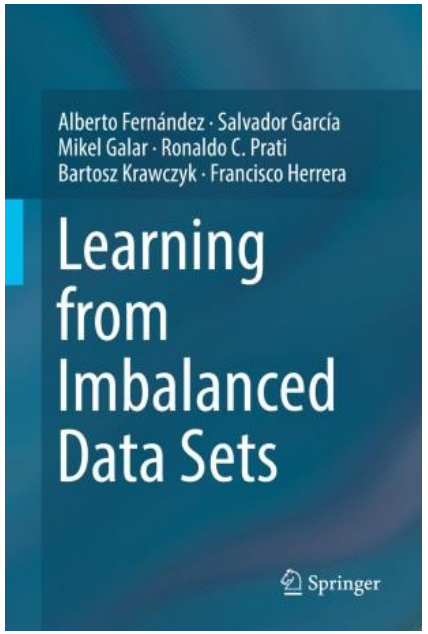
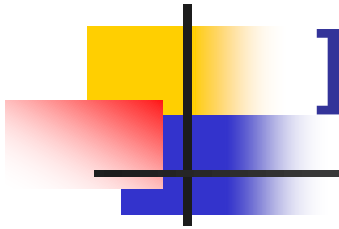
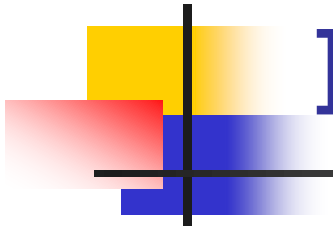
Imbalanced Classification





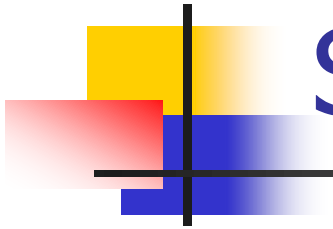
Introdução

- O problema de dados desbalanceados está presente em diversas áreas
  - Detecção de fraudes, evasão escolar, etc.
- Qualquer conjunto de dados com uma distribuição de classes desigual é tecnicamente desbalanceado
  - Entretanto, um conjunto de dados é dito desbalanceado quando existe uma significativa, ou em alguns casos extremos, desproporção entre o número de exemplos de uma dada classe em relação as outras



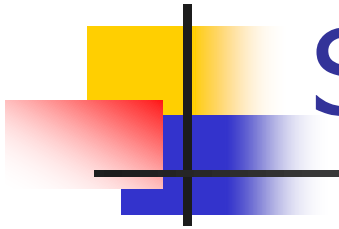
Introdução

- Este problema acaba por prejudicar o desempenho dos classificadores devido ao seu design orientado para a acurácia, o que normalmente faz com que a classe minoritária seja negligenciada
- Nos casos de classificação binária, a classe com maior proporção é referenciada como majoritária (ou negativa), e a de menor proporção como minoritária (ou positiva)



Soluções

- Abordagens em **nível de algoritmo** (**internas**): voltadas a adaptar os algoritmos de aprendizado de máquina existentes para que o modelo seja direcionado à classe minoritária
- Abordagens em **nível de dado** (**externas**): voltadas a equilibrar a proporção das classes por meio de técnicas de amostragem (etapa de pré-processamento)

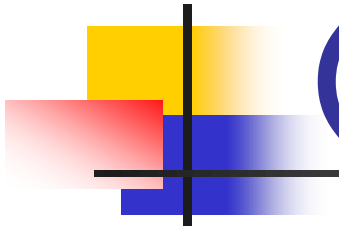


Soluções

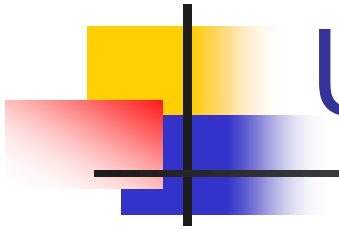
- Abordagens **sensível a custo**: voltadas a explorar situações entre as abordagens em nível de algoritmo e em nível de dado, inserindo custos às instâncias e/ou aos algoritmos, de maneira que classificações incorretas da classe minoritária possuam custos elevados

- Abordagens **baseadas em ensemble**: voltadas a combinar ensemble com alguma das abordagens anteriores, especificamente, em nível de dado ou sensível a custo

Abordagens em nível de dado (Externas)



- Divididas em 3 grupos
  - Undersampling
    - RUS, TL, US-CNN, OSS, US-CNN + TL, ENN, NCL, etc.
  - Oversampling
    - ROS, SMOTE, etc.
  - Híbridas
    - SMOTE+TL, etc.



Undersampling

- Na técnica de undersampling, instâncias da classe majoritária são removidas para que o conjunto de dados seja balanceado
  - A remoção pode ocorrer de maneira aleatória ou por intermédio de algum critério de seleção

RUS - Random Undersampling

- A técnica seleciona e remove aleatoriamente instâncias da classe majoritária do conjunto de dados
- A quantidade de instâncias a serem removidas, neste caso, variam de acordo com o tamanho do conjunto de dados, embora, em geral, busque-se uma proporção 1:1 entre as classes ao final do balanceamento
- Possui a desvantagem de aumentar a probabilidade de se perder dados úteis, por se tratar de uma solução não-heurística

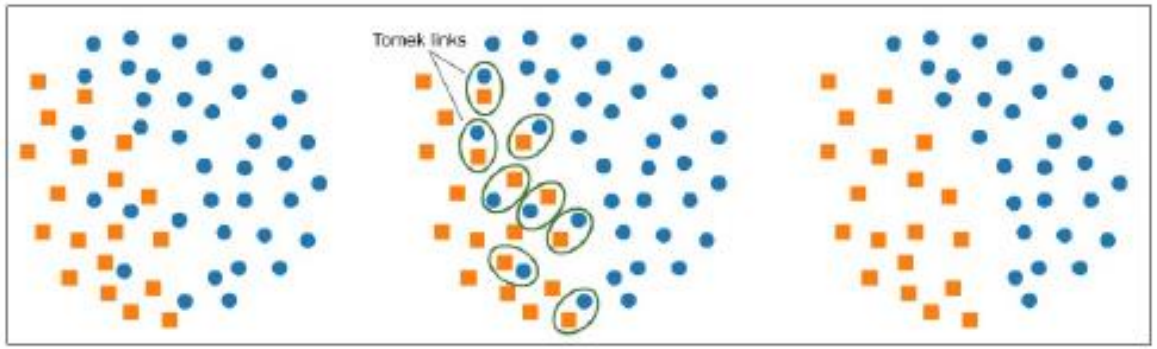
Técnicas Heurísticas

- Possuem um viés de limpeza de dados
  - Tomek Links (TL)
  - Condensed Nearest Neighbor Rule (US-CNN)
  - One-Sided Selection (OSS)
  - US-CNN + TL
  - Wilson’s Edited Nearest Neighbor Rule (ENN)
  - Neighborhood Cleaning Rule (NCL)
  - entre outras... (vide livro)

Tomek Links (TL)

Tomek Links (TL) (TOMEK, 1976). Nesta técnica as instâncias da classe majoritária a serem removidas são escolhidas da seguinte maneira: dado dois exemplos $E_i = (x_i, y_i)$ e $E_j = (x_j, y_j)$, em que $y_i \neq y_j$ (i.e., E_i e E_j pertencem a classes distintas) e $d(E_i, E_j)$ representa a distância entre E_i e E_j. O par (E_i, E_j) é chamado de *Tomek Link* se não existir um E_l tal que $d(E_i, E_l) < d(E_i, E_j)$ ou $d(E_j, E_l) < d(E_i, E_j)$. Ao final, para cada Tomek Link encontrado, elimina-se o exemplo pertencente a classe majoritária. A Figura 10 ilustra o resultado da aplicação da técnica.

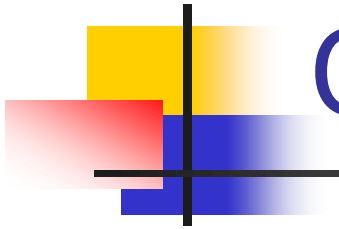
Figura 10 – Exemplo de *Tomek Links*. Adaptada de TOMEK (1976).



Neighborhood Cleaning Rule (NCL)

Wilson's Edited Nearest Neighbor Rule (ENN) (Wilson, 1972). Nesta técnica remove-se qualquer instância da classe majoritária cujo rótulo de classe difere do rótulo de classe de pelo menos dois de seus 3 vizinhos mais próximos (k-NN, k=3).

Neighborhood Cleaning Rule (NCL) (LAURIKKALA, 2001). Esta técnica modifica o ENN para aumentar a limpeza dos dados. Considerando um problema de duas classes, a técnica funciona da seguinte maneira: para cada exemplo $E_i = (x_i, y_i)$, seus três vizinhos mais próximos são encontrados. Se E_i pertence a classe majoritária e sua classificação, por meio de seus três vizinhos mais próximos, difere do rótulo original da classe, remove-se E_i. Se E_i pertence a classe minoritária e é classificada incorretamente, por meio de seus três vizinhos mais próximos, removem-se os vizinhos mais próximos que pertençam a classe majoritária.



Oversampling

- A técnica visa balancear os dados replicando instâncias ou criando novas instâncias da classe minoritária

ROS - Random Oversampling

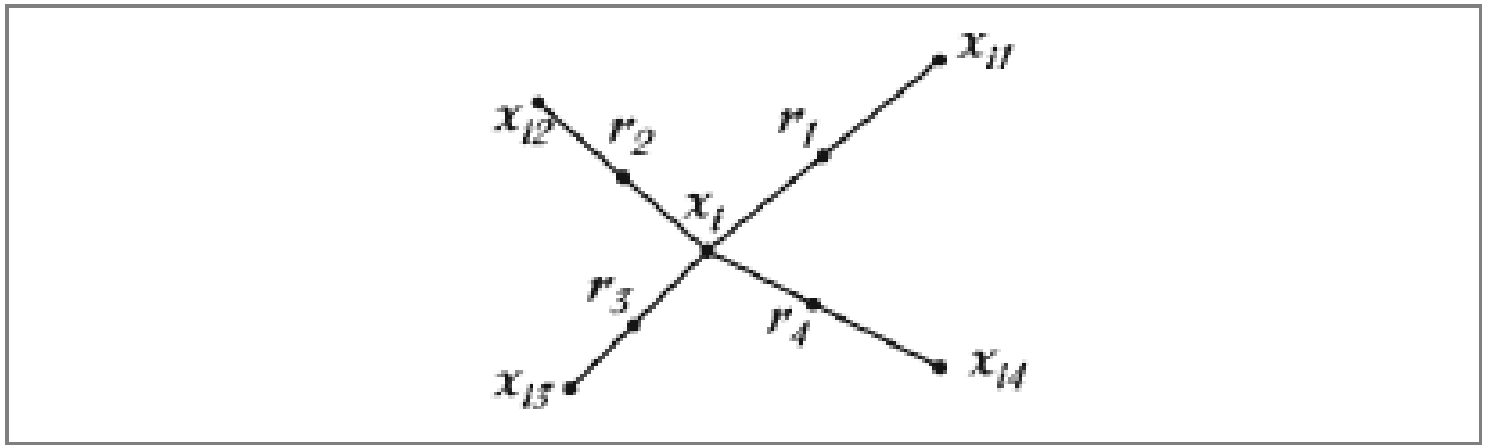
- Replica os exemplos selecionado-os de maneira aleatória
- Possui a desvantagem de gerar problemas na obtenção do modelo, uma vez que pode ocorrer overfitting
- Portanto, assim como no undersampling, técnicas alternativas a aleatória são utilizadas

SMOTE (SMOTE - Synthetic Minority Oversampling Technique)

- A ideia principal é criar exemplos sintéticos interpolando instâncias da classe minoritária com instâncias que pertençam as suas vizinhanças, aumentando assim a capacidade de generalização dos classificadores

SMOTE (SMOTE - Synthetic Minority Oversampling Technique)

- Uma instância x_i, pertencente a classe minoritária, é selecionada para criar novas instâncias sintéticas
- Considerando uma medida de distância, os k vizinhos mais próximos da instância selecionada, que também pertencem a classe minoritária (x_{i1} a x_{i4}), são selecionados
- Por fim, uma interpolação randomizada é realizada de modo a obter as no



SMOTE (SMOTE - Synthetic Minority Oversampling Technique)

- Interpolação
  - Para que cada instância i gere N instâncias sintéticas, um dos k vizinhos de i é selecionado aleatoriamente
  - A interpolação é realizada entre i e o vizinho selecionado considerando a diferença entre os seus vetores de características (atributos) (por atributo)
  - As diferenças são multiplicadas por um valor entre 0 e 1 e a nova instância sintética é gerada a partir da instância i acrescendo-se a mesma essa diferença "modificada"
  - Essa instância sintética representa um ponto aleatório entre a instância e um dos seus k vizinhos mais próximos

```
para (attr = 1 to numattrs) faça
    dif = mInstancias[nnarray[nn]][attr] - mInstancias[i][attr]
    gap = random(0,1)
    mISinteticas[newindex][attr] = mInstancias[i][attr] + (dif . gap)
fim
```

SMOTE (SMOTE - Synthetic Minority Oversampling Technique)

- Assim como as técnicas de subamostragem, as técnicas de sobreamostragem possuem diversas variantes, em geral, extensões do SMOTE
- Essa diversidade ocorre devido a sua configuração original poder gerar sobreposição entre as classes – pode-se aumentar o espaço da classe minoritária, introduzindo instâncias sintéticas dessa classe ao espaço da classe majoritária

Técnicas Híbridas

- Buscam o equilíbrio ideal entre a remoção de instâncias pertencentes a classe majoritária (undersampling) e a geração de instâncias sintéticas pertencentes a classe minoritária (oversampling)
  - Objetivo: obter o melhor desempenho possível em qualquer conjunto de dados desbalanceado

Técnicas Híbridas

- Frequentemente, grupos de classes não são bem definidos, uma vez que exemplos da classe majoritária podem estar "invadindo" o espaço da classe minoritária
- O oposto também pode ser verdadeiro, uma vez que a interpolação via SMOTE pode aumentar o espaço da classe minoritária, introduzindo instâncias sintéticas dessa classe ao espaço da classe majoritária, causando uma sobreposição entre as classes

Técnicas Híbridas

- Assim, a ideia das técnicas é criar grupos de classes bem definidos realizando uma limpeza nos dados após o oversampling, removendo instâncias de ambas as classes

SMOTE + Tomek Links

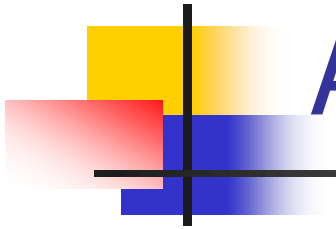
SMOTE + Tomek Links (BATISTA; PRATI; MONARD, 2004). Nesta técnica instâncias sintéticas da classe minoritária são criadas e instâncias da classe majoritária e minoritária são removidas da seguinte maneira: inicialmente, o conjunto de dados é sobreamostrado (*oversampling*) via SMOTE e, na sequência, os Tomek Links são identificados e removidos, produzindo um balanceamento com grupos de classes bem definidos.

Abordagens sensível a custo

- Cost-sensitive learning refers to a specific set of algorithms that are sensitive to different costs associated with certain characteristics of considered problems

Abordagens sensível a custo

- Two distinct views on cost-sensitive classifiers exist
  - Cost associated with features: this scenario assumes that acquiring a certain feature is connected with a given cost, being also known as test cost
    - Example: obtaining a feature involves invasive tests on humans, or the measurement procedure is unpleasant, painful, or difficult to perform
    - Goal: create a classifier that obtains the best possible predictive performance, while utilizing features that can be obtained at lowest possible cost (or the sum of costs being below a given threshold)
      - This can be seen as a multi-objective learning, where we try to strike a balance between performance and cost of used features

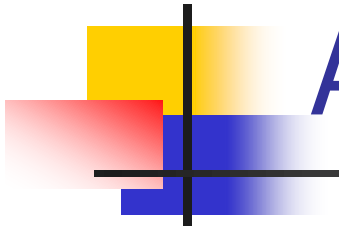


Abordagens sensível a custo

- Two distinct views on cost-sensitive classifiers exist…
  - Cost associated with classes: this scenario assumes that making errors on instances coming from certain classes causes is connected with a higher cost
    - Example: giving a credit to a person with a bad credit score will potentially cause higher loses to a bank than declining credit to a person with a good score
    - Goal: train a classifier in such a way that it will focus on classes that have higher costs assigned to them

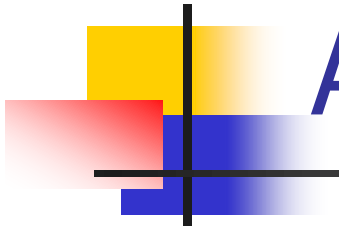
Abordagens sensível a custo

- It is also often used in balanced scenarios, where incorrect classification outcomes may lead to severe consequences
- The cost-sensitive learning can be seen as a specific type of algorithm-level approach
- It assumes asymmetric misclassification costs between classes, defined in a form of a cost matrix



Abordagens sensível a custo

- Standard machine learning methods most commonly use so-called 0–1 loss function, which assigns value 0 to a correctly classified instance and value 1 to an incorrectly classified one
- Then the training procedure aims at minimizing the overall cost, i.e., minimizing the number of wrong predictions



Abordagens sensível a custo

- Therefore, the 0–1 loss function over imbalanced data can be easily minimized by focusing on majority class and overlooking (or in extreme cases even completely ignoring) minority class

Abordagens sensível a custo

- Cost-sensitive learning aims at alleviate this problem by adapting a different loss function, with different costs associated with each class

	True positive	True negative
Predicted positive	$C(0,0)$	$C(0,1)$
Predicted negative	$C(1,0)$	$C(1,1)$

Cost matrix for a two-class problem

Such a cost can be seen as a penalty factor introduced during a classifier training procedure (or, in some cases, during prediction step)

Therefore, during the training procedure, the classifier is forced to minimize the overall cost, focusing on instances coming from this skew distribution

Abordagens sensível a custo

- For a standard two-class problem a cost-sensitive classifier will classify given instance x as belonging to positive class if and only if:

Predito

$$P(0|x)\cdot C(1,0)+P(1|x)\cdot C(1,1)\leq P(0|x)\cdot C(0,0)+P(1|x)\cdot C(0,1),$$

FN TN TP FP

Em geral, zero

Em geral, zero

$$P(0|x)\cdot C(1,0)\leq P(1|x)\cdot C(0,1).$$

Abordagens sensível a custo

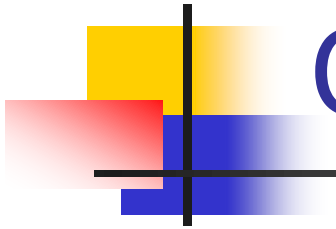
- Cost-sensitive learning algorithms can be separated into two main groups:
  - Direct approaches: this methodology is based on directly introducing the misclassification cost into the training procedure of a classifier
    - This directly corresponds to other algorithm-level approaches, with difference of utilizing the cost matrix

Abordagens sensível a custo

- Cost-sensitive learning algorithms can be separated into two main groups (cont.):
  - Meta-learning approaches: this methodology is based on modifying either the training data or the outputs of a classifier
    - It does not modify the underlying training algorithm, thus making this a suitable approach for almost any type of a classifier

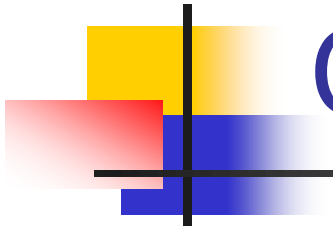
Abordagens sensível a custo

- Cost-sensitive learning algorithms can be separated into two main groups (cont.):
  - Meta-learning approaches (cont.): can be applied during two different steps of the classification process
    - Preprocessing: aim at modifying the training set
      - The most popular approach includes weighting the instances according to a provided cost matrix, thus allowing for assigning higher importance to minority objects
    - Postprocessing: aim at modifying the outputs of a classifier during the classification phase
      - The entire effort is moved to introducing the cost factor when a decision about a new instance is being made



Obtaining the Cost Matrix

- The effectiveness of cost-sensitive learning relies strongly on the supplied cost matrix
  - Too low costs will not allow to properly adjust the classification boundaries, while too high cost will lead to loss of generalization capabilities on the remaining classes



Obtaining the Cost Matrix

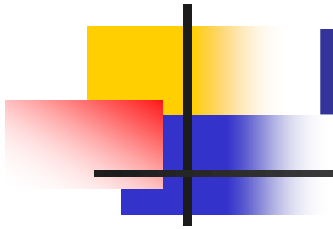
- Two possibilities:
  - Cost matrix provided by an expert
    - Example: credit card fraud detection
      - The cost is given directly as an average monetary loss in a case of accepting a fraudulent transaction (this is C(i, j)) and in case of losing a customer after rejecting a valid transaction (this is C(j, i))

Obtaining the Cost Matrix

- Two possibilities (cont.):
  - Cost matrix estimated using training data
    - This requires either heuristic setting of cost values or learning them from training data
      - The most popular heuristic approach lies in utilizing the IR* as a direct way to estimate the cost
        - In this set-up C(i, j) = IR and C(j, i) = 1, where minority is the i^{th} and majority is the j^{th} class (to allow for cases with multiple classes)

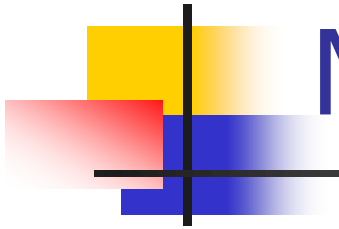
IR* = number of negative class examples divided by the number of positive class examples (Example: 1:100 → IR=100/1 = 100; 139/77=1,81 (≅ 1:2))

https://homes.cs.washington.edu/~pedrod/papers/kdd99.pdf



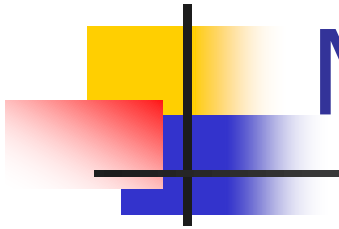
MetaCost

- Flexible in adapting classifiers to cost-sensitive scenarios
- It works as a wrapper method that can be applied to any type of classifier, regardless of the type of output it returns (either class labels or probability estimates)



MetaCost

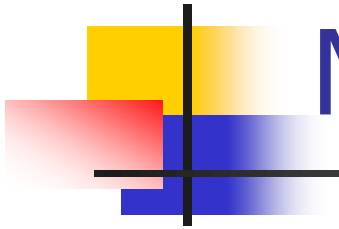
- It is based on the assumption that with the introduction of a cost matrix, the classification boundary should be adapted in favor of the classes with a higher cost assigned



MetaCost

- It postulates that if these instances would be relabeled to their optimal classes suggested by the cost matrix, then there is no further need for data preprocessing and a standard classifier using 0–1 loss function can be used

- Modified training set should allow any classifier to find optimal decision boundaries that will minimize the misclassification cost



MetaCost

- It is a preprocessing meta-learning approach that utilizes ensemble-based data manipulation
  - The original training set is used to learn multiple classifiers by bootstrapping instances (following Bagging idea)
  - Then a probability for each instance belonging to each of classes is estimated using a fraction of votes it receives from the ensemble
  - Then training instances are relabeled to minimize the cost (conditional risk)
  - Finally, the ensemble is discarded (as it was used only for preprocessing step) and a new classifier is being trained on the modified set of instances

MetaCost

"Força" a aprender a fronteira de decisão correta

Algorithm 1 MetaCost algorithm

Require: S: Training set; S': relabeled training set; L: Number of Bagging iterations; n: Bootstrap size; Ψ: classification algorithm; M: # of classes.

1:
2: **for** $l = 1$ to L **do**
3: $S_l \leftarrow$ RandomSampleReplacement(n,S)
4: $\Psi_l \leftarrow$ train Ψ on S_l
5: **end for**
6:
7: **for** $s = 1$ to $|S|$ **do**
8: **for** $i = 1$ to M **do**
9: **if** Ψ returns probabilistic outputs **then**
10: obtain $P(i|x_s, \Psi_l)$
11: **else**
12: $P(i|x_s, \Psi_l) = 1$ for the class predicted and 0 for others
13: **end if**
14: **if** use all bootstraps for each instance **then**
15: l ranges over all Ψ_l Likely to have lower variance
16: **else**
17: l ranges over all Ψ_l such that $x_s \notin S_l$ Likely to have lower bias
18: **end if**
19: $P(i|x_s) = \frac{1}{\sum_l 1} \sum_l P(i|x_s, \Psi_l)$
20: **end for**
21: $x'_s \leftarrow$ relabel x_s according to $\text{argmin}_l \sum_j^M P(m|x_s)C(i, j)$
22: $S' \leftarrow S' \cup x'_s$
23: **end for**
24:
25: $\Psi_{final} \leftarrow$ train Ψ on S'

https://github.com/Treers/MetaCost

Cost-Sensitive Decision Trees

- The two most important approaches to cost-sensitive decision tree induction are:
  - Direct Approach with Cost-Sensitive Splitting (splitting criterion modification)
    - https://icml.cc/Conferences/2004/proceedings/papers/136.pdf
  - Meta-learning Approach with Instance Weighting (instance weighting)

Direct Approach with Cost-Sensitive Splitting

- Induction of a cost-sensitive tree takes into account two different costs
  - Test cost of a^{th} feature tc(a)
  - Misclassification cost of instance mc(x)
    - An uniform value of tc(a) can be assumed when costs are unknown, which allows the algorithm to focus on minimizing mc(x)
- Both of these costs are to be minimized, allowing to eliminate the skew towards majority class, while reducing the cost of used features

Direct Approach with Cost-Sensitive Splitting

T = inducted decision tree
S = training set
$x \in S$ = selected training instance
B(x) = set of features describing instance x

- Split considering the following metric:

$$ATC(S) = \frac{\sum_{x \in S} (tc(x) + mc(x))}{|S|}.$$

ATC = Average Total Cost = takes into account both the cost associated with misclassified instances (mc(x)) and features used by them (tc(x))

S′ or S′(l) = the set of instances in a given leaf node l

$$tc(x) = tc\left(B'(x)\right) = \sum_{a \in S'} tc(a).$$

tc(x) = the test cost associated with x using the subset of features B′(x) that are used by path from the root of T to one of its leaves to which x belongs

Direct Approach with Cost-Sensitive Splitting

T = inducted decision tree
S = training set
$x \in S$ = selected training instance
B(x) = set of features describing instance x

- Split considering the following metric:

$$ATC(S) = \frac{\sum_{x \in S} (tc(x) + mc(x))}{|S|}.$$

ATC = Average Total Cost

$$mc(x) = \begin{cases} C(i, j) & \text{if } d_t(x) = i \text{ and } d_c(x) = j, \\ C(j, i) & \text{if } d_t(x) = j \text{ and } d_c(x) = i. \end{cases}$$

mc(x) = the misclassification cost = computed for each instance x in S′ that has different true class label ($d_t(x)$) than the label associated with this node ($d_c(x)$) = remember that each leaf node l is associated with only a single class label (next slide)

Direct Approach with Cost-Sensitive Splitting

T = inducted decision tree
S = training set
x ∈ S = selected training instance
B(x) = set of features describing instance x

- Split considering the following metric:

$$ATC(S) = \frac{\sum_{x \in S} (tc(x) + mc(x))}{|S|}.$$

ATC = Average Total Cost

$|S_i'(l)|$ and $|S_j'(l)|$ are the numbers of instances from i^{th} and j^{th} class in this l^{th} node

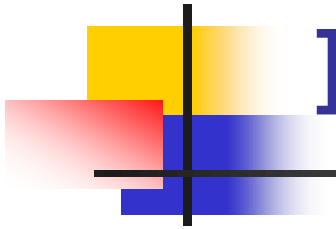
Then for any $x \in S'(l)$ the assigned class is calculated as:

$$d_c(x) = \begin{cases} i\text{-th class} & \text{if } |S_j'(l)| \times C(j,i), > |S_i'(l)| \times C(i,j). \\ j\text{-th class} & \text{if } |S_i'(l)| \times C(i,j). > |S_j'(l)| \times C(j,i), \end{cases}$$

Cost-Sensitive Decision Trees

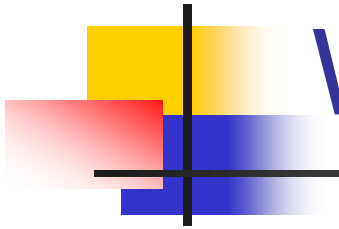
- The two most important approaches to cost-sensitive decision tree induction are:
  - Direct Approach with Cost-Sensitive Splitting (splitting criterion modification)
  - Meta-learning Approach with Instance Weighting (instance weighting)
    - https://sci2s.ugr.es/keel/pdf/algorithm/articulo/2002%20-%20Ting%20-%20An%20instance-weighting%20method%20to%20induce%20cost-sensitive%20trees%20-%20IEEETKDE.pdf

Meta-learning Approach with Instance Weighting



- An alternative solution to using the cost directly when creating splits lies in weighting the training instances
- Higher weights is assigned to instances coming from the class with higher value of misclassification cost
- This is done instead of sampling procedures, thus the size of the training set is not altered

Meta-learning Approach with Instance Weighting



- The following process allows for modifying any decision tree induction scheme to take into account misclassification costs expressed as weighted instances

Meta-learning Approach with Instance Weighting

The probability that an instance is in class j given that it falls into node t is given by the ratio of the total number of class j instances to the total number of instances in this node.

Entropia

$$-\sum_j p(j|t) log[p(j|t)].$$

$$p(j|t) = \frac{N_j(t)}{\sum_i N_i(t)}.$$

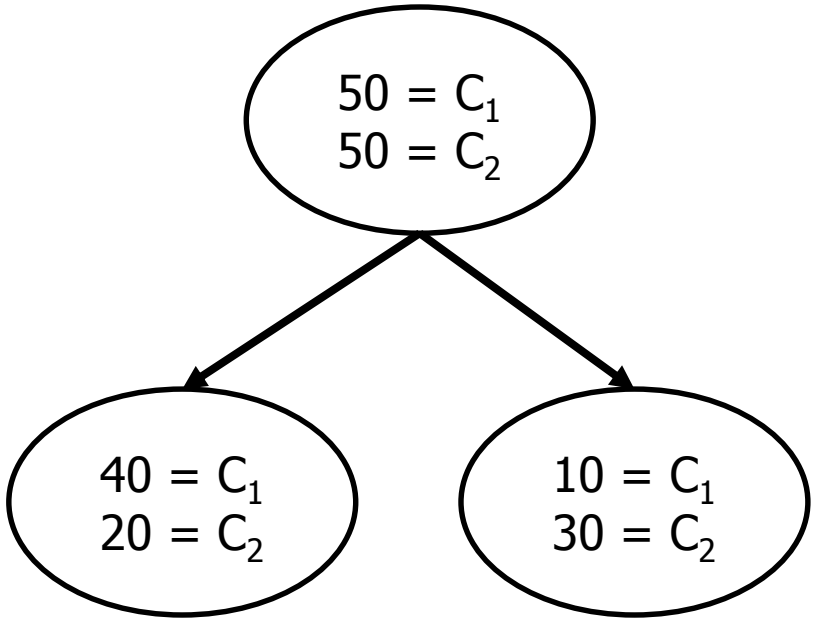
$$p_w(j|t) = \frac{W_j(t)}{\sum_i W_i(t)} = \frac{w(j)N_j(t)}{\sum_i w(i)N_i(t)}.$$

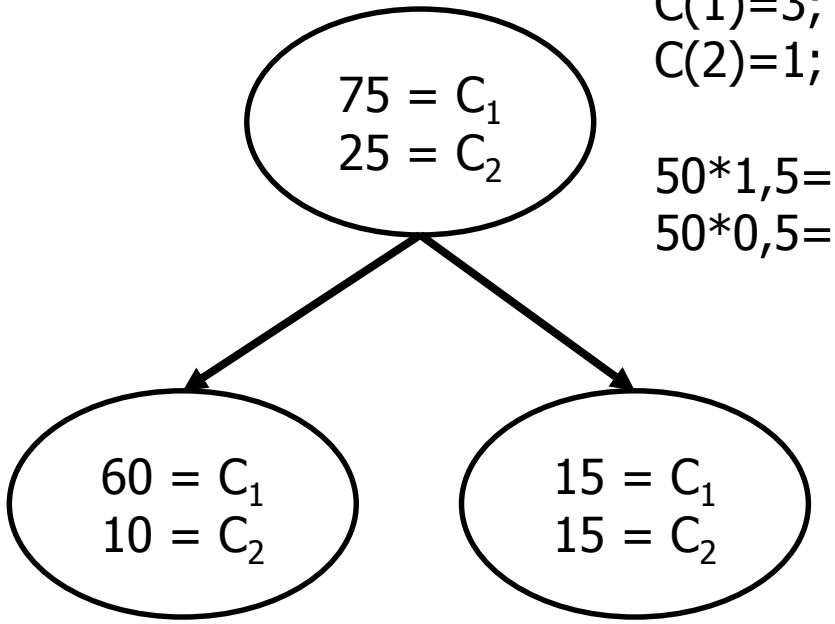
Trocar

$$w(j) = C(j)\frac{N}{\sum_i C(i)N_i}$$

This modification effectively converts the standard tree induction procedure that seeks to minimize *the number of errors*, regardless of cost, to a procedure that seeks to minimize *the number of errors with high weight or cost*. Note that minimizing the

Meta-learning Approach with Instance Weighting



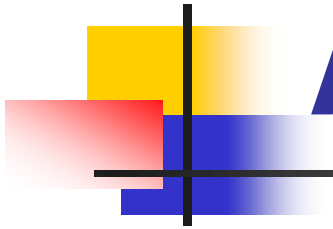


C(1)=3; w(1)=1,5
C(2)=1; w(2)=0,5

50*1,5=75
50*0,5=25

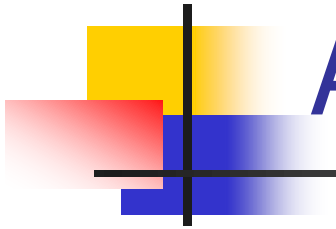
w(1) = C(1) * (100 / (C(1)*50 + C(2)*50))
w(1) = 3 * (100 / (3*50 + 1*50))
w(1) = 1,5

Observe that no algorithm-level modifications are required



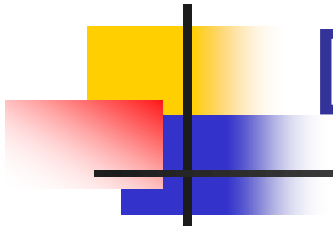
Algorithm-Level Approaches

- Algorithm-level methods concentrate on modifying existing learners to alleviate their bias towards majority class instead on altering the supplied training set



Algorithm-Level Approaches

- Algorithm-level solutions are not as popular in the literature, as they are arguably more difficult to design and implement than pre-processing methods
  - Many of popular machine learning algorithms have been subject to such alternations, including SVMS, Decision Trees, NN approaches, Bayesian classifiers, ANNs, etc.



Decision Trees

- There is a number of modifications available, most of which concentrate on cost-sensitive solutions
- From the algorithm-level point of view the most straightforward approach lies in modifying their key component – split function

Decision Trees

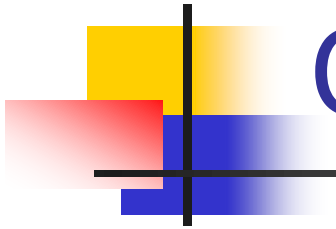
- The most efficient solution lies in using Hellinger distance for creating splits

$$d_H(TPR, FPR) = \sqrt{\left(\sqrt{TPR} - \sqrt{FPR}\right)^2 + \left(\sqrt{1 - TPR} - \sqrt{1 - FPR}\right)^2}.$$

Hellinger distance for binary imbalanced domain

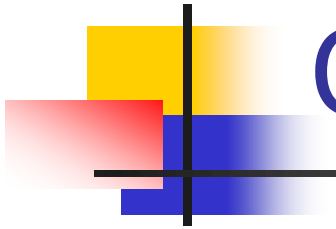
Limitation of Hellinger distance lies in its binary nature, as it cannot be directly applied to multi-class imbalanced problems

ROC = TPR x FPR



One-Class Classifiers (OCC)

- Designed for scenarios in which during the training step we have access only to objects coming from a single class
- A one-class classifier aims at capturing characteristics of training instances, in order to be able to distinguish between them and potential outliers to appear



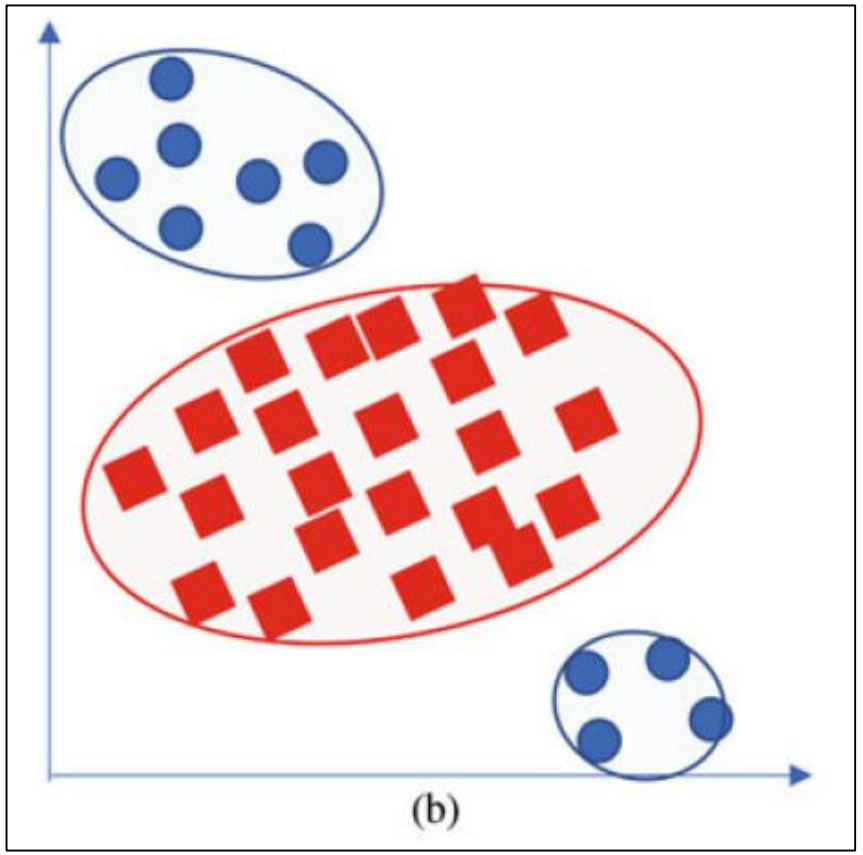
One-Class Classifiers (OCC)

- The idea is to train a one-class classifier on the better represented class and treat the remaining one as outliers
  - The majority class will be treated as the target concept and minority class as outliers
  - This solution has proven to be especially useful when the minority class lack any structure, being predominantly composed of small disjuncts or noisy instances

One-Class Classifiers (OCC)

- It is common that the "concept" beneath a class is split into several sub-concepts
- Small disjuncts = the models are divided in several pieces, and a concept class is represented as disjunction of different pieces for each class (divide-and-conquer and set covering approaches)

The presence of sub-concepts (and small disjunct) may accentuate the problem of class imbalance, as different levels of imbalance may exist for each sub-concept



Evaluating Performance with Class Imbalance

- **True Positive Rate** (**TPR**) = **Sensitivity** = **Recall** = fraction of positive instances correctly predicted by the classifier

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}.$$

- **True Negative Rate** (**TNR**) = **Specificity** = fraction of negative instances correctly predicted by the classifier

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}}.$$

Evaluating Performance with Class Imbalance

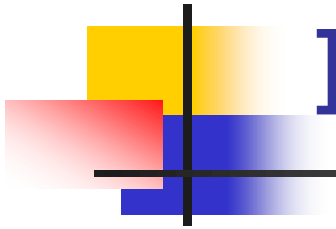
- **False Positive Rate** (**FPR**) = 1 − TNR

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}.$$

- **False Negative Rate** (**FNR**) = 1 − TPR

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}}.$$

Evaluating Performance with Class Imbalance



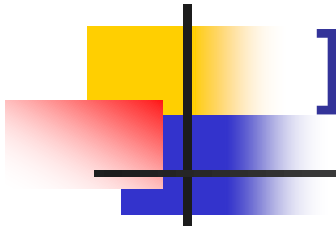
- The evaluation measures previously defined do not take into account the skew among the classes, which can be formally defined as $\alpha = P/(P + N)$
  - Changing the relative numbers of P and N will have no effect on TPR, TNR, FPR, or FNR, since they depend only on the fraction of correct classifications for every class, independently of the other class

Evaluating Performance with Class Imbalance

- **Precision** = fraction of correct predictions of the positive class over the total number of positive predictions (sensitive to the skew)

$$\text{Precision, } p = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Evaluating Performance with Class Imbalance



- The measures previously defined provide an incomplete representation of performance
  - They either only capture the effect of false positives (e.g., FPR and precision) or the effect of false negatives (e.g., TPR or recall), but not both

Evaluating Performance with Class Imbalance

- If we optimize only one of these measures, we may end up with a classifier that shows low FN but high FP, or vice-versa

$$\text{Positive Likelihood Ratio} = \frac{\text{TPR}}{\text{FPR}}.$$

$$F_1 \text{ measure} = \frac{2rp}{r+p} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}.$$

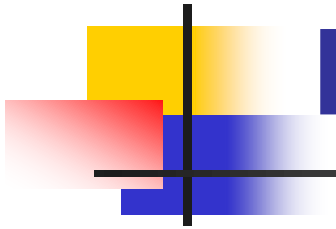
$$G \text{ measure} = \sqrt{rp} = \frac{\text{TP}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})}}.$$

$$G\text{-}mean = \sqrt{TPR \cdot TNR}$$

$$BAC = \frac{TPR + TNR}{2}$$

Pacotes

- Imbalanced-learn [https://imbalanced-learn.org/stable/index.html]
- An empirical comparison and evaluation of minority oversampling techniques on a large number of imbalanced datasets, 2019 (artigo)
  - Smote-variants: A python implementation of 85 minorityoversampling techniques, 2019 (artigo/pacote)
    - https://github.com/analyticalmindsltd/smote_variants
- UBL: An Implementation of Re-Sampling Approaches to Utility-Based Learning for Both Classification and Regression Tasks [https://cran.r-project.org/web/packages/UBL/index.html]



Datasets

- KEEL-dataset repositor [https://sci2s.ugr.es/keel/datasets.php]
  - Imbalanced data sets [https://sci2s.ugr.es/keel/imbalanced.php]