational challenge of imbalanced classification
    - Unequal Cost of Misclassification Errors
- These two aspects, the skewed class distribution and cost sensitivity, are typically referenced when describing the difficulty of imbalanced classification

Causes of Class Imbalance

- Two main groups of causes for the imbalance:
  1. Data sampling
  2. Properties of the domain
- It is possible that the imbalance in the examples across the classes was caused by the way the examples were collected or sampled from the problem domain
  1. This might involve biases introduced during data collection, and errors made during data collection
- The imbalance might be a property of the problem domain
- *As such, it is often infeasible or intractable to simply collect more samples from the domain in order to improve the class distribution. Instead, a model is required to learn the difference between the classes*

Classification Model Evaluation Metrics

- An evaluation metric quantifies the performance of a predictive model
- A classifier is only as good as the metric used to evaluate it
- If you choose the wrong metric to evaluate your models, you are likely to choose a poor model, or in the worst case, be misled about the expected performance of your model
- Choosing an appropriate metric is challenging generally in applied machine learning, but is particularly difficult for imbalanced classification problems
- *3.3. Metrics and scoring: quantifying the quality of predictions — scikit-learn 0.22.2 documentation*

Choosing Model Evaluation Metrics

- All metrics make assumptions about the problem or about what is important in the problem
- Therefore an evaluation metric must be chosen that best captures what you or your project stakeholders believe is important about the model or predictions, which makes choosing model evaluation metrics challenging
- Most of the standard metrics that are widely used assume a balanced class distribution, and because typically not all classes, and therefore, not all prediction errors, are equal for imbalanced classification

Accuracy and its Drawback(s)

- Classification accuracy is a metric that summarizes the performance of a classification model as the number of correct predictions divided by the total number of predictions
- How we calculate it:
  - First using a classification model to make a prediction for each example in a test dataset
  - The predictions are then compared to the known labels for those examples in the test set
  - Accuracy = Correct Predictions / Total Predictions
- Error Rate = Incorrect Predictions / Total Predictions
- Accuracy = 1 – Error Rate
- Error Rate = 1 – Accuracy

More on Accuracy

- Consider the case of an imbalanced dataset with a 1:100 class imbalance where each example of the minority class (class 1) will have a corresponding 100 examples for the majority class (class 0)
- In problems of this type, the majority class represents "*normal*" and the minority class represents "*abnormal*," such as a fault, a diagnosis, or a fraud
  - *Good performance on the minority class will be preferred over good performance on both classes*
- On this problem, a model that predicts the majority class (class 0) for all examples in the test set will have a classification accuracy of **99%**

Confusion Matrix

- A confusion matrix is a summary of the predictions made by a classification model organized into a table by class
- Each row of the table indicates the actual class and each column represents the predicted class
- A value in the cell is a count of the number of predictions made for a class that are actually for a given class

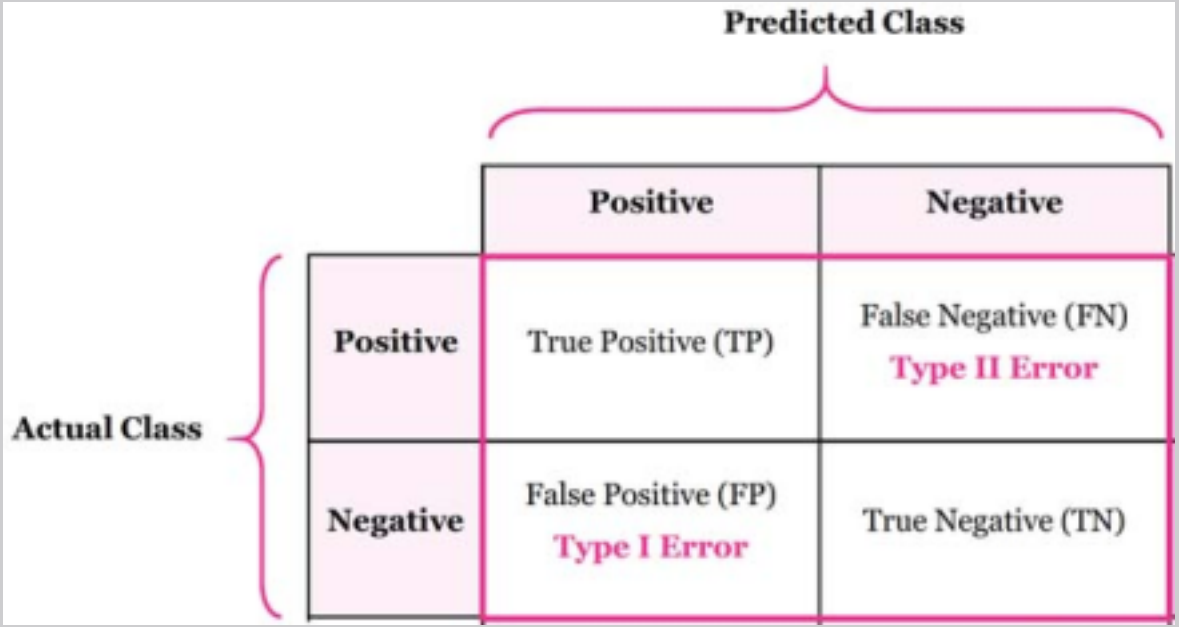
- The cells on the diagonal represent correct predictions, where a predicted and expected class align

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

Basic Counts

- A **true positive (TP)** is an outcome where the model *correctly* predicts the *positive* class
- A **true negative (TN)** is an outcome where the model *correctly* predicts the *negative* class
- A **false positive (FP)** is an outcome where the model *incorrectly* predicts the *positive* class
- A **false negative (FN)** is an outcome where the model *incorrectly* predicts the *negative* class

Confusion Matrix



The Boy Who Cried Wolf

A shepherd boy gets bored tending the town's flock. To have some fun, he cries out, "Wolf!" even though no wolf is in sight. The villagers run to protect the flock, but then get really mad when they realize the boy was playing a joke on them.

[Iterate previous paragraph *N* times.]

One night, the shepherd boy sees a real wolf approaching the flock and calls out, "Wolf!" The villagers refuse to be fooled again and stay in their houses. The hungry wolf turns the flock into lamb chops. The town goes hungry. Panic ensues.

The Boy Who Cried Wolf

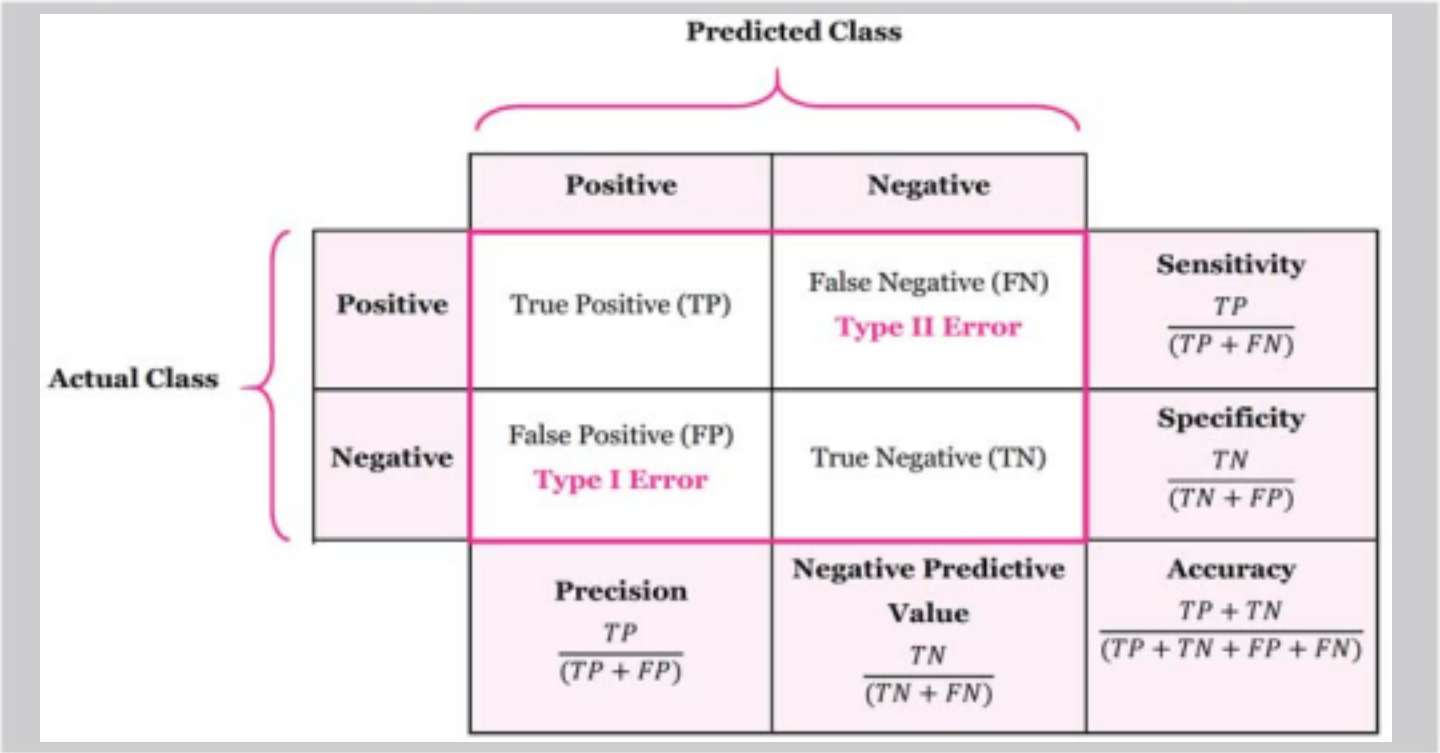
- "Wolf" is a **positive class**
- "No wolf" is a **negative class**

True Positive (TP): • Reality: A wolf threatened. • Shepherd said: "Wolf." • Outcome: Shepherd is a hero.	**False Positive (FP):** • Reality: No wolf threatened. • Shepherd said: "Wolf." • Outcome: Villagers are angry at shepherd for waking them up.
False Negative (FN): • Reality: A wolf threatened. • Shepherd said: "No wolf." • Outcome: The wolf ate all the sheep.	**True Negative (TN):** • Reality: No wolf threatened. • Shepherd said: "No wolf." • Outcome: Everyone is fine.

Precision, Recall and F-Measure

- **Precision** quantifies the number of positive class predictions that actually belong to the positive class:
  - Precision = TruePositives / (TruePositives + FalsePositives)
  - Appropriate when **minimizing false positives** is the focus
- **Recall** quantifies the number of positive class predictions made out of all positive examples in the dataset:
  - Recall = TruePositives / (TruePositives + FalseNegatives)
  - Appropriate when **minimizing false negatives** is the focus
- **F-Measure** provides a single score that balances both the concerns of precision and recall in one number:
  - F-Measure = (2 * Precision * Recall) / (Precision + Recall)

Confusion Matrix



Code Example 3

ROC and Precision-Recall Curves

- It is useful to predict a class probability instead of predicting a class value directly
- The reason for this is to provide the capability to choose and even calibrate the threshold for how to interpret the predicted probabilities
- For example, a default might be to use a threshold of 0.5, meaning that a probability in [0.0, 0.49] is a negative outcome (0) and a probability in [0.5, 1.0] is a positive outcome (1)
- This threshold can be adjusted to tune the behavior of the model for a specific problem
    - An example would be to reduce more of one or another type of error

ROC and Precision-Recall Curves

When making a prediction for a binary or two-class classification problem, there are two types of errors that we could make

- **False Positive**: Predict an event when there was no event
- **False Negative**: Predict no event when in fact there was an event

By predicting probabilities and calibrating a threshold, a balance of these two concerns can be chosen by the operator of the model

ROC and Precision-Recall Curves

For example, in a COVID-19 prediction system, we may be far more concerned with having low false negatives than low false positives

- A false negative would mean not treating a patient when in fact they need treatment
- A false positive means the person would receive treatment (or another check) when they didn't need to

ROC Curve

- A useful tool when predicting the probability of a binary outcome is the Receiver Operating Characteristic curve, or ROC curve
- It is a plot of the false positive rate (x-axis) versus the true positive rate (y-axis) for a number of different candidate threshold values between 0.0 and 1.0
    - It plots the false alarm rate versus the hit rate
- True Positive Rate (Sensitivity) = True Positives / (True Positives + False Negatives)
- Specificity = True Negatives / (True Negatives + False Positives)
- False Positive Rate (1 - Specificity) = False Positives / (False Positives + True Negatives)

ROC Curve

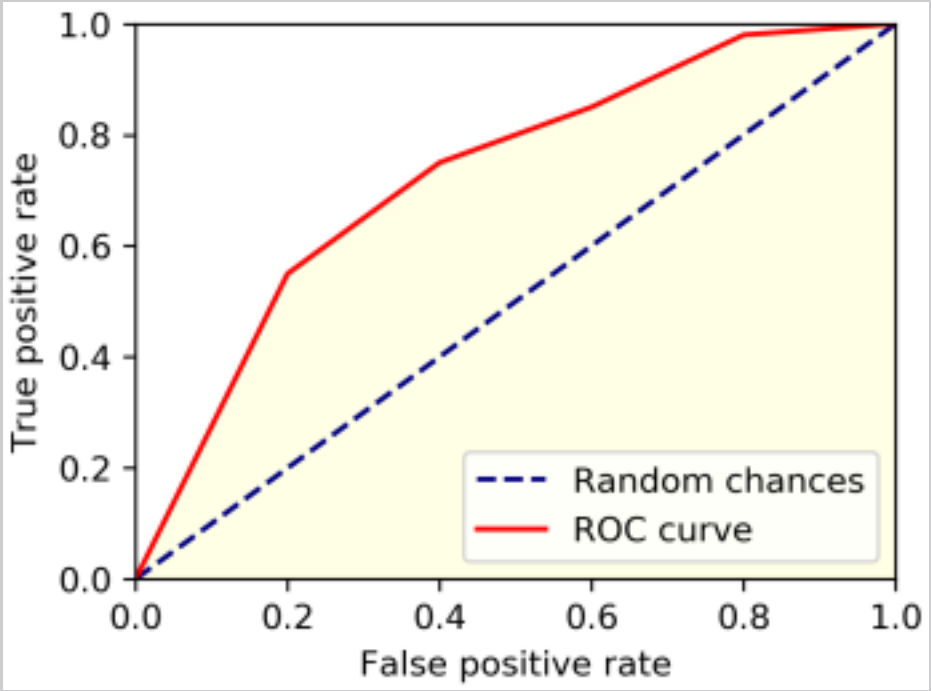
The ROC curve is a useful tool for a few reasons:

- The curves of different models can be compared directly in general or for different thresholds
- The area under the curve (AUC) can be used as a summary of the model skill

The shape of the curve contains a lot of information, including what we might care about most for a problem, the expected false positive rate, and the false negative rate

ROC Curve

- Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives
- Larger values on the y-axis of the plot indicate higher true positives and lower false negatives



ROC Curve

- How to build it in detail:
- Part 1: https://www.youtube.com/watch?v=fKo-HSBx4G0
- Part 2: https://www.youtube.com/watch?v=DiFL-i_zsFg

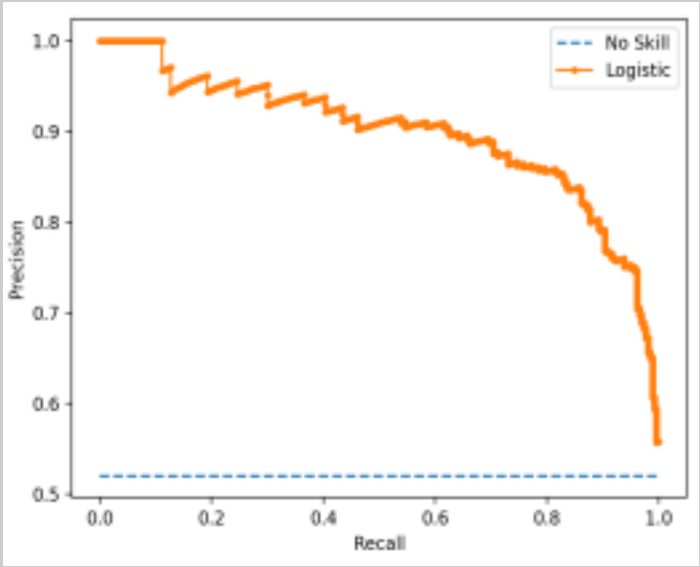
Code Example 4

Precision-Recall Curve

- Positive Predictive Power (Precision) = True Positives / (True Positives + False Positives)
- Recall (Sensitivity) = True Positives / (True Positives + False Negatives)
- Because of the high class imbalance, we are less interested in the skill of the model at predicting class 0 correctly, e.g. high true negatives
- Key to the calculation of precision and recall is that the calculations do not make use of the true negatives
  - It is only concerned with the correct prediction of the minority class, class 1

Precision-Recall Curve

- A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve
- A skilful model is represented by a curve that bows towards (1,1)
- For model selection, F-Measure summarizes model skill for a specific probability threshold (e.g. 0.5), whereas the area under curve summarize the skill of a model across thresholds, like ROC AUC.



Code Example 5

ROC Curve vs Precision-Recall Curve

- ROC curves should be used when there are roughly equal numbers of observations for each class
- Precision-Recall curves should be used when there is a moderate to large class imbalance
- The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance
- Some suggest that using a ROC curve with an imbalanced dataset might be deceptive and lead to incorrect interpretations of the model skill
- The main reason for this optimistic picture is because of the use of True Negatives in the False Positive Rate in the ROC Curve and the careful avoidance of this rate in the Precision-Recall curve

Probability Scoring Methods

- **The log loss score:** heavily penalizes predicted probabilities far away from their expected value
- **The Brier score:** gentler than log loss but still penalizes proportional to the distance from the expected value
- **The area under ROC curve:** summarizes the likelihood of the model predicting a higher probability for true positive cases than true negative cases

Log Loss Score

- AKA “logistic loss,” “logarithmic loss,” or “cross entropy”
- It can be used as a measure for evaluating predicted probabilities
- Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value
- The penalty is logarithmic, offering a small score for small differences (0.1 or 0.2) and enormous score for a large difference (0.9 or 1.0)
- A model with perfect skill has a log loss score of 0.0
- In order to summarize the skill of a model using log loss, the log loss is calculated for each predicted probability, and the average loss is reported

The Brier Score

- Calculates the mean squared error between predicted probabilities and the expected values
  - The score summarizes the magnitude of the error in the probability forecasts
- The error score is always between 0.0 and 1.0, where a model with perfect skill has a score of 0.0
- Predictions that are further away from the expected probability are penalized, but less severely as in the case of log loss
- The skill of a model can be summarized as the average Brier score across all probabilities predicted for a test dataset

The ROC AUC Score 1/2

- A predicted probability for a binary (two-class) classification problem can be interpreted with a threshold
- The threshold defines the point at which the probability is mapped to class 0 versus class 1, where the default threshold is 0.5
- Alternate threshold values allow the model to be tuned for higher or lower false positives and false negatives
- Tuning the threshold by the operator is particularly important in problems where one type of error is more or less important than another or when a model makes disproportionately more or less of a specific type of error

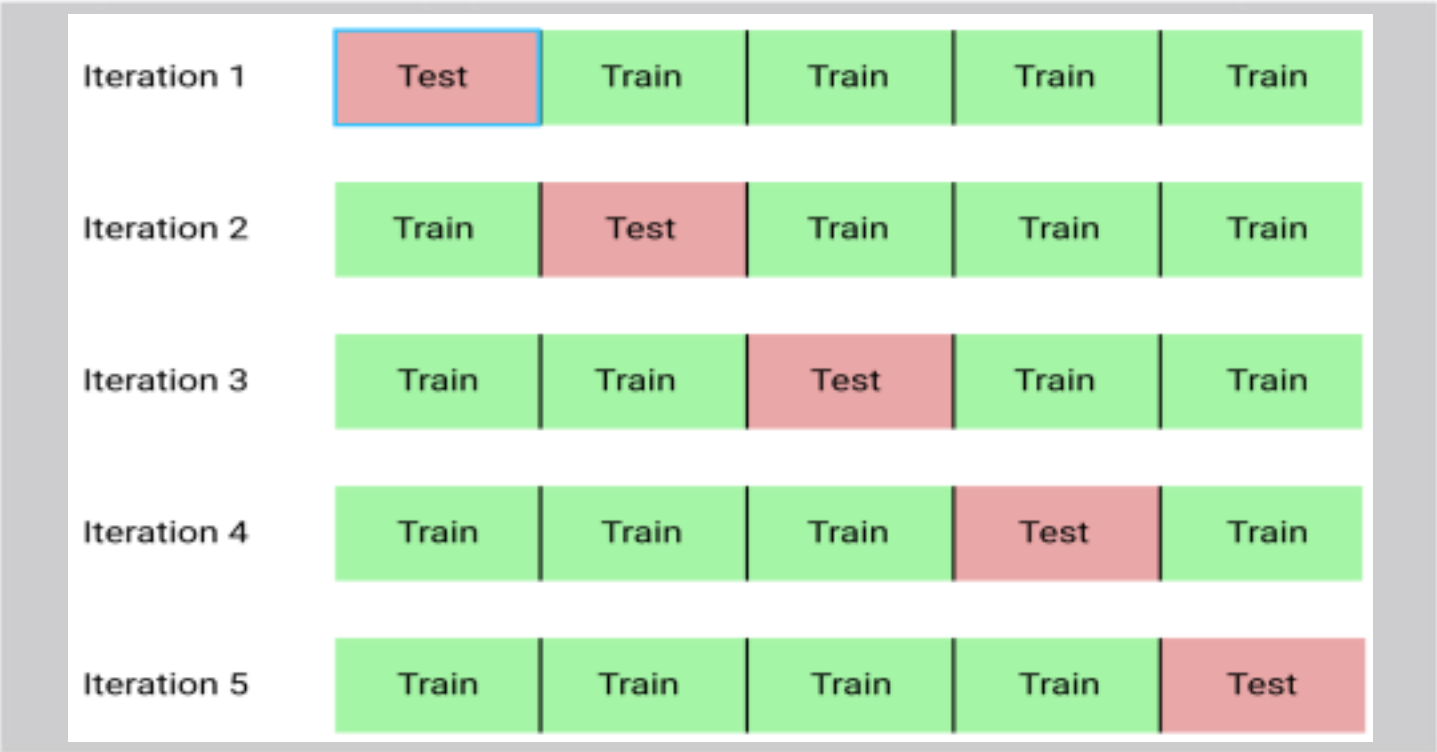
The ROC AUC Score 2/2

- The Receiver Operating Characteristic, or ROC, curve is a plot of the true positive rate versus the false positive rate for the predictions of a model for multiple thresholds between 0.0 and 1.0
- Predictions that have no skill for a given threshold are drawn on the diagonal of the plot from the bottom left to the top right
    - This line represents no-skill predictions for each threshold
- Models that have skill have a curve above this diagonal line that bows towards the top left corner

Cross Validation

- The k-fold cross-validation procedure involves splitting the training dataset into *k* folds
- The first *k-1* folds are used to train a model, and the holdout *k*th fold is used as the test set
- This process is repeated and each of the folds is given an opportunity to be used as the holdout test set
- A total of *k* models are fit and evaluated, and the performance of the model is calculated as the mean of these runs

Cross Validation



CV for Imbalanced Classification

- In CV the data is usually split into *k*-folds with a uniform probability distribution
- This is not appropriate for evaluating imbalanced classifiers
- It is likely that one or more folds will have few or no examples from the minority class
- This means that some or perhaps many of the model evaluations will be misleading, as the model need only predict the majority class correctly
- The Solution: split a dataset randomly in a way that maintains the same class distribution in each subset
- This is called stratification or stratified sampling and the target variable (*y*), the class, is used to control the sampling process

Code Example 6

End of Part 1

Data Sampling Methods

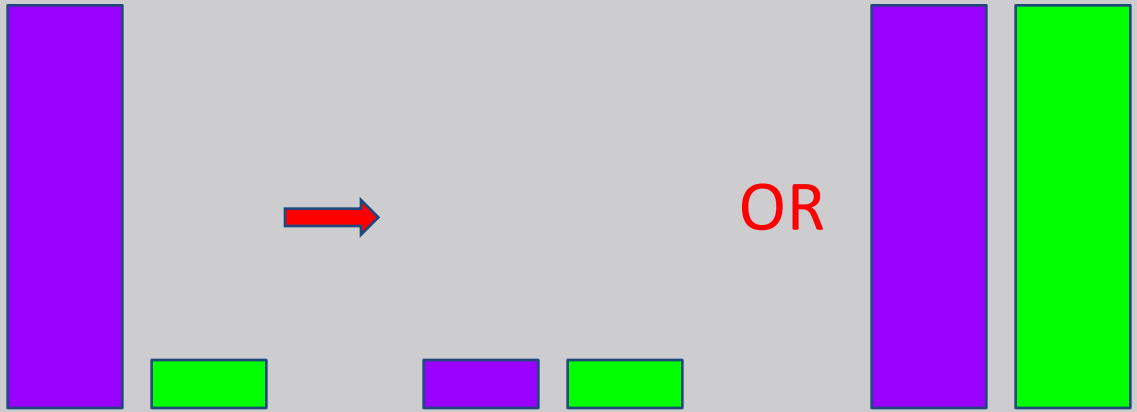
- Using data sampling we change the composition of the training dataset (the most popular solution to an imbalanced classification problem)
- These methods are usually simple to understand and implement
- Once applied to transform the training dataset, so many standard machine learning algorithms can then be used directly
- In fact, fitting models that learn the likelihood of observing examples from each class from the training dataset on a sampled training dataset with an artificially more equal class distribution allows them to learn a less biased prior probability and instead focus on the specifics (or evidence) from each input variable to discriminate the classes

Data Sampling Methods

- Sampling is only performed on the training dataset, the dataset used by an algorithm to learn a model
- It is not performed on the holdout test or validation dataset
- The reason is that the intent is not to remove the class bias from the model fit but to continue to evaluate the resulting model on data that is both real and representative of the target problem domain
- As such, we can think of data sampling methods as addressing the problem of relative class imbalanced in the training dataset, and ignoring the underlying cause of the imbalance in the problem domain

Data Sampling Methods

- Oversampling
- Undersampling



Oversampling Techniques

Oversampling methods duplicate examples in the minority class or synthesize new examples from the examples in the minority class

Example Methods:

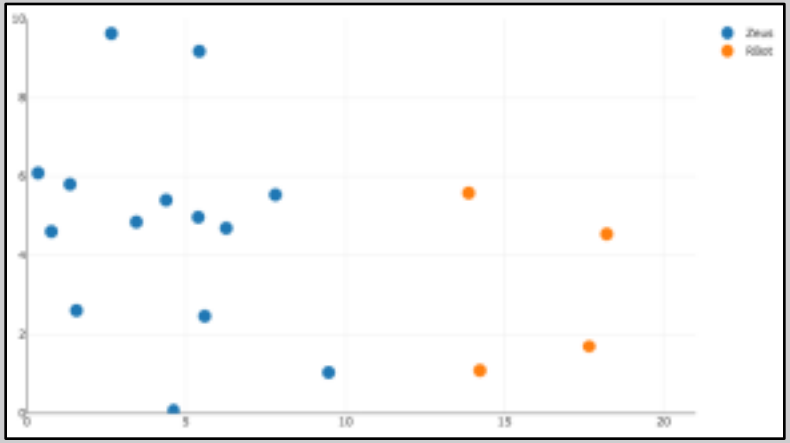
- Random Oversampling
- Synthetic Minority Oversampling Technique (SMOTE)
- Borderline-SMOTE
- Borderline Oversampling with SVM
- Adaptive Synthetic Sampling (ADASYN)

Oversampling Techniques

- **Random Oversampling**: randomly duplicating examples from the minority class in the training dataset
- **Synthetic Minority Oversampling Technique SMOTE:** works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample as a point along that line
  - The most popular and perhaps most successful oversampling method
  - There are many extensions to the SMOTE method that aim to be more selective for the types of examples in the majority class that are synthesized

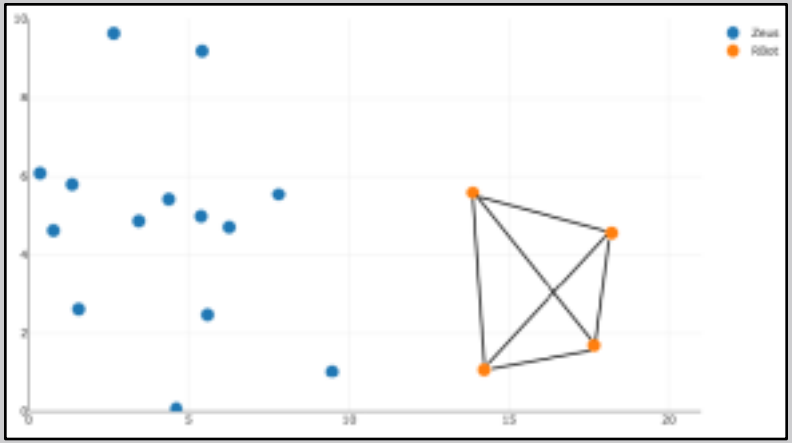
How SMOTE works 1/3

- Generates new instances (not real) based on existing real instances
- Needs at least two instances of the class to work



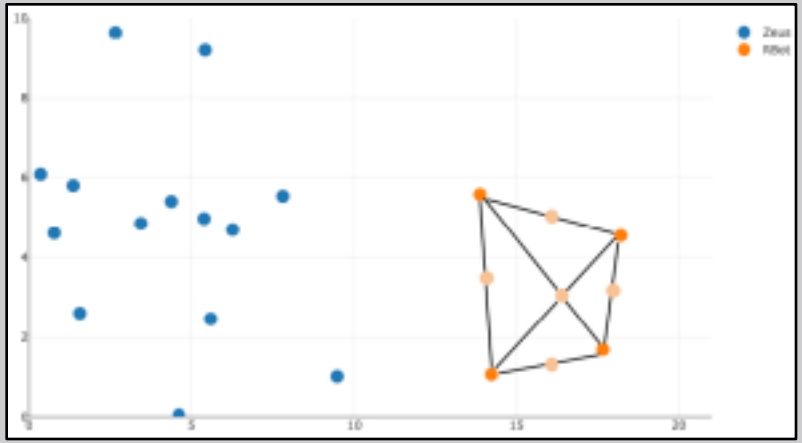
How SMOTE works 2/3

- Generates new instances (not real) based on existing real instances
- Needs at least two instances of the class to work



How SMOTE works 3/3

- Generates new instances (not real) based on existing real instances
- Needs at least two instances of the class to work



Code Example 7

Oversampling Techniques

- **Borderline-SMOTE:** involves selecting instances of the minority class that are misclassified and only generating synthetic samples that are "*difficult*" to classify
- **Borderline Oversampling with SVM:** an extension to SMOTE that fits an SVM to the dataset and uses the decision boundary as defined by the support vectors as the basis for generating synthetic examples, based on the idea that the decision boundary is the area where more minority examples are required
- **Adaptive Synthetic Sampling (ADASYN):** another extension to SMOTE that is designed to create synthetic examples in regions of the feature space where the density of minority examples is low, and fewer or none where the density is high (inversely proportional to the density of the examples in the minority class)

Undersampling Techniques

Undersampling methods delete or select a subset of examples from the majority class

Example Methods:

- Random Undersampling
- Condensed Nearest Neighbor Rule (CNN)
- Near Miss Undersampling
- Tomek Links Undersampling
- Edited Nearest Neighbors Rule (ENN)
- One-Sided Selection (OSS)
- Neighborhood Cleaning Rule (NCR)

Undersampling Techniques

- **Random Undersampling:** randomly deleting examples from the majority class in the training dataset
- **Condensed Nearest Neighbor rule (CNN):** works by enumerating the examples in the dataset and adding them to a store only if they cannot be classified correctly by the current contents of the store. It can be applied to reduce the number of examples in the majority class after all examples in the minority class have been added to the store
- **Near Miss Undersampling:** uses kNN to select examples from the majority class that have the smallest average distance to the X closest/furthest/ examples from the minority class.

Undersampling Techniques

- **Tomek Links Undersampling:** A Tomek Link refers to a pair of examples in the training dataset that are both nearest neighbors (have the minimum distance in feature space) and belong to different classes
  i. Tomek Links are often misclassified examples found along the class boundary and the examples in the majority class are deleted
- **Edited Nearest Neighbors (ENN):** involves using k=3 nearest neighbors to locate those examples in a dataset that are misclassified and deleting them. The ENN procedure can be repeated multiple times on the same dataset, better refining the selection of examples in the majority class

Undersampling Techniques

- **One-Sided Selection (OSS):** is an undersampling technique that combines Tomek Links and the Condensed Nearest Neighbor (CNN) Rule
  i. The Tomek Links method is used to remove noisy examples on the class boundary, whereas CNN is used to remove redundant examples from the interior of the density of the majority class
- **Neighborhood Cleaning Rule (NCR):** combines both the Condensed Nearest Neighbor (CNN) Rule to remove redundant examples and the Edited Nearest Neighbors (ENN) Rule to remove noisy or ambiguous examples

Combining Over/Undersampling Methods

- Usually using one method or the other on the training dataset is effective
- In some cases applying both types of techniques together can often result in better overall performance of a model fit on the resulting transformed dataset
- The purpose is to remove noisy points along the class boundary from both classes
- Some common combinations:
    a. SMOTE and Random Undersampling
    b. SMOTE and Tomek Links
    c. SMOTE and Edited Nearest Neighbors Rule

Resources

- https://en.wikipedia.org/wiki/Oversampling_and_undersampling_in_data_analysis
- The SMOTE Method: https://arxiv.org/abs/1106.1813
- Book: https://www.amazon.com/dp/1118074629/ref=as_li_ss_tl?&linkCode=sl1&tag=inspiredalgor-20&linkId=615e87a9105582e292ad2b7e2c7ea339&language=en_US

End of Part 2

Cost-sensitive Learning

- A subfield of machine learning that takes the costs of prediction errors (and potentially other costs) into account when training a machine learning model
- It is closely related to the field of imbalanced learning that is concerned with classification on datasets with a skewed class distribution
- Many conceptualizations and techniques developed and used for cost-sensitive learning can be adopted for imbalanced classification problems
- Based on the concept: **Not All Classification Errors Are Equal**

Classification Errors

- **Majority Class**: Negative or no-event assigned the class label 0
- **Minority Class**: Positive or event assigned the class label 1
- In Imbalanced Classification, classifying a negative case as a positive case is typically far less of a problem than classifying a positive case as a negative case (false positive vs false negative)
- Remember: the goal of a classifier on imbalanced binary classification problems is to detect the positive cases correctly and positive cases represent an exceptional event that we are most interested in
- Example areas (what is the cost of classification errors?):
    - Cancer Diagnosis Problem
    - Fraud Detection Problem
- Predicting positive cases as a negative case is more harmful, more expensive, or worse in whatever way!

Cost Sensitive Learning

A subfield of machine learning that is focused on learning and using models on data that have uneven penalties or costs when making predictions and more.

- **Error Minimization**: The conventional goal when training a machine learning algorithm is to minimize the error of the model on a training dataset
- **Cost**: The penalty associated with an incorrect prediction.
- **Cost Minimization**: The goal of cost-sensitive learning is to minimize the cost of a model on a training dataset

There is a tight-coupling between imbalanced classification and cost-sensitive learning. Specifically, an imbalanced learning problem can be addressed using cost-sensitive learning.

Cost-Sensitive Imbalanced Classification

Focused on first assigning different costs to the types of misclassification errors that can be made, then using specialized methods to take those costs into account.

The varying misclassification costs are best understood using the idea of a cost matrix

Cost Matrix: A matrix that assigns a cost to each cell in the confusion matrix

Actual		Predicted	
		Positive	Negative
	Positive	0	C(FN)
	Negative	C(FP)	0

Cost-Sensitive Algorithms

- Machine learning algorithms are rarely developed specifically for cost-sensitive learning
- Instead, the wealth of existing machine learning algorithms can be modified to make use of the cost matrix
- The scikit-learn Python machine learning library provides examples of these cost-sensitive extensions via the *class_weight* argument on the following classifiers:
  a. SVC
  b. DecisionTreeClassifier

Cost-Sensitive Algorithms

- Another approach to modifying existing algorithms is to use the costs as a penalty for misclassification when the algorithms are trained
- Given that most ML algorithms are trained to minimize error, cost for misclassification is added to the error or used to weigh the error during the training process
- This approach can be used for iteratively trained algorithms, such as logistic regression and artificial neural networks.

The scikit-learn library provides examples of these cost-sensitive extensions via the *class_weight* argument on the following classifiers:

- LogisticRegression
- RidgeClassifier

Class Weights

- By default, classes and the errors for each class are usually considered to have the same weighting (e.g. 1.0)
- These weightings can be adjusted based on the importance of each class
- The weighting is applied to the loss so that smaller weight values result in a smaller error value, and in turn, less update to the model coefficients
- A larger weight value results in a larger error calculation, and in turn, more update to the model coefficients.
- **Small Weight**: Less importance, less update to the model coefficients.
- **Large Weight**: More importance, more update to the model coefficients.

Assigning Class Weights

The weight assigned to class i should be proportional to the cost of misclassifying a member of class i

- A good starting point for imbalanced classification tasks is to assign costs based on the inverse class distribution
- For example: the class distribution of the test dataset is a 1:100 ratio for the minority class to the majority class. The inversion of this ratio could be used with 1 for the majority class and 100 for the minority class
- How to do Cost-Sensitive Learning
- Practical tips for class imbalance in binary classification

Logistic Regression

- **How Logistic Regression works:**
- Part 1: https://www.youtube.com/watch?v=BrmS1LFw-dQ
- Part 2: https://www.youtube.com/watch?v=PuUyya-ouvQ

Cost-Sensitive Logistic Regression

- LR is an effective model for binary classification tasks, although by default, it is not effective at imbalanced classification
- LR can be modified to be better suited for Imbalanced Classification
- The coefficients of the LR algorithm are fit using an optimization algorithm that minimizes the negative log likelihood (loss) for the model on the training dataset
- This involves the repeated use of the model to make predictions followed by an adaptation of the coefficients in a direction that reduces the loss of the model
- The calculation of the loss for a given set of coefficients can be modified to take the class balance into account

Cost-Sensitive Logistic Regression

- By default, the errors for each class may be considered to have the same weighting, say 1.0 (these weightings can be adjusted based on the importance of each class)
- The weighting is applied to the loss so that smaller weight values result in a smaller error value, and in turn, less update to the model coefficients
- A larger weight value results in a larger error calculation, and in turn, more update to the model coefficients
- The weightings are sometimes referred to as importance weightings
- The challenge of weighted LR is the choice of the weighting to use for each class

Code Example 8

Decision Trees

How Decision Trees work:

- *Part 1: https://www.youtube.com/watch?v=O__7lAqni7A*
- *Part 2: https://www.youtube.com/watch?v=WpA-Xbw4z_A*

Cost-Sensitive Decision Trees

- *Among all of the classifiers, induction of cost-sensitive decision trees has arguably gained the most attention*
- The decision tree algorithm is effective for balanced classification, although it does not perform well on imbalanced datasets
- The split points of the tree are chosen to best separate examples into two groups with minimum mixing
- When both groups are dominated by examples from one class, the criterion used to select a split point will see good separation, when in fact, the examples from the minority class are being ignored
- This can be overcome by modifying the criterion used to evaluate split points to take the importance of each class into account

Cost-Sensitive Decision Trees

- The tree is constructed by splitting the training dataset using values for variables in the dataset. At each point, the split in the data that results in the purest (least mixed) groups of examples is chosen
- Here, purity means a clean separation of examples into groups where a group of examples of all 0 or all 1 class is the purest, and a 50-50 mixture of both classes is the least pure
- The calculation of a purity measure involves calculating the probability of an example of a given class being misclassified by a split. Calculating these probabilities involves summing the number of examples in each class within each group
- The splitting criterion can be updated to take the purity of the split into account as well as the importance of each class (using weights)

Code Example 9

Support Vector Machines

- **How SVM Works:**
- Part 1: https://www.youtube.com/watch?v=-3URiBiFgIg
- Part 2: https://www.youtube.com/watch?v=BomqrcbJS4g

Cost-Sensitive SVM

- The SVM training algorithm seeks a line or hyperplane that best separates the classes
- The hyperplane is defined by a margin that maximizes the distance between the decision boundary and the closest examples from each of the two classes
- The data may be transformed using a kernel to allow linear hyperplanes to be defined to separate the classes in a transformed feature space that corresponds to a nonlinear class boundary in the original feature space.
- Typically, the classes are not separable, even with data transforms. As such, the margin is softened to allow some points to appear on the wrong side of the decision boundary. This softening of the margin is controlled by a regularization hyperparameter referred to as the soft-margin parameter, *lambda*, or capital-C ("*C*")

Cost-Sensitive SVM

- SVMs are known to perform poorly when there is a severe skew in the class distribution
    - As such, there are many extensions to the algorithm in order to make them more effective on imbalanced datasets
- The *C* parameter is used as a penalty during the fit of the model, specifically the finding of the decision boundary. By default, each class has the same weighting, which means that the softness of the margin is symmetrical
- Given that there are significantly more examples in the majority class than the minority class, it means that the soft margin and, in turn, the decision boundary will favor the majority class
- One way to extend SVM for imbalanced classification is to weight the *C* value in proportion to the importance of each class

Code Example 10

Cost-Sensitive Deep Learning

- We usually train neural network models using the backpropagation of error algorithm
- This involves using the current state of the model to make predictions for training set examples, calculating the error for the predictions, then updating the model weights using the error, and assigning credit for the error to different nodes and layers backward from the output layer back through to the input layer
- This training procedure can be modified so that some examples have more or less error than others
- The simplest way to implement this is to use a fixed weighting of error scores for examples based on their class where the prediction error is increased for examples in a more important class and decreased or left unchanged for those examples in a less important class

Code Example 11

Cost-Sensitive XGBoost

- Extreme Gradient Boosting (https://xgboost.readthedocs.io/en/latest/)
- It is an ensemble of decision trees algorithm where new trees fix errors of those trees that are already part of the model
- Trees are added until no further improvements can be made to the model
- XGBoost is an effective machine learning model, even on datasets where the class distribution is skewed
- XGBoost provides a hyperparameter designed to tune the behavior of the algorithm for imbalanced classification problems; this is the **scale_pos_weight** hyperparameter (default value is 1.0)

Cost-Sensitive XGBoost

- XGBoost is trained to minimize a loss function and the "*gradient*" in gradient boosting refers to the steepness of this loss function, e.g. the amount of error
- A small gradient means a small error and, in turn, a small change to the model to correct the error
- A large error gradient during training results in a large correction
- This has the effect of scaling errors made by the model during training on the positive class and encourages the model to over-correct them
- In turn, this can help the model achieve better performance when making predictions on the positive class
- If pushed too far, it may result in the model overfitting the positive class

Code Example 12

The End

The Course Ends Here!


thank you
danke
謝謝
ngiyabonga
teşekkür ederim
gracias
спасибо
Баярлалаа
рахмат
dank je
tapadh leat
bedankt
dziękuję
mochchakkeram
go raibh maith agat
obrigado
sukriya
kop khun krap
terima kasih
grazie
arigatô
takk
dakujem
мерси
merci
감사합니다
ευχαριστώ
ありがとう
hvala
maururu
sagolun