Pemodelan Klasifikasi pada Data dengan Kelas Tidak Seimbang (Imbalance-Class Data)

Apa itu Data dengan Kelas Tak Seimbang?

- Data dengan kelas tidak seimbang merujuk pada situasi dimana keberadaan amatan dari masing-masing kelas timpang jumlahnya.

- Sebagai contoh, kita barangkali memiliki 100 buah amatan dimana kelas pertama sebanyak 800 amatan dan kelas kedua sebanyak 200 amatan, atau dengan rasio 4:1. Situasi lain dapat saja terjadi dengan ketimpangan yang jauh lebih tinggi.

- Meskipun pendekatan-pendekatan yang akan dibahas dapat juga diterapkan pada kasus multiclass, diskusi hanya dibatasi pada kasus dua kelas saja.

Ketakseimbangan adalah masalah yang umum ditemui

- Data dengan kelas yang tidak seimbang jumlahnya merupakan masalah yang umum dijumpai.
- Bahkan seringkali memang diharapkan demikian. Seperti pada kasus analisis fraud. Kejadian “fraud”merupakan kejadian yang sangat jarang sehingga proporsi nasabah yang masuk dalam kelas ini akan sangat kecil. Termasuk di dalamnya kejadian “default” pada nasabah kredit.
- Ilustrasi lain adalah pada data “churn” dimana pelanggan yang churn sangatlah sedikit.
- Kelas yang memiliki proporsi yang sedikit disebut sebagai kelas “minoritas”, sedangkan kelas yang proporsinya dominan disebut kelas “mayoritas”.

Accuracy Paradox

- Bayangkan kita punya data dimana perbandingan banyaknya amatan antara kelas 0 dan kelas 1 adalah 95:5

- Jika kita memperoleh model, dan dugaan dari model tersebut menghasilkan prediksi kelas 0 untuk semua amatan.

- Akurasinya 95%....

- Tapi model itu gagal memprediksi dengan benar satupun amatan dari kelas minoritas.

Beberapa trik…

- https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

1) Kumpulkan lebih banyak data

- You might think it’s silly, but collecting more data is almost always overlooked.

- Can you collect more data? Take a second and think about whether you are able to gather more data on your problem.

- Pada kasus pembuatan approval credit scoring… ada proses yang dikenal sebagai reject inference. Melibatkan data calon nasabah yang di-reject sebagai data training dengan terlebih dahulu memberikan label Bad/Good kepadanya.

2) Resampling Your Dataset

- You can change the dataset that you use to build your predictive model to have more balanced data.
- This change is called sampling your dataset and there are two main methods that you can use to even-up the classes:
  - You can add copies of instances from the under-represented class called **over-sampling** (or more formally sampling with replacement), or
  - You can delete instances from the over-represented class, called u**nder-sampling**.
- These approaches are often very easy to implement and fast to run. They are an excellent starting point.

- **Some Rules of Thumb**
  - Consider testing under-sampling when you have an a lot data (tens- or hundreds of thousands of instances or more)
  - Consider testing over-sampling when you don't have a lot of data (tens of thousands of records or less)
  - Consider testing random and non-random (e.g. stratified) sampling schemes.
  - Consider testing different resampled ratios (e.g. you don't have to target a 1:1 ratio in a binary classification problem, try other ratios)

3) Try Generate Synthetic Samples

- A simple way to generate synthetic samples is to randomly sample the attributes from instances in the minority class.
- There are systematic algorithms that you can use to generate synthetic samples. The most popular of such algorithms is called SMOTE or the Synthetic Minority Over-sampling Technique.
- As its name suggests, SMOTE is an oversampling method. It works by creating synthetic samples from the minor class instead of creating copies. The algorithm selects two or more similar instances (using a distance measure) and perturbing an instance one attribute at a time by a random amount within the difference to the neighboring instances.

4) Try Different Algorithms

- As always, I strongly advice you to not use your favorite algorithm on every problem. You should at least be spot-checking a variety of different types of algorithms on a given problem.
- That being said, decision trees often perform well on imbalanced datasets. The splitting rules that look at the class variable used in the creation of the trees, can force both classes to be addressed.
- If in doubt, try a few popular decision tree algorithms like C4.5, C5.0, CART, and Random Forest.

5) Try Penalized Models

- You can use the same algorithms but give them a different perspective on the problem.
- Penalized classification imposes an additional cost on the model for making classification mistakes on the minority class during training. These penalties can bias the model to pay more attention to the minority class.
- Often the handling of class penalties or weights are specialized to the learning algorithm. There are penalized versions of algorithms such as penalized-SVM and penalized-LDA.
- Using penalization is desirable if you are locked into a specific algorithm and are unable to resample or you're getting poor results. It provides yet another way to "balance" the classes. Setting up the penalty matrix can be complex. You will very likely have to try a variety of penalty schemes and see what works best for your problem.

Random Under/Over-Sampling

- Random Undersampling
  - Gunakan semua amatan dari kelas minoritas
  - Gunakan sebagian amatan dari kelas mayoritas yang dipilih secara random
- Random Oversampling
  - Gunakan semua amatan dari kelas mayoritas
  - Lakukan duplikasi data amatan kelas mayoritas secara random (sampling with replacement)

Ilustrasi…

- Yang akan kita kerjakan
  - Ada data dengan nama “bank-additional-full.csv”
  - Berisi 41188 observasi dengan (12.27% yes, 88.73% no)
  - Data dibagi menjadi dua bagian: 70% training set, 30% testing set
  - Pemodelan TREE dikerjakan dengan tiga prosedur
    - Data training **tidak diseimbangkan** dan langsung dijalankan algoritma tree (rpart)
    - Dilakukan proses **undersampling** terhadap data training, kemudian dilanjutkan pemodelan tree (rpart)
    - Dilakukan proses **oversampling** terhadap data training, kemudian dilanjutkan pemodelan tree (rpart)
  - Model-model diterapkan pada data testing untuk dievaluasi ketepatan prediksinya

Ilustrasi…

```
#membaca data
cnth<-read.csv("D:/bank-additional-full.csv",sep=";")
library(caret)
library(rpart)

nrow(cnth) #melihat banyaknya observasi

#melihat distribusi kelas variabel target
table(cnth$y)
prop.table(table(cnth$y))
```

Ilustrasi...

```
> #membaca data
> cnth<-read.csv("D:/bank-additional-
full.csv",sep=";")
> library(caret)
> library(rpart)

> nrow(cnth) #melihat banyaknya observasi
[1] 41188

> #melihat distribusi kelas variabel target
> table(cnth$y)
no      yes
36548   4640
> prop.table(table(cnth$y))
no          yes
0.8873458   0.1126542
>
```

Ilustrasi…

```
#mempartisi data
set.seed(100)
test_idx <- createDataPartition(cnth$y, p=0.3, list=FALSE)

cnth_tst <- cnth[test_idx,]  #membuat data testing
cnth_trn <- cnth[-test_idx,]  #membuat data training

nrow(cnth_trn) #banyaknya observasi data training
nrow(cnth_tst) #banyaknya observasi data testing
```

```
> nrow(cnth_trn) #banyaknya observasi data training
[1] 28831
> nrow(cnth_tst) #banyaknya observasi data testing
[1] 12357
```

Ilustrasi… Pemodelan tanpa Perlakuan terhadap Data Training


```
#pemodelan TREE pada data training
#tidak ada perlakuan apa-apa terhadap data training
tree_mod<-rpart(y~., data=cnth_trn, method="class")

#memprediksi data testing
prob_tree<-predict(tree_mod,cnth_tst)[,2]
pred_tree<-as.factor(ifelse(prob_tree<0.5,"no","yes"))

#mengevaluasi ketepatan prediksi dari model
eval.ori<-confusionMatrix(pred_tree, cnth_tst$y,
positive="yes")

eval.ori
```

Ilustrasi…hasil tanpa perlakuan

```
Confusion Matrix and Statistics

          Reference
Prediction    no   yes
       no  10559   688
       yes   406   704

               Accuracy : 0.9115
                 95% CI : (0.9063, 0.9164)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5142
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.50575
            Specificity : 0.96297
         Pos Pred Value : 0.63423
         Neg Pred Value : 0.93883
             Prevalence : 0.11265
         Detection Rate : 0.05697
   Detection Prevalence : 0.08983
      Balanced Accuracy : 0.73436

       'Positive' Class : yes
```

Ilustrasi… undersampling

```
#melakukan undersampling
Set.seed(200)
down_train <- downSample(x = cnth_trn[, -ncol(cnth_trn)],y
= cnth_trn$y)

#distribusi yes-no
table(cnth_trn$y) #sebelum undersampling
table(down_train$Class) #setelah undersampling
```

```
> table(cnth_trn$y) #sebelum undersampling

no     yes
25583  3248
> table(down_train$Class) #setelah undersampling

no     yes
3248   3248
```

Ilustrasi... undersampling

```
#pemodelan TREE menggunakan data hasil undersampling
tree_under_mod<-rpart(Class~., data=down_train, method="class")

#memprediksi data testing
prob_tree_under<-predict(tree_under_mod,cnth_tst)[,2]
pred_tree_under<-as.factor(ifelse(prob_tree_under<0.5,"no","yes"))

#mengevaluasi ketepatan prediksi dari model
eval.under<-confusionMatrix(pred_tree_under, cnth_tst$y,
positive="yes")
eval.under
```

Ilustrasi...perbandingan hasil...

Model dengan data undersampling

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  8982  147
       yes 1983 1245

               Accuracy : 0.8276
                 95% CI : (0.8209, 0.8343)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4528
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8944
            Specificity : 0.8192
         Pos Pred Value : 0.3857
         Neg Pred Value : 0.9839
             Prevalence : 0.1126
         Detection Rate : 0.1008
   Detection Prevalence : 0.2612
      Balanced Accuracy : 0.8568

       'Positive' Class : yes
```

Model dengan data tanpa perlakuan

```
Confusion Matrix and Statistics

          Reference
Prediction    no   yes
       no  10559   688
       yes   406   704

               Accuracy : 0.9115
                 95% CI : (0.9063, 0.9164)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5142
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.50575
            Specificity : 0.96297
         Pos Pred Value : 0.63423
         Neg Pred Value : 0.93883
             Prevalence : 0.11265
         Detection Rate : 0.05697
   Detection Prevalence : 0.08983
      Balanced Accuracy : 0.73436

       'Positive' Class : yes
```

Ilustrasi…


```
#melakukan oversampling
Set.seed(300)
up_train <- upSample(x = cnth_trn[, -ncol(cnth_trn)],y =
cnth_trn$y)

#distribusi yes-no
table(cnth_trn$y) #sebelum oversampling
table(up_train$Class) #setelah oversampling
```

```
> table(cnth_trn$y) #sebelum oversampling
no     yes
25583  3248

> table(up_train$Class) #setelah oversampling
no     yes
25583  25583
```

Ilustrasi…

```
#pemodelan TREE menggunakan data hasil oversampling
tree_over_mod<-rpart(Class~., data=up_train, method="class")

#memprediksi data testing
prob_tree_over<-predict(tree_over_mod,cnth_tst)[,2]
pred_tree_over<-as.factor(ifelse(prob_tree_over<0.5,"no","yes"))

#mengevaluasi ketepatan prediksi dari model
eval.over<-confusionMatrix(pred_tree_over, cnth_tst$y,
positive="yes")

eval.over
```

Ilustrasi…perbandingan hasil…

Model dengan data oversampling

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  8931  143
       yes 2034 1249

               Accuracy : 0.8238
                 95% CI : (0.817, 0.8305)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4468
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8973
            Specificity : 0.8145
         Pos Pred Value : 0.3804
         Neg Pred Value : 0.9842
             Prevalence : 0.1126
         Detection Rate : 0.1011
   Detection Prevalence : 0.2657
      Balanced Accuracy : 0.8559

       'Positive' Class : yes
```

Model dengan data tanpa perlakuan

```
Confusion Matrix and Statistics

          Reference
Prediction    no   yes
       no  10559   688
       yes   406   704

               Accuracy : 0.9115
                 95% CI : (0.9063, 0.9164)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5142
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.50575
            Specificity : 0.96297
         Pos Pred Value : 0.63423
         Neg Pred Value : 0.93883
             Prevalence : 0.11265
         Detection Rate : 0.05697
   Detection Prevalence : 0.08983
      Balanced Accuracy : 0.73436

       'Positive' Class : yes
```

Ilustrasi... perbandingan ketiganya...

```
akurasi <- c(eval.ori$overall[1],
             eval.under$overall[1],
             eval.over$overall[1])
sensitivitas <- c(eval.ori$byClass[1],
                  eval.under$byClass[1],
                  eval.over$byClass[1])
spesifisitas <- c(eval.ori$byClass[2],
                  eval.under$byClass[2],
                  eval.over$byClass[2])

hasil<-cbind(akurasi,sensitivitas,spesifisitas)

row.names(hasil)<-c("Original","Undersampling","Oversampling")

hasil
```

Ilustrasi... perbandingan ketiganya...

bagusco

```
> hasil
              akurasi sensitivitas spesifisitas
Original      0.9114672    0.5057471    0.9629731
Undersampling 0.8276281    0.8943966    0.8191518
Oversampling  0.8238246    0.8972701    0.8145007
```

SMOTE

- singkatan dari Synthetic Minority Oversampling Technique.
- Dikembangkan oleh Chawla, Hall, & Kegelmeyer (2002).
- Membuat data sintetis untuk kelas minoritas, sehingga banyaknya amatan kelas minoritas menjadi lebih banyak.
- Dalam perkembangannya, tidak hanya men-generate amatan minoritas, tapi juga melakukan Random Undersampling terhadap amatan kelas mayoritas.
- Banyak studi empiris yang menunjukkan bahwa teknik SMOTE efektif dalam proses menghasilkan model klasifikasi yang baik.

SMOTE – prosedur

- untuk setiap amatan kelas minoritas
  - cari k tetangga terdekat yang juga merupakan amatan kelas minoritas
  - pilih secara acak j amatan dari k amatan di atas (nilai j tergantung pada banyaknya oversampling yang diinginkan)
  - generate amatan buatan baru yang terletak pada garis lurus antara amatan minoritas dengan tetangga yang terpilih.
  - Randomly generate synthetic samples along the lines joining the minority sample and its j selected neighbours

SMOTE – prosedur

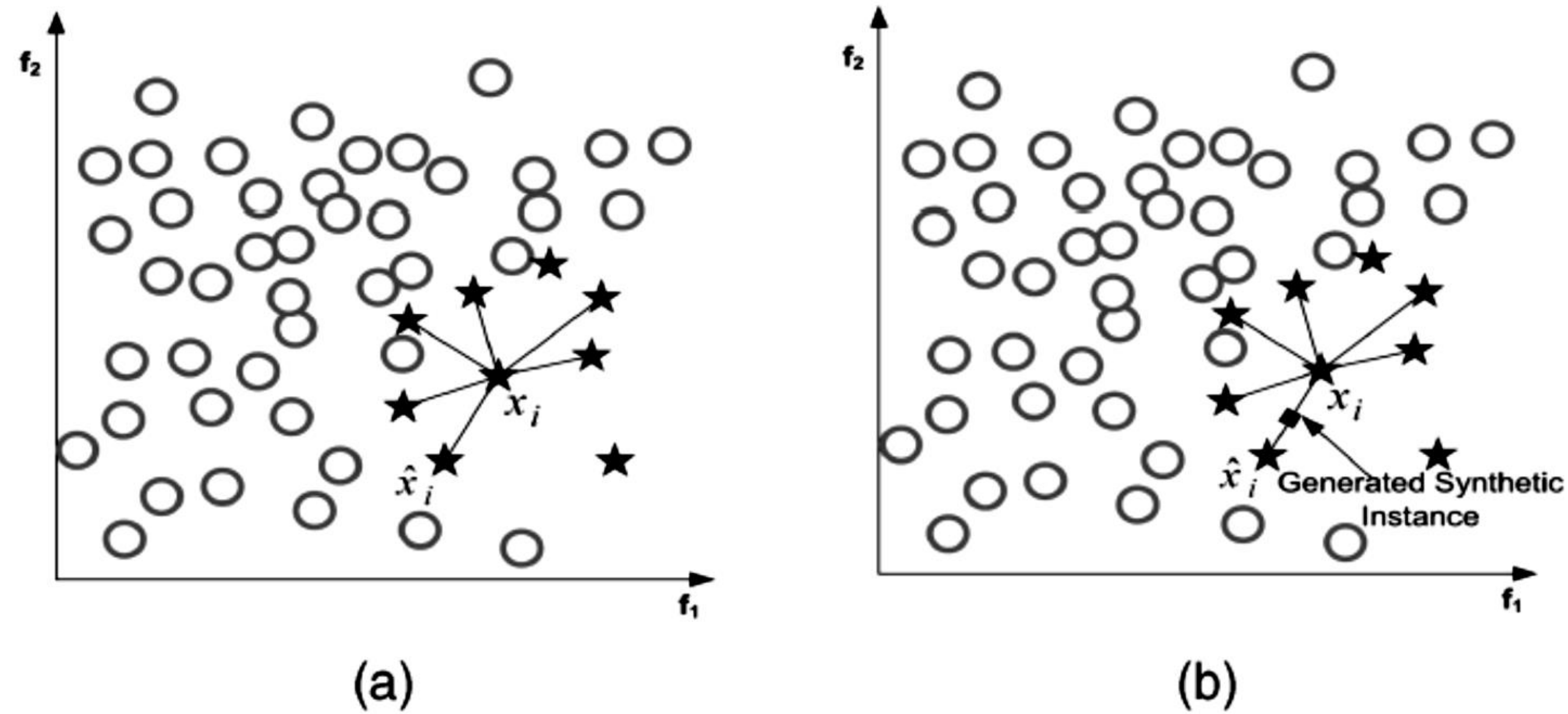


Fig. 3. (a) Example of the K-nearest neighbors for the x_i example under consideration ($K = 6$). (b) Data creation based on euclidian distance.

Ilustrasi…

```
#menjalankan algoritma SMOTE
library(DMwR)
Set.seed(400)
smote_train <- SMOTE(y ~ ., data  = cnth_trn, k = 5,
                     perc.over=200, perc.under=200)

#distribusi yes-no
table(cnth_trn$y) #sebelum SMOTE
table(smote_train$y) #sesudah SMOTE
```

```
> table(cnth_trn$y) #sebelum SMOTE
no     yes
25583  3248

> table(smote_train$y) #sesudah SMOTE
no     yes
12992  9744
```

Ilustrasi…

```
#pemodelan TREE menggunakan data hasil SMOTE
tree_smote_mod<-rpart(y~., data=smote_train, method="class")

#memprediksi data testing
prob_tree_smote<-predict(tree_smote_mod,cnth_tst)[,2]
pred_tree_smote<-
as.factor(ifelse(prob_tree_smote<0.5,"no","yes"))

#mengevaluasi ketepatan prediksi dari model
eval.smote<-confusionMatrix(pred_tree_smote, cnth_tst$y,
positive="yes")

eval.smote
```

Ilustrasi...perbandingan hasil...

Model dengan data SMOTE

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  9956  353
       yes 1009 1039

               Accuracy : 0.8898
                 95% CI : (0.8841, 0.8952)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : 0.2009

                  Kappa : 0.5427
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.74641
            Specificity : 0.90798
         Pos Pred Value : 0.50732
         Neg Pred Value : 0.96576
             Prevalence : 0.11265
         Detection Rate : 0.08408
   Detection Prevalence : 0.16574
      Balanced Accuracy : 0.82719

       'Positive' Class : yes
```

Model dengan data tanpa perlakuan

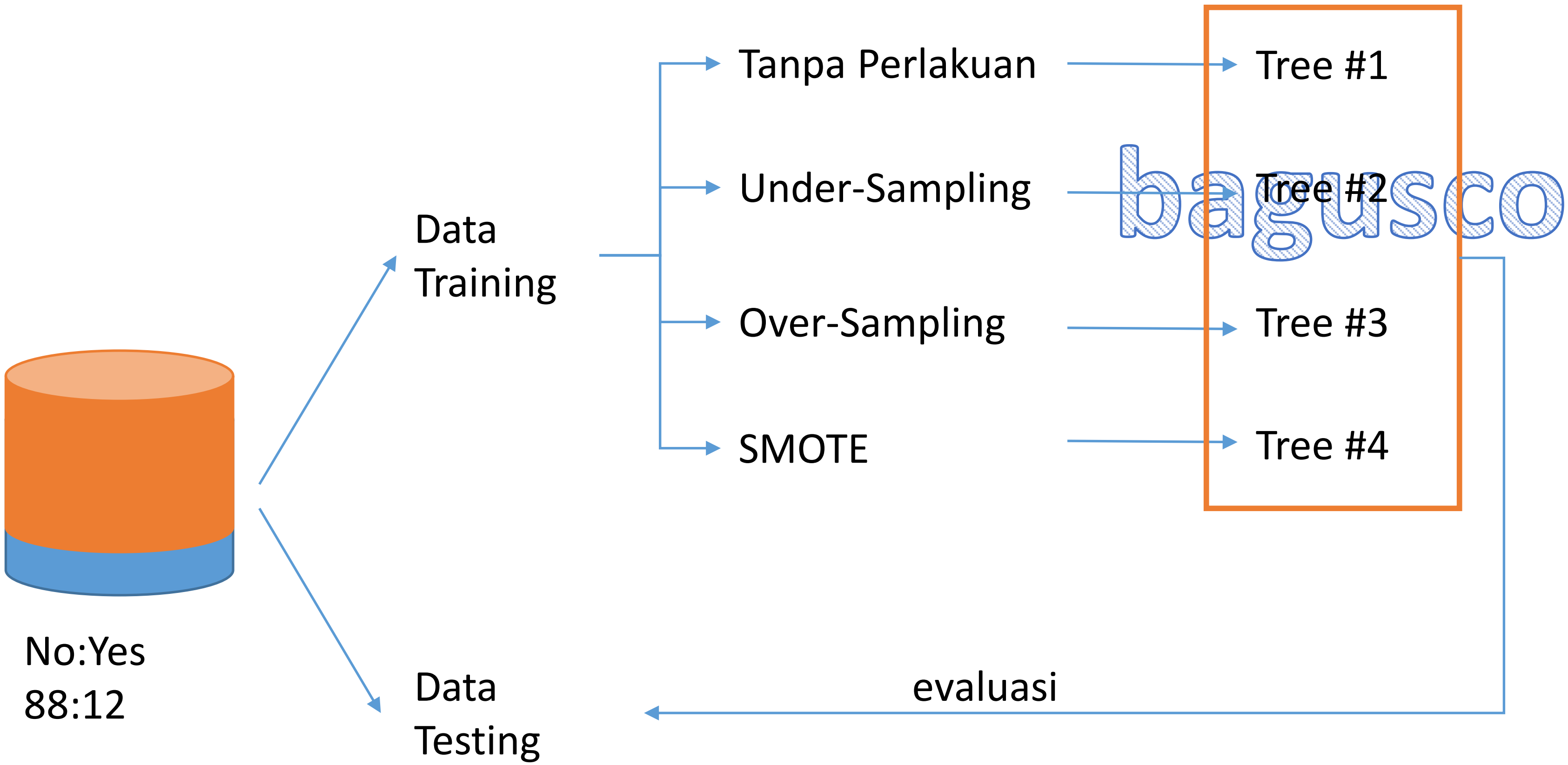
```
Confusion Matrix and Statistics

          Reference
Prediction    no   yes
       no  10559   688
       yes   406   704

               Accuracy : 0.9115
                 95% CI : (0.9063, 0.9164)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5142
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.50575
            Specificity : 0.96297
         Pos Pred Value : 0.63423
         Neg Pred Value : 0.93883
             Prevalence : 0.11265
         Detection Rate : 0.05697
   Detection Prevalence : 0.08983
      Balanced Accuracy : 0.73436

       'Positive' Class : yes
```

Ilustrasi… perbandingan dengan sebelumnya…

```
hasil<-rbind(hasil,c(eval.smote$overall[1],
                     eval.smote$byClass[1],
                     eval.smote$byClass[2]))
row.names(hasil)[4]<-"SMOTE"
hasil
```

```
> hasil
               akurasi sensitivitas spesifisitas
Original     0.9114672    0.5057471    0.9629731
Undersampling 0.8276281    0.8943966    0.8191518
Oversampling 0.8238246    0.8972701    0.8145007
SMOTE        0.8897791    0.7464080    0.9079799
```

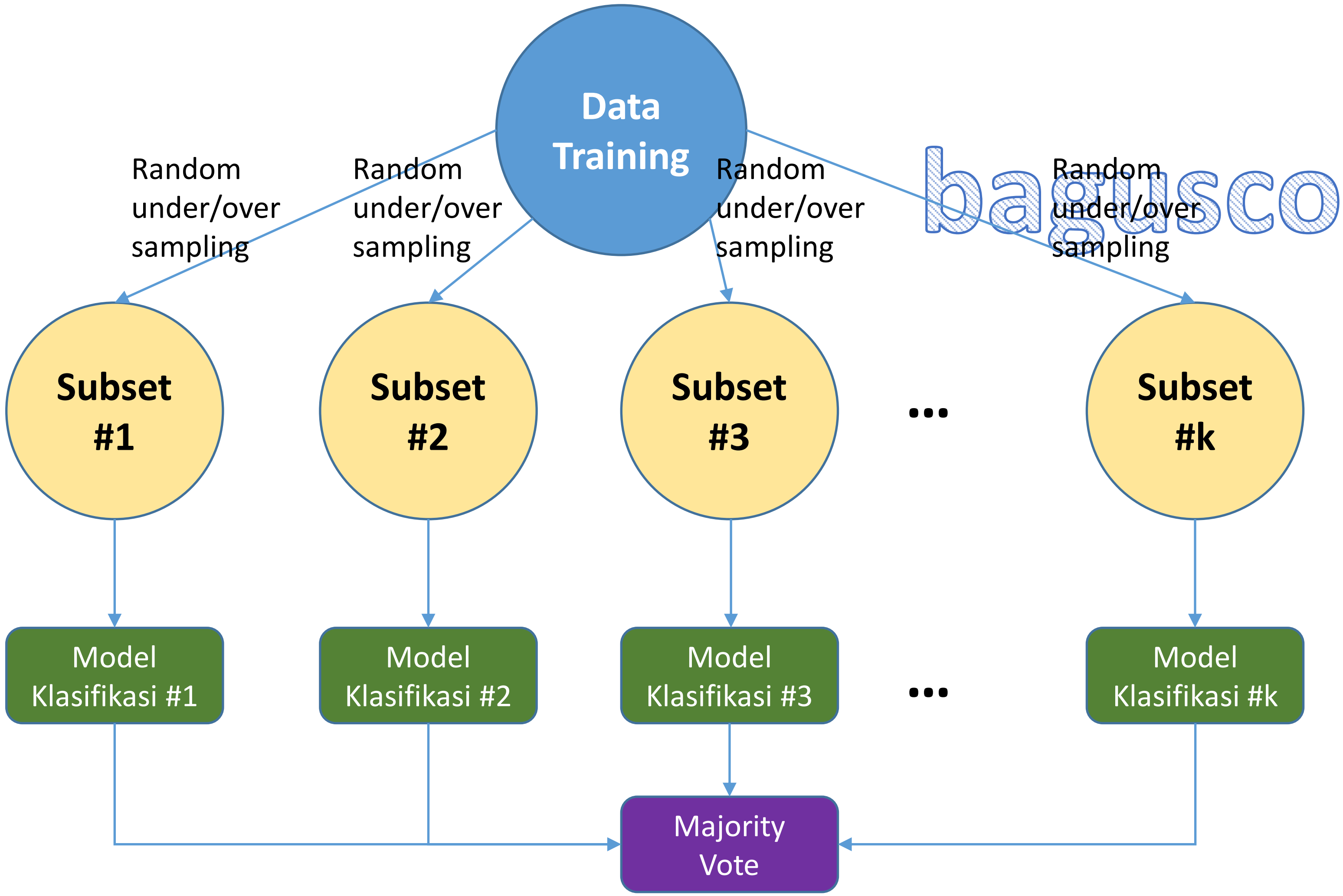
Yang sudah kita diskusikan



> hasil

	akurasi	sensitivitas	spesifisitas
Original	0.9114672	0.5057471	0.9629731
Undersampling	0.8276281	0.8943966	0.8191518
Oversampling	0.8238246	0.8972701	0.8145007
SMOTE	0.8897791	0.7464080	0.9079799

Under/Over-Bagging



Ilustrasi... UnderBagging


```
set.seed(500)
k<-100
pred_tree_under1<-matrix(NA,nrow(cnth_tst),k)
for(i in 1:k){
  down_train1 <- downSample(x = cnth_trn[, -ncol(cnth_trn)],y = cnth_trn$y)
  table(down_train1$Class)

  tree_under_mod1<-rpart(Class~., data=down_train1, method="class")
  prob_tree_under1<-predict(tree_under_mod1,cnth_tst)[,2]
  pred_tree_under1[,i]<-ifelse(prob_tree_under1<0.5,0,1)
}

pred_tree_under1a<-apply(pred_tree_under1,1,sum)
pred_tree_under1a<-as.factor(ifelse(pred_tree_under1a<k/2,"no","yes"))

eval.underbag<-confusionMatrix(pred_tree_under1a, cnth_tst$y, positive="yes")

eval.underbag
```

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  8920  133
       yes 2045 1259

               Accuracy : 0.8237
                 95% CI : (0.8169, 0.8304)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4488
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9045
            Specificity : 0.8135
         Pos Pred Value : 0.3811
         Neg Pred Value : 0.9853
             Prevalence : 0.1126
         Detection Rate : 0.1019
   Detection Prevalence : 0.2674
      Balanced Accuracy : 0.8590

       'Positive' Class : yes
```

Ilustrasi... OverBagging

bagusco

```
set.seed(600)
k<-100
pred_tree_over1<-matrix(NA,nrow(cnth_tst),k)
for(i in 1:k){
  up_train1 <- upSample(x = cnth_trn[, -ncol(cnth_trn)],y = cnth_trn$y)
  table(up_train1$Class)

  tree_over_mod1<-rpart(Class~., data=up_train1, method="class")
  prob_tree_over1<-predict(tree_over_mod1,cnth_tst)[,2]
  pred_tree_over1[,i]<-ifelse(prob_tree_over1<0.5,0,1)
}

pred_tree_over1a<-apply(pred_tree_over1,1,sum)
pred_tree_over1a<-as.factor(ifelse(pred_tree_over1a<k/2,"no","yes"))

eval.overbag<-confusionMatrix(pred_tree_over1a, cnth_tst$y, positive="yes")

eval.overbag
```

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  8951  146
       yes 2014 1246

               Accuracy : 0.8252
                 95% CI : (0.8184, 0.8319)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : 1

                  Kappa : 0.4486
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8951
            Specificity : 0.8163
         Pos Pred Value : 0.3822
         Neg Pred Value : 0.9840
             Prevalence : 0.1126
         Detection Rate : 0.1008
   Detection Prevalence : 0.2638
      Balanced Accuracy : 0.8557

       'Positive' Class : yes
```

```
hasil<-rbind(hasil,c(eval.underbag$overall[1],
                     eval.underbag$byClass[1],
                     eval.underbag$byClass[2]),
             c(eval.overbag$overall[1],
               eval.overbag$byClass[1],
               eval.overbag$byClass[2]))
row.names(hasil)[5:6]<-c("Underbagging","Overbagging")
hasil
```



```
> hasil
                akurasi sensitivitas spesifisitas
Original      0.9114672    0.5057471    0.9629731
Undersampling 0.8276281    0.8943966    0.8191518
Oversampling  0.8238246    0.8972701    0.8145007
SMOTE         0.8897791    0.7464080    0.9079799
Underbagging  0.8237436    0.9044540    0.8134975
Overbagging   0.8252003    0.8951149    0.8163247
```

RUS-Boost

Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., & Napolitano, A. (2010). RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, *40*(1), 185-197.

Algorithm RUSBoost

Given: Set S of examples $(x_1, y_1), ..., (x_m, y_m)$ with minority class $y^r \in Y$, $|Y| = 2$

Weak learner, $WeakLearn$

Number of iterations, T

Desired percentage of total instances to be represented by the minority class, N

1 Initialize $D_1(i) = \frac{1}{m}$ for all i.

2 Do for $t = 1, 2, ..., T$

a Create temporary training dataset S'_t with distribution D'_t using random undersampling

b Call $WeakLearn$, providing it with examples S'_t and their weights D'_t.

c Get back a hypothesis $h_t : X \times Y \rightarrow [0, 1]$.

d Calculate the pseudo-loss (for S and D_t):

$$\epsilon_t = \sum_{(i,y):y_i \neq y} D_t(i)(1 - h_t(x_i, y_i) + h_t(x_i, y)).$$

e Calculate the weight update parameter:

$$\alpha_t = \frac{\epsilon_t}{1 - \epsilon_t}.$$

f Update D_t:

$$D_{t+1}(i) = D_t(i)\alpha_t^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_i,y:y\neq y_i))}.$$

g Normalize D_{t+1}: Let $Z_t = \sum_i D_{t+1}(i)$.

$$D_{t+1}(i) = \frac{D_{t+1}(i)}{Z_t}.$$

3 Output the final hypothesis:

$$H(x) = \underset{y \in Y}{\text{argmax}} \sum_{t=1}^{T} h_t(x, y) \log \frac{1}{\alpha_t}.$$

```
library(ebmc)
set.seed(700)
train_rus<-cnth_trn
test_rus<-cnth_tst
train_rus$y <- factor(train_rus$y, levels = c("no", "yes"), labels = c("0", 
"1"))
test_rus$y <- factor(test_rus$y, levels = c("no", "yes"), labels = c("0", 
"1"))

rus_boost_mod <- rus(y ~ ., data = train_rus, size = 10, 
                 alg = "c50", ir = 1)

prob_rus_boost<-predict(rus_boost_mod, newdata=test_rus, type="prob")
pred_rus_boost<-as.factor(ifelse(prob_rus_boost<0.5,"no","yes"))
eval.rusboost<-confusionMatrix(pred_rus_boost, cnth_tst$y, positive="yes")
eval.rusboost
```

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  9353  123
       yes 1612 1269

               Accuracy : 0.8596
                 95% CI : (0.8533, 0.8657)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5212
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9116
            Specificity : 0.8530
         Pos Pred Value : 0.4405
         Neg Pred Value : 0.9870
             Prevalence : 0.1126
         Detection Rate : 0.1027
   Detection Prevalence : 0.2331
      Balanced Accuracy : 0.8823

       'Positive' Class : yes
```

```
hasil<-rbind(hasil,c(eval.rusboost$overall[1],
                     eval.rusboost$byClass[1],
                     eval.rusboost$byClass[2]))
row.names(hasil)[7]<-c("RUSBoost")
hasil
```

```
> hasil
               akurasi sensitivitas spesifisitas
Original     0.9114672    0.5057471    0.9629731
Undersampling 0.8276281   0.8943966    0.8191518
Oversampling 0.8238246    0.8972701    0.8145007
SMOTE        0.8897791    0.7464080    0.9079799
Underbagging 0.8237436    0.9044540    0.8134975
Overbagging  0.8252003    0.8951149    0.8163247
RUSBoost     0.8595938    0.9116379    0.8529868
```

EasyEnsemble

Algorithm 1 The `EasyEnsemble` algorithm.

1: {Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from $\mathcal{N}$, and s_i, the number of iterations to train an AdaBoost ensemble H_i}

2: $i \Leftarrow 0$

3: **repeat**

4: $\quad i \Leftarrow i + 1$

5: $\quad$ Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.

6: $\quad$ Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i, i.e.

$$H_i(x) = \operatorname{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right).$$

7: **until** $i = T$

8: Output: An ensemble:

$$H(x) = \operatorname{sgn}\left(\sum_{i=1}^{T} \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right).$$

Liu, X. Y., Wu, J., & Zhou, Z. H. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(2), 539-550.

```
library(ada)
library(ROSE)
set.seed(800)
easy_mod <- NULL
for (z in 1:10){
  easy_trn <- ovun.sample(y~.,data=cnth_trn,method="under")$data
  easy_mod[[z]] <- ada(y~.,data=easy_trn,type="discrete")
  print(z)
}

pred_iter <- matrix(NA,nrow(cnth_tst),length(easy_mod))
for (z in 1:length(easy_mod)){
  pred_iter[,z] <- predict(easy_mod[[z]],cnth_tst, type="F")
}
#pred_iter1 <- ifelse(pred_iter==1,-1,1)
pred_easy <- sign(apply(pred_iter,1,sum))
pred_easy <- as.factor(ifelse(pred_easy==-1,"no","yes"))
eval.easy <- confusionMatrix(pred_easy,cnth_tst$y,positive="yes")
eval.easy
```

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  9277   99
       yes 1688 1293

               Accuracy : 0.8554
                 95% CI : (0.8491, 0.8615)
    No Information Rate : 0.8874
    P-Value [Acc > NIR] : 1

                  Kappa : 0.5172
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9289
            Specificity : 0.8461
         Pos Pred Value : 0.4337
         Neg Pred Value : 0.9894
             Prevalence : 0.1126
         Detection Rate : 0.1046
   Detection Prevalence : 0.2412
      Balanced Accuracy : 0.8875

       'Positive' Class : yes
```

```
hasil<-rbind(hasil,c(eval.easy$overall[1],
                     eval.easy$byClass[1],
                     eval.easy$byClass[2]))
row.names(hasil)[8]<-c("easyEnsemble")
hasil
```



```
> hasil
```

	akurasi	sensitivitas	spesifisitas
Original	0.9114672	0.5057471	0.9629731
Undersampling	0.8276281	0.8943966	0.8191518
Oversampling	0.8238246	0.8972701	0.8145007
SMOTE	0.8897791	0.7464080	0.9079799
Underbagging	0.8237436	0.9044540	0.8134975
Overbagging	0.8252003	0.8951149	0.8163247
RUSBoost	0.8595938	0.9116379	0.8529868
easyEnsemble	0.8553856	0.9288793	0.8460556

BalanceCascade

Algorithm 2 The `BalanceCascade` algorithm.

{Input: A set of minority class examples $\mathcal{P}$, a set of majority class examples $\mathcal{N}$, $|\mathcal{P}| < |\mathcal{N}|$, the number of subsets T to sample from $\mathcal{N}$, and s_i, the number of iterations to train an AdaBoost ensemble H_i}

$i \Leftarrow 0$, $f \Leftarrow \sqrt[T-1]{\frac{|\mathcal{P}|}{|\mathcal{N}|}}$, f is the false positive rate (the error rate of misclassifying a majority class example to the minority class) that H_i should achieve.

repeat

  $i \Leftarrow i + 1$

  Randomly sample a subset $\mathcal{N}_i$ from $\mathcal{N}$, $|\mathcal{N}_i| = |\mathcal{P}|$.

  Learn H_i using $\mathcal{P}$ and $\mathcal{N}_i$. H_i is an AdaBoost ensemble with s_i weak classifiers $h_{i,j}$ and corresponding weights $\alpha_{i,j}$. The ensemble's threshold is θ_i i.e.
  $H_i(x) = \mathrm{sgn}\left(\sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \theta_i\right)$.

  Adjust θ_i such that H_i's false positive rate is f.

  Remove from $\mathcal{N}$ all examples that are correctly classified by H_i.

until $i = T$

Output: A single ensemble:
$H(x) = \mathrm{sgn}\left(\sum_{i=1}^{T} \sum_{j=1}^{s_i} \alpha_{i,j} h_{i,j}(x) - \sum_{i=1}^{T} \theta_i\right)$.

Liu, X. Y., Wu, J., & Zhou, Z. H. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(2), 539-550.