# Class imbalance problem and cost-sensitive learning

Katarzyna Macioszek, Ada Majchrzak

01

# Problem introduction

# Basic concepts

- Class imbalance – situation when the class distributions of data are highly imbalanced.
- Minority class – the class that makes up a smaller proportion in the dataset.
- Majority class – the class that makes up a larger proportion in the dataset.
- In our case study: minority class = heart disease (1), majority class = no heart disease (0).

# Cost matrix

- Cost matrix – tool used to evaluate and analyze the cost associated with missclassification of each class.
- Example for binary classification:

| | Actual negative | Actual positive |
|---|---|---|
| Predict negative | C(0,0) or TN | C(0,1) or FN |
| Predict positive | C(1,0) or FP | C(1,1) or TP |

Where

- C(i,i) – negated cost (benefit) when an instance is predicted correctly,
- C(i,j) – cost of misclassifying class *j* as class *i*.

# Cost-sensitive learning for class imbalance

- Common approach to solve the class imbalance problem.
- For binary classification: positive class (minority), negative class (majority).
- Usually bigger cost associated with positive class misclassification.
- Our case study: positive – heart disease, negative – no heart disease.

# Real-world applications

- Fraud detection
- Medical diagnosis
- Business decision-making

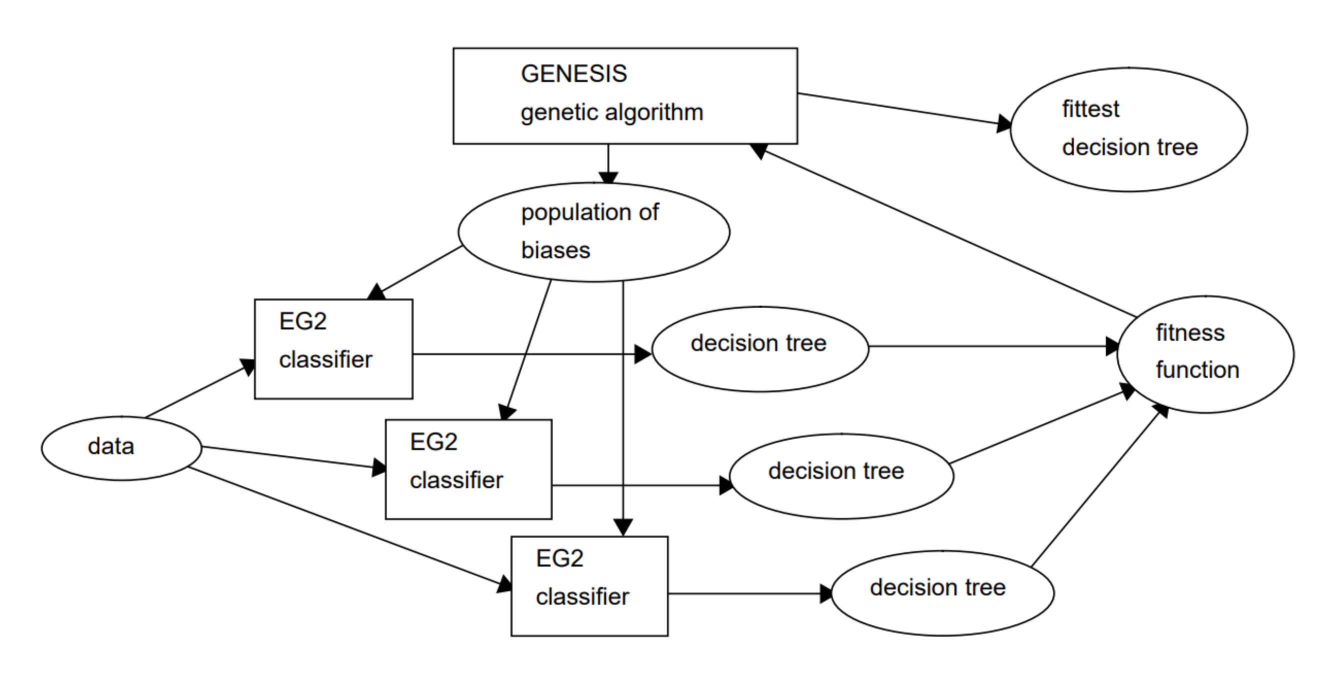
02

# Direct methods

# ICET algorithm

- To calculate the cost of a particular case, we follow its path down the decision tree (each split has a cost).
- If the case has been misclassified, the cost is added to the cost of splits.
- ICET is a hybrid of a genetic algorithm and a decision tree induction algorithm.
- The genetic algorithm evolves a population of biases for the decision tree induction algorithm .

# ICET algorithm




Source: [Turney 1995]

# Cost-sensitive decision trees


- Drummond, C., and Holte, R. 2000. *Exploiting the cost (in)sensitivity of decision tree splitting criteria*. In Proceedings of the 17th International Conference on Machine Learning, 239-246.
- Ling, C.X., Yang, Q., Wang, J., and Zhang, S. 2004. *Decision Trees with Minimal Costs*. In Proceedings of 2004 International Conference on Machine Learning (ICML'2004).

03

# Meta-learning

# Sampling

# Cost-proportionate instance weighting

- We want to alter the original instance distribution by introducing weights (fractions) proportional to the relative missclassification cost for each instance.
- Transparent box: supply the weights directly to the classifier – can't be applied to all classifiers, gives good results.
- Black box: resample according to the weights – can be applied to all classifiers, often leads to overfitting.

# Cost-proportionate rejection sampling

- Cost-proportionate instance weighting black box approach, but instead of classical sampling with replacement, we employ so-called rejection sampling.
- We draw from distribution $D$ with domain $X \times Y \times C$, where $X$ – classifier input space, $Y$ – binary output, $C$ – misclassification cost of given instance.
- The goal: learn a classifier minimizing the expected cost $E_{x,y,c}{\sim}D[\mathrm{C}\ \mathbb{I}_{h(x)\neq y}]$.
- Instead of cost matrix, we use one number per instance.

# Cost-proportionate rejection sampling

1. We sample from $D$.
2. We keep the sample with probability $\frac{c}{Z}$, where $Z$ is a constant satysfying $max_{(x,y,c)\in S} C \leq Z$, where $S$ – training set.
3. We repeat as many times as there are samples and we obtain a new training set $S'$.

# Costing

- Cost-proportionate rejection sampling with aggregation.
- CPRS produces a different training set each time, and each time it is quite small.
- We can take advantage of that by producing an ensemble classifier.

# Costing

Costing(learner $A$, sample set $S$, count $t$)

1. For $i = 1{:}\, t$

   - $S'$ = rejection sample from $S$ with acceptance probability $\frac{c}{Z}$

   - $h_i = A(S')$

2. Return $h(x) = sign(\sum_{i=1}^{t} h_i(x))$

# Instance weighting – cost-sensitive trees

- $N$ – number of instances in training set, $N_i/N_j$ - number of instances of class $i$/j in training set, $C(i)/C(j)$ – cost of misclassifying an instance of class $i/j$.
- The weight of class $j$ instance: $w(j) = C(j)\frac{N}{\sum_i C(i)N_i}$, where $\sum_j w(j)N_j = N$.
- To use above formula, we need to convert cost matrix to cost vector.
- We use the standard decision tree procedure, but instead of $N_j(t)$ – number of instances at given node, we use $W_j(t) = w(j)N_j(t)$ when computing the test selection criterion.

# Instance weighting – cost-sensitive trees

- Instead of minimizing the number of errors, we minimize the number of errors with high cost (greater than 1).
- As a result, usually the number of low-cost errors is increased.
- This method can be regarded as sampling because the instances with $w(j) > 1$ can be viewed as instance duplication.

# Instance weighting – cost-sensitive trees

- C4.5 model already uses $W_j(t)$ instead of $N_j(t)$, to create a cost-sensitive C4.5 we need to properly initialize weights (formula from previous slide).
- According to experiments conducted in [Ting 1998], the CS C4.5 performes better than C4.5 for binary classification, but comparably in case of multiple classes (due to problematic cost matrix -> cost vector conversion).

# Thresholding

# Basic concepts for thresholding

- The expected cost of classifying an instance x into class j:

  $R(i|x) = \sum_j P(j|x)C(i,j)$

- In binary case the threshold $p^*$obtained from cost matrix:

  $$p^* = \frac{C(1,0)}{C(1,0)+C(0,1)} = \frac{FP}{FP+FN} \,.$$

# MetaCost

- The algorithm first uses bagging on decision trees to obtain reliable probability estimations for training instances.
- The training examples are relabeled according to threshold $p^*$.
- After that a cost-insensitive classifier is built for relabeled instances to produce predictions for test instances.

# CostSensitiveClassifier

- If a cost-insensitive classifier outputs probabilities associated with each instance, the CSC can predict classes with the smallest expected misclassification cost.
- In binary case the threshold $p^*$ is used for the classifier to classify instance x to positive class if $P(1|x) \geq p^*$.
- Drawback: we need a cost-insensitive classifier that produces accurate posterior probability estimations.

# Empirical thresholding

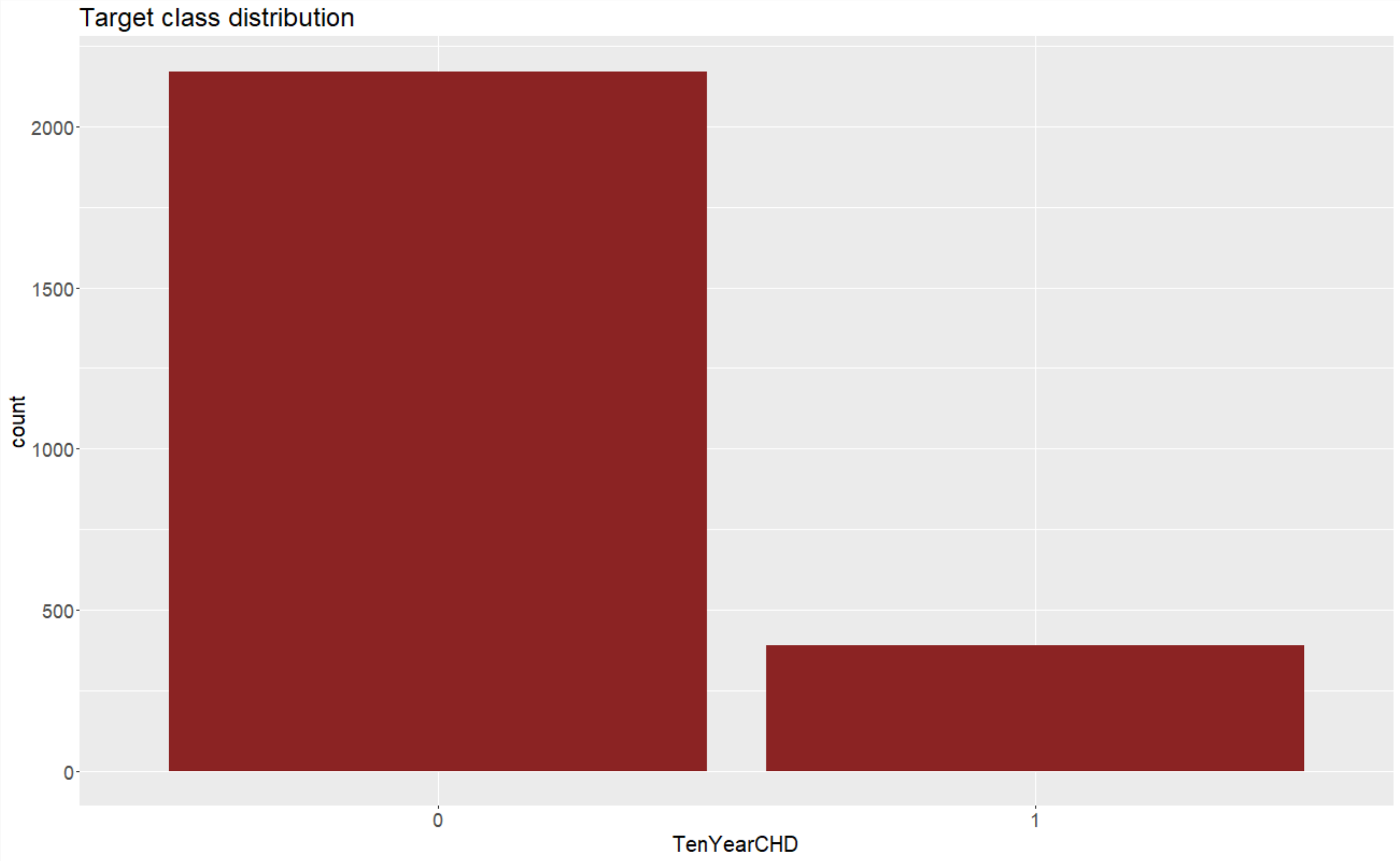
- Does not require accurate probability estimations.
- The total misclassification cost is a function of threshold $M_C = f(p^*)$.
- The algorithm calculates $M_C$ only for probability estimates.
- Empirical threshold, that minimizes the total misclassification cost, is used to predict class labels of test instances.
- To avoid overfitting, an m-fold cross-validation is applied, and the best threshold is chosen using validation set.

04

# Case study

# Framingham dataset

- Target class: TenYearCHD (Congenital Heart Deffect)
- Minority – 1 (sick), majority – 0 (healthy)



Target class distribution
2000
1500
1000
500
0
count
0
1
TenYearCHD

# Thresholding algorithms

CostSensitiveClassifier implementation using mlr package in R:

```
In [44]: classification_task <- makeClassifTask(data = training, target = "TenYearCHD", positive = 1)
         lrn <- makeLearner("classif.rpart", predict.type = "prob")
         mod <- train(lrn, classification_task)
         pred <- predict(mod, newdata = test)
         pred_threshold <- setThreshold(pred, threshold)
         calculateConfusionMatrix(pred_threshold)

...

In [46]: # 5-fold cross-validation performance for threshold = 0.17
         classification_cost <- makeCostMeasure(id = "class_cost", name = "Classification cost", cost = costs, best = 0, worst = 5)
         rin <- makeResampleInstance("CV", iters = 5, task = classification_task)
         lrn <- makeLearner("classif.rpart", predict.type = "prob", predict.threshold = threshold)#, trace=FALSE)
         r <- resample(lrn, classification_task, resampling = rin, measures = list(classification_cost, mmce), show.info = FALSE)
         r

         Resample Result
         Task: training
         Learner: classif.rpart
         Aggr perf: class_cost.test.mean=1.5163298,mmce.test.mean=0.3532238
         Runtime: 0.197485
```

# CSC results

| Original results | | | |
|---|---|---|---|
| | | Predicted | |
| | | 1 | 0 |
| Actual | 1 | 534 | 8 |
| | 0 | 83 | 14 |

| $p^* = 0.33$ | | | |
|---|---|---|---|
| | | Predicted | |
| | | 1 | 0 |
| Actual | 1 | 534 | 8 |
| | 0 | 83 | 14 |

| $p^* = 0.17$ | | | |
|---|---|---|---|
| | | Predicted | |
| | | 1 | 0 |
| Actual | 1 | 316 | 226 |
| | 0 | 37 | 60 |

| $p^* = 0.091$ | | | |
|---|---|---|---|
| | | Predicted | |
| | | 1 | 0 |
| Actual | 1 | 310 | 232 |
| | 0 | 35 | 35 |

# Thresholding algorithms

Empirical Thresholding implementation using mlr package in R:

```
fram.costs <- makeCostMeasure(id = "fram.costs", name = "Framigham costs", costs = costs,
  best = 0, worst = 5)
fram.task <- makeClassifTask(data = training, target = "TenYearCHD")

lrn <- makeLearner("classif.rpart", predict.type = "prob")
rin <- makeResampleInstance("CV", iters = 5, task = fram.task)
r <- resample(lrn, fram.task, rin, measures = list(fram.costs, mmce), show.info = FALSE)
tuned_res <- tuneThreshold(pred = r$pred)
pred_emp_threshold <- setThreshold(pred, tuned_res$th)
```

```
round(tuned_res$th, 5)
```

0.12048

# Empirical thresholding results

```
round(performance(pred_emp_threshold, measures = list(fram.costs, mmce)), 5)
```

**fram.costs:** 1.87011 **mmce:** 0.41784

| Results for ET | | | |
|---|---|---|---|
| | | Predicted | |
| | | 1 | 0 |
| Actual | 1 | 310 | 232 |
| | 0 | 35 | 62 |

# Rejection sampling

Implementation using caret and costsensitive packages in R

```
weights <- ifelse(training$TenYearCHD == 1, 0.7, 0.3)
weights2 <- ifelse(training$TenYearCHD == 1, 0.95, 0.05)
classifier <- caret::train
X_train <- training[, c(-9)]
y_train <- training$TenYearCHD
X_test <- test[, c(-9)]
y_test <- test$TenYearCHD
```

```
knn_rs <- cost.proportionate.classifier(X_train, y_train, weights, classifier, method = 'knn', trControl=knn_control, tuneGrid=knn_grid)
knn_pred_rs <- predict(knn_rs, X_test, aggregation = 'weighted', type = 'prob', output_type='class')
```

```
knn_rs2 <- cost.proportionate.classifier(X_train, y_train, weights2, classifier, method = 'knn', trControl=knn_control, tuneGrid=knn_grid)
knn_pred_rs2 <- predict(knn_rs2, X_test, aggregation = 'weighted', type = 'prob', output_type='class')
```

# Rejection sampling results

| Basic KNN | | |
|---|---|---|
| | Reference | |
| Prediction | 0 | 1 |
| 0 | 539 | 90 |
| 1 | 3 | 7 |

Accuracy: 0.85
Sensitivity: 0.07
Specificity: 0.99

| $c_1 = 0.7, c_0 = 0.3$ | | |
|---|---|---|
| | Reference | |
| Prediction | 0 | 1 |
| 0 | 502 | 77 |
| 1 | 40 | 20 |

Accuracy: 0.82
Sensitivity: 0.20
Specificity: 0.93

| $c_1 = 0.95, c_0 = 0.05$ | | |
|---|---|---|
| | Reference | |
| Prediction | 0 | 1 |
| 0 | 56 | 5 |
| 1 | 486 | 92 |

Accuracy: 0.23
Sensitivity: 0.95
Specificity: 0.10

# Thank you for your attention

# Sources


- **[Ling 2010]** Ling, Charles & Sheng, Victor. (2010). *Cost-Sensitive Learning and the Class Imbalance Problem.*
- **[Turney 1995]** Turney, P.D. 1995. *Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm*.
- **[Zadrozny 2003]** Zadrozny, B., Langford, J., and Abe, N. 2003. *Cost-sensitive learning by Cost-Proportionate instance Weighting*.
- **[Ting 1998]** Ting, K.M. 1998. *Inducing Cost-Sensitive Trees via Instance Weighting*.
- **[Witten & Frank 2005]** Witten, I.H., and Frank, E. 2005. *Data Mining – Practical Machine Learning Tools andTechniques with Java Implementations.*
- **[Sheng & Ling 2006]** Sheng, V.S. and Ling, C.X. 2006. *Thresholding for Making Classifiers Cost-sensitive. In Proceedings of the 21st National Conference on Artificial Intelligence.*
- **[Domingos 1999]** Domingos, P. 1999. *MetaCost: A general method for making classifiers cost-sensitive.*