ITERA

Modul 6 Praktikum Statistika Sains Data

Teknik Resampling

Program Studi Sains Data
Fakultas Sains
Institut Teknologi Sumatera

2024

A. Tujuan Praktikum

1. Mahasiswa mampu memahami konsep teknik resampling
2. Mahasiswa mampu mengimplementasi Teknik resampling

B. Teori Dasar

Klasifikasi data yang tidak seimbang (*imbalanced classification)* adalah suatu masalah klasifikasi dimana distribusi kelas target memiliki rasio berbeda jauh. Kelas yang mengampil proporsi terbesar pada data tersebut disebut dengan **kelas mayoritas** dan sebaliknya yang mengambil proporsi terkecil disebut dengan **kelas minoritas**.

Perbedaan rasio kelas target bisa saja berbeda minimal ataupun ekstrim, seperti yang dikategorikan oleh developer google berikut dibawah:

Degree of imbalance	Proportion of Minority Class
Mild	20-40% of the data set
Moderate	1-20% of the data set
Extreme	<1% of the data set

Adapun alasan kenapa data bisa tidak seimbang :

- *Biased Sampling,* ketika pengambilan data terdapat bias atau memberatkan sebelah pihak.
- *Measurements Errors,* ketika terdapat masalah pengukuran saat pengambilan data.

Beberapa contoh masalah data yang tidak seimbang bisa ditemukan di beberapa kasus berikut :

1. *Fraud detection*
2. *Spam detection*
3. *Loan approval detection,* dan lain-lain.

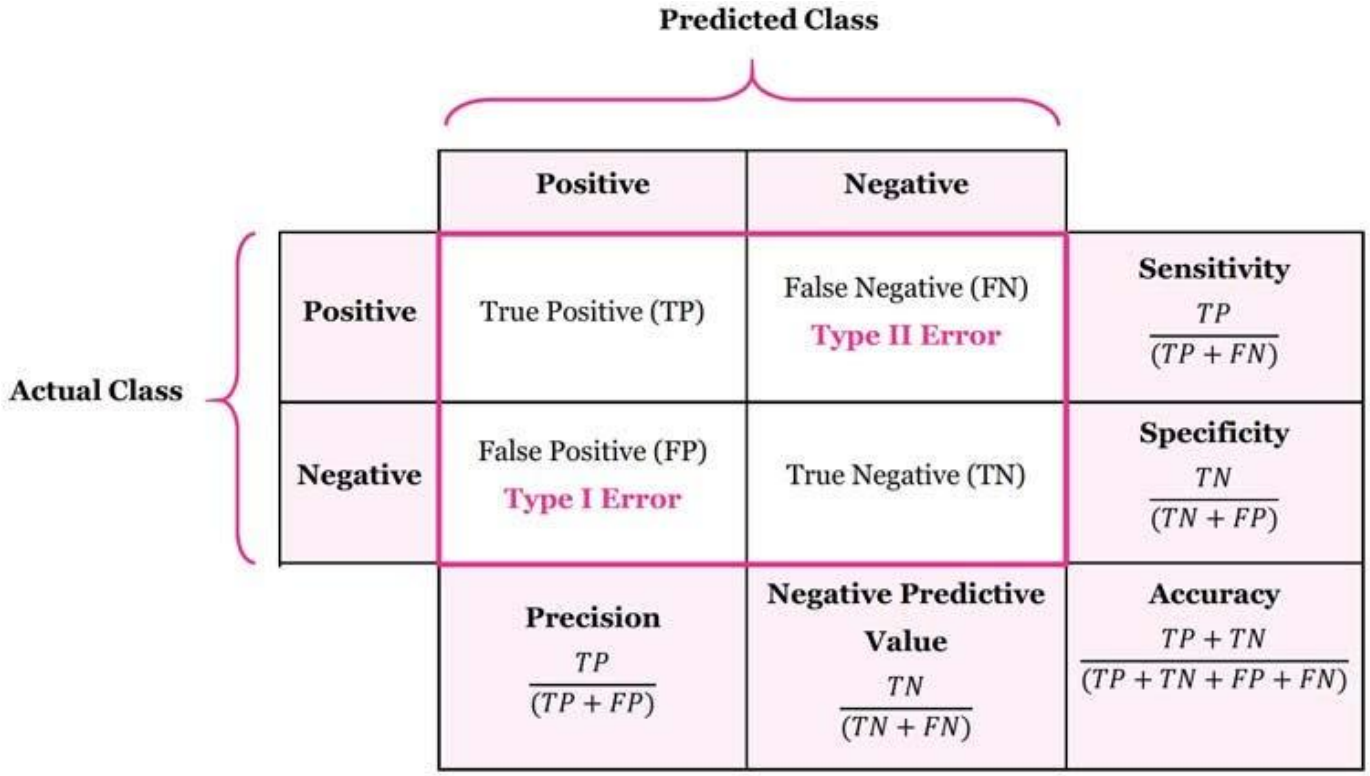
Alasan kenapa kita harus peduli apabila data kita tidak seimbang adalah karena metrik yang diberikan ketika kita mengukur performa model memberi interpretasi yang menyesatkan. Seperti contohnya ketika kita menggunakan metrik akurasi untuk mengukur performa model. Metrik akurasi merupakan metrik yang meringkas performa model klasifikasi secara keseluruhan dengan cara menghitung total prediksi benar dibagi oleh total semua prediksi. Metrik akurasi merupakan metrik yang sering digunakan karena mudah dipahami, tetapi metrik ini bisa menyesatkan kita apabila datanya tidak seimbang.

Kesalahan ini saking seringnya dilakukan oleh pemula sehingga ada nama spesial untuk kesalahan interpretasi ini, disebut "*The Accuracy Paradox*".

The Accuracy Paradox

The accuracy paradox is the paradoxical finding that accuracy is not a good metric for predictive models when classifying in predictive analysis. This is because a simple model may have a high level of accuracy but be too crude to be useful -Wikipedia

Akan diberikan contoh kasusnya, pertama perhatikan *confusion matrix* berikut :



Confusion matrix beserta rumus metrik lainnya

Dimisalkan ada masalah klasifikasi dengan data yang tidak seimbang dengan rasio kelas 1:99 dari total 100 baris data. Yang dimana artinya setiap 1 kelas minoritas mempunyai 99 kelas mayoritas.

Lalu dimisalkan, kelas minoritas adalah kelas-0 dan kelas mayoritas ada kelas-1.

Ketika kita menggunakan data tersebut sebagai training ke model machine learning, hasil laporan akan memberikan nilai akurasi tinggi. Sebenarnya nilai akurasi yang dilaporkan memanglah benar, tetapi dapat menyesatkan kita apabila kita berpikir bahwa model yang kita latih mampu membedakan kelas-0 dan kelas-1 dengan baik dari nilai metrik tersebut. Perhatikan contoh confusion matrix berikut dari evaluasi model dengan data yang tidak seimbang :

		Predicted	
		Kelas-0	Kelas-1
Actual	Kelas-0	0	1
	Kelas-1	0	99

Confusion matrix dari contoh kasus

Dengan menggunakan persamaan akurasi, kita mendapatkan nilai 99% tetapi apabila kelas-0 adalah menjadi prioritas dari kasus ini maka akurasi tinggi dari model ini bukanlaha berarti apa-apa malah bisa menyesatkan bagi yang salah menafsirkannya.

Adapun metrik alternatif untuk mengukur performa model dengan data yang tidak seimbang :

Precision, Recall & F-score Metrics

Precision, *Recall* dan *F-score* merupakan metrik yang populer digunakan ketika data yang ditangani tidak seimbang.

- *Precision,* Memberikan ringkasan bahwa prediksi kelas positif sebenarnya kelas positif sesungguhnya.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

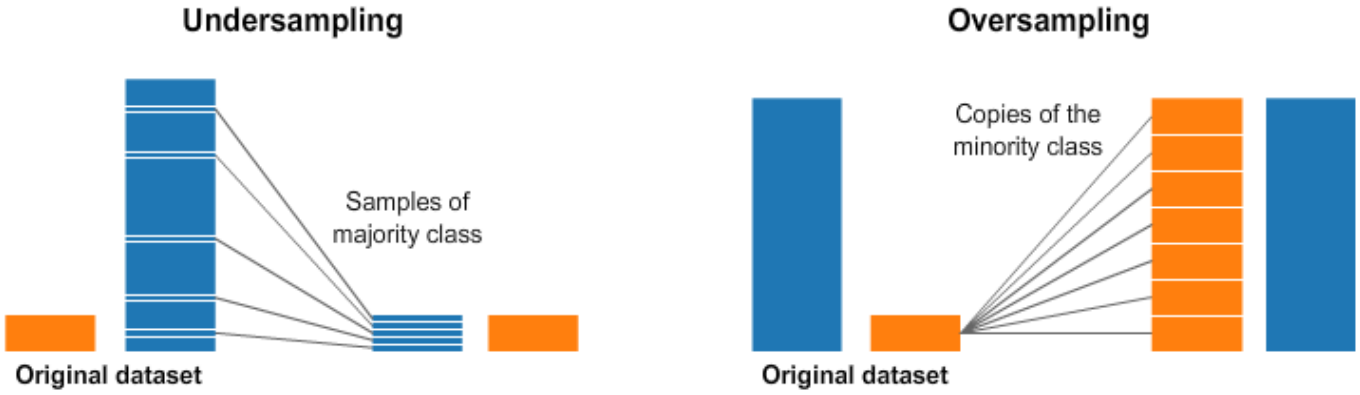
- *Recall,* Memberikan ringkasan bahwa seberapa baik prediksi kelas positif oleh model.

$$Recall = \frac{TP}{TP + FN};$$

- *F-score*, Merupakan metrik perpaduan antara *Precision* & *Recall* dan memberi nilai harmonis antara kedua nilai tersebut (Semakin tinggi, semakin baik).

$$F-score = 2.\frac{precision\ .\ recall}{precision + recall}$$

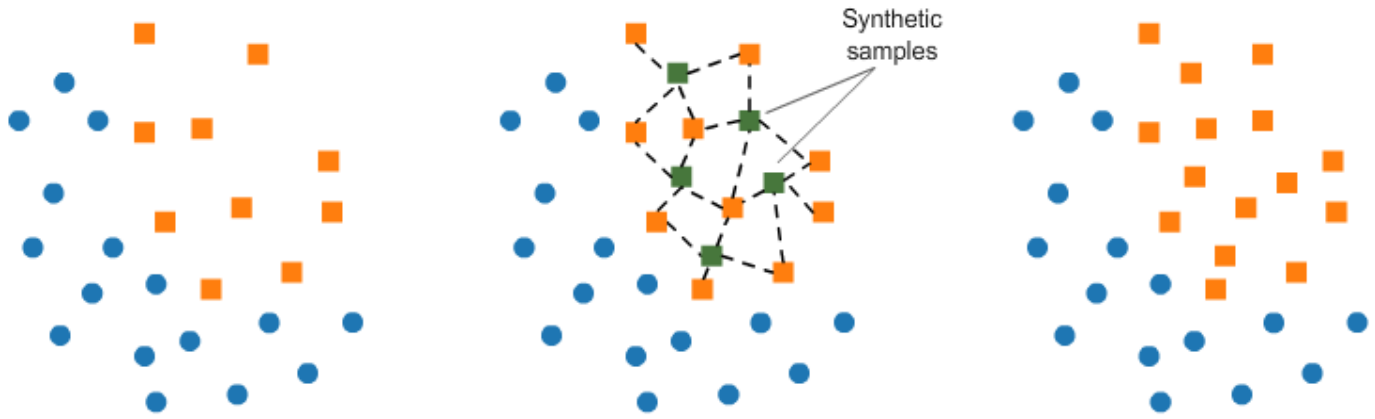
Terdapat beberapa metode untuk mengatasi data yang tidak seimbang untuk model klasifikasi seperti contohnya *weighted parameter* pada model, ensemble model dan cara yang terbaik adalah dengan membuat ulang pertanyaan dari masalah tersebut.
Tetapi pada artikel kali ini akan dibahas salah satu teknik populer untuk mengatasi data yang tidak seimbang, yaitu teknik **Teknik Resampling.**
Teknik Resampling merupakan suatu teknik atau metode untuk membuat sample baru dari sample atau populasi yang sudah ada pada data. Sederhananya, metode ini bisa dibagi jadi dua kategori : menghapus sampel dari kelas mayoritas sehingga rasio sama disebut **Undersampling** dan kedua, menambah sampel ke kelas minoritas sehingga rasio sama disebut **Oversampling.**



Visualisasi dari teknik Undersampling & Oversampling

Dari kedua metode tersebut terbagi menjadi banyak metode, di artikel ini akan dijelaskan dan di-implementasi beberapa saja.

1. **Undersampling**
   - Random Undersampling, Menghapus sampel kelas mayoritas secara acak.
   - Prototype Generation, Menghapus sampel dan menambah sampel berdasarkan metode clustering.
   - Near Miss, Menghapus sampel kelas mayoritas yang dekat dengan kumpulan data kelas minoritas.
   - Tomek's Link, Menghapus sampel kelas mayoritas yang berpasangan dengan datapoin kelas minoritas.
2. **Oversampling**
   - Random Oversampling, Menambah sampel kelas minoritas secara acak dari sampel yang sudah ada.
   - SMOTE (*Synthetic Minority Oversampling Technique*), Menambah sampel kelas minoritas dengan cara mensintesis data baru berdasarkan metode k-nearest neighbour.



Konsep bagaimana cara SMOTE bekerja

- ADASYN (*Adaptive synthetic*), Mensintesis data baru untuk kelas minoritas yang dimana mudah untuk dipelajari oleh model.

Mengenai metode atau teknik apa yang harus dipakai ketika mendapat masalah data tidak seimbang saat klasifikasi itu tergantung data yang kalian urusi masing-masing. Terkadang, menggunakan Random Sampling cukup menambah performa model tetapi terkadang juga tidak ada metode satupun yang bisa menambah performa model. Disaat inilah kalian harus menggunakan metode teknik lain selain resampling atau mungkin kalian bisa mempertanyakan ulang bagaimana cara kalian mem-framing masalahnya dan mengambil data baru yang memiliki rasio kelas yang cukup seimbang.

Implementasi Teknik Resampling

Implementasi teknik resampling untuk mengatasi data tidak seimbang akan menggunakan Python dan library imblearn. Alasannya karena bahasa pemrograman Python merupakan bahasa yang sangat mudah dipelajari dan dipahami yang dimana bagus untuk bekerja bersama tim dan alasan lainnya adalah bagus untuk melakukan riset akademik seperti untuk data science. Library imblearn merupakan library tambahan dari Python yang bisa di-install manual melalup package manager, library ini berguna untuk mengatasi data yang tidak seimbang sebelum di train untuk model. Pada implementasi ini akan diberikan cara implementasi SMOTE untuk mengatasi data yang tidak seimbang, dan dalam pengerjaan implementasi ini juga akan diberi tahu cara mengukur performa model sebelum dan sesudah menggunakan teknik Resampling.

Pertama sebelum memulai kalian perlu menginstall beberapa package python :

Instalasi package yang diperlukan

- Menggunakan pip

```
pip install matplotlib sklearn seaborn imbalanced-learn
```

- Menggunakan Anaconda

```
conda install matplotlib sklearn seaborn imbalanced-learn
```

Penulis juga menggunakan Jupyter Notebook sebagai editornya.

Kode Implementasi

Pertama, import package yang diperlukan.

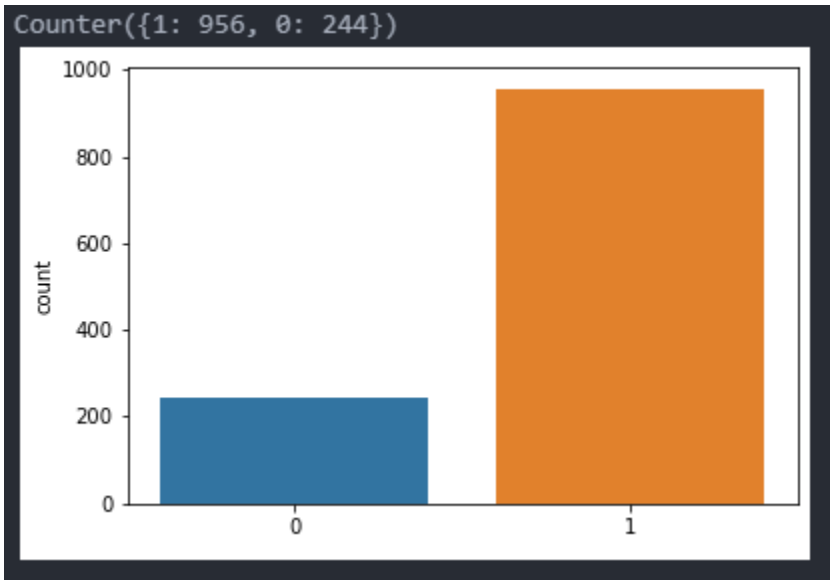
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from collections import Counter
```

Buat data blob fiktif menggunakan library sklearn lalu simpan ke X dan y.

```
X, y = make_classification(n_samples=1200, n_features=2, weights=[0.2, 0.8], n_redundant=0, n_informative=2, class_sep=0.25, n_clusters_per_class=1, random_state=13)
```

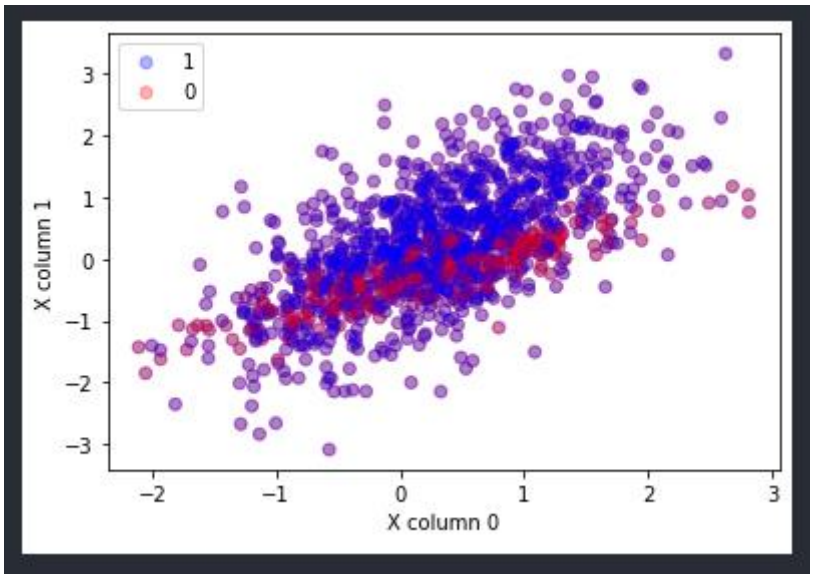
Output jumlah masing-masing kelas dan visualisasinya

```
print(Counter(y))
sns.countplot(x=y)
```



Visualisasi data X dan y pada scatter plot dengan warna dan opasitas kecil.

```
for i, color in enumerate(['1', '0']):
    plt.scatter(X[:, 0], X[:, 1], c=['b' if p == i else 'r' for p in y], alpha=0.3, label=color)plt.legend()
plt.xlabel('X column 0')
plt.ylabel('X column 1')
```



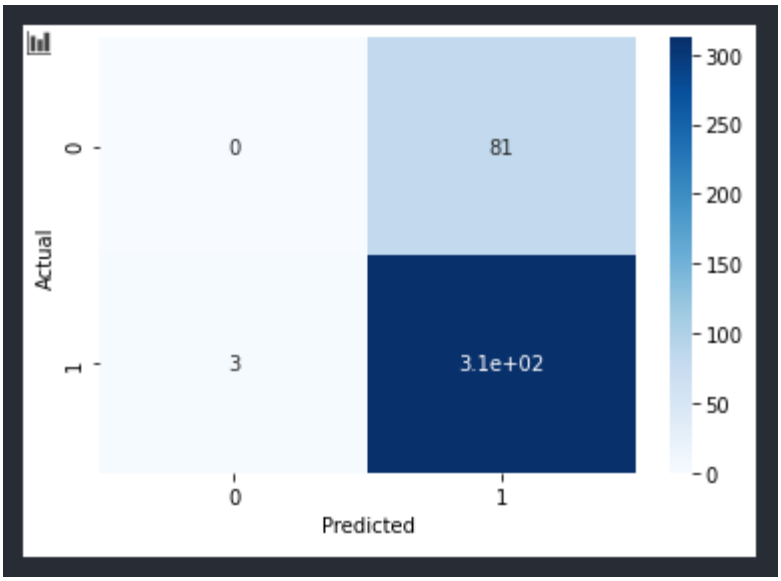
Bagi data untuk train dan test lalu latih model tanpa teknik resampling untuk melihat performa model.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y, random_state=13)clf = LogisticRegression(random_state=13)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('Accuracy is ', accuracy_score(y_test, y_pred))
```

Dari kode diatas, kita mendapat akurasi 78%, cukup lumayan tetapi hal ini berita bagus jikalau data kita seimbang, mari kita lihat confusion matrix dan classification reportnya.

```
print(classification_report(y_test, y_pred))cf_matrix = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cf_matrix, columns=['0', '1'], index=['0', '1'])
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
sns.heatmap(df_cm, cmap='Blues', annot=True)
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	81
1	0.79	0.99	0.88	315
accuracy			0.79	396
macro avg	0.40	0.50	0.44	396
weighted avg	0.63	0.79	0.70	396



Seperti yang dilihat, jika seandainya prioritas kita adalah model mampu memprediksi kelas-0 berarti model ini gagal total. Dari 3data kelas-0 tak ada satupun prediksi yang benar dan model salah melabel 81 data kelas-0 yang asli menjadi kelas-1. Jika dilihat F-scorenya pun untuk kelas-0 adalah nol, ini artinya sangatlah buruk. Harus ada penanganan data untuk meningkat performa model.
Mari kita coba terapkan salah satu teknik Resampling yaitu teknik Over Sampling metode SMOTE.

```
sm = SMOTE(random_state=13)
X_res, y_res = sm.fit_resample(X_train, y_train)print('Original dataset shape is ', Counter(y_train))
print('Resample dataset shape is ', Counter(y_res))
```

```
Original dataset shape is  Counter({1: 641, 0: 163})
Resample dataset shape is  Counter({1: 641, 0: 641})
```

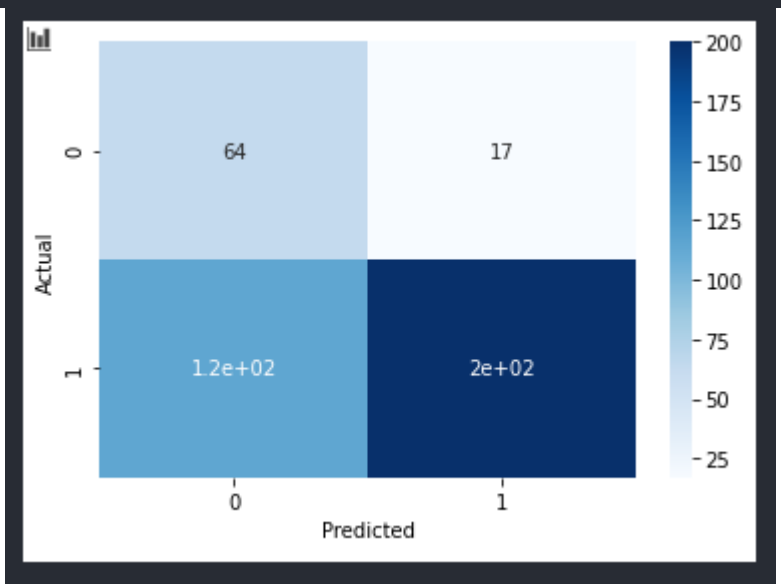
Terlihat bahwa data yang jumlah kelas minoritasnya adalah 163 setelah diresample menjadi 641 yaitu karena mengikut jumlah kelas mayoritas.
Selanjutnya mari kita bagi data yang telah diresample menjadi train dan test lalu kita latih dan ukur model kita.

```
clf = LogisticRegression(random_state=13)
clf.fit(X_res, y_res)y_pred = clf.predict(X_test)
print('Accuracy is ', accuracy_score(y_test, y_pred))
```

Akurasi model kita sekarang adalah 66% dari data yang telah diresampel. Apabila dibandingkan dengan akurasi sebelumnya memang turun, tetapi apabila kita cek confusion matrix beserta classification reportnya akan terdapat peningkatan.

```
print(classification_report(y_test, y_pred))cf_matrix = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cf_matrix, columns=['0', '1'], index=['0', '1'])
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
sns.heatmap(df_cm, cmap='Blues', annot=True)
```

	precision	recall	f1-score	support
0	0.36	0.79	0.49	81
1	0.92	0.63	0.75	315
accuracy			0.67	396
macro avg	0.64	0.71	0.62	396
weighted avg	0.81	0.67	0.70	396



Terlihat dari classification report bahwa F-score untuk kelas minoritas naik menjadi 0.49 yang berarti model lebih baik dari sebelumnya, hal ini dibuktikan juga dari grafik confusion matrix dari hasil prediksi data test. Model kita mampu memprediksi 64 kelas-0 dengan benar! bila dibandingan dengan sebelumnya (total nol) tanpa teknik Resampling ini tentu saja naik drastis.

Jadi, terbukti bahwa teknik Resampling dapat digunakan untuk kasus masalah klasifikasi seperti ini yaitu masalah klasifikasi dengan data yang tidak seimbang.

C. Latihan Praktikum

Memperkirakan performa generalisasi algoritma pembelajaran mesin pada tugas tertentu memerlukan data tambahan yang tidak digunakan selama pelatihan. Teknik Resampling mengacu pada proses berulang kali membagi data yang tersedia menjadi set pelatihan dan pengujian untuk memungkinkan estimasi kinerja yang tidak memiliki bias. Pada praktikum ini akan memperkenalkan strategi resampling umum danmengilustrasikan penggunaannya dengan ekosistem mlr3. Benchmarking dibangun berdasarkan resampling, mencakup perbandingan yang adil dari beberapa algoritma pembelajaran mesin pada setidaknya dilakukan dalam satu tugas. Yang mana menunjukkan bagaimana pembandingan dapat dilakukan dalam ekosistem mlr3, mulai dari konstruksi desain pembandingan hingga analisis statistik dari hasil pembandingan. Dalam Supervised

learning, model yang diterapkan dalam praktiknya diharapkan dapat menggeneralisasi dengan baik ke data baru yang tidak terlihat. Estimasi akurasi dari apa yang disebut kinerja generalisasi ini sangat penting untuk banyak aspek aplikasi dan penelitian pembelajaran mesin — apakah kita ingin membandingkan secara adil algoritma baru dengan algoritma yang sudah ada atau untuk menemukan algoritma terbaik untuk tugas tertentu setelah dilakukan penyetelan — yang mana selalu mengandalkan perkiraan kinerja ini. Oleh karena itu, estimasi kinerja adalah konsep dasar yang digunakan untuk pemilihan model, perbandingan model, dan penyetelan hyperparameter dalam pembelajaran mesin yang diawasi (Supervised Learning). Untuk menilai kinerja generalisasi model dengan benar, pertama-tama kita harus memutuskan ukuran kinerja yang sesuai untuk tugas dan tujuan evaluasi kita. Ukuran kinerja biasanya menghitung skor numerik yang menunjukkan, misalnya, seberapa cocok prediksi model dengan kebenaran dasar. Namun, itu juga dapat mencerminkan kualitas lain seperti waktu untuk melatih model.

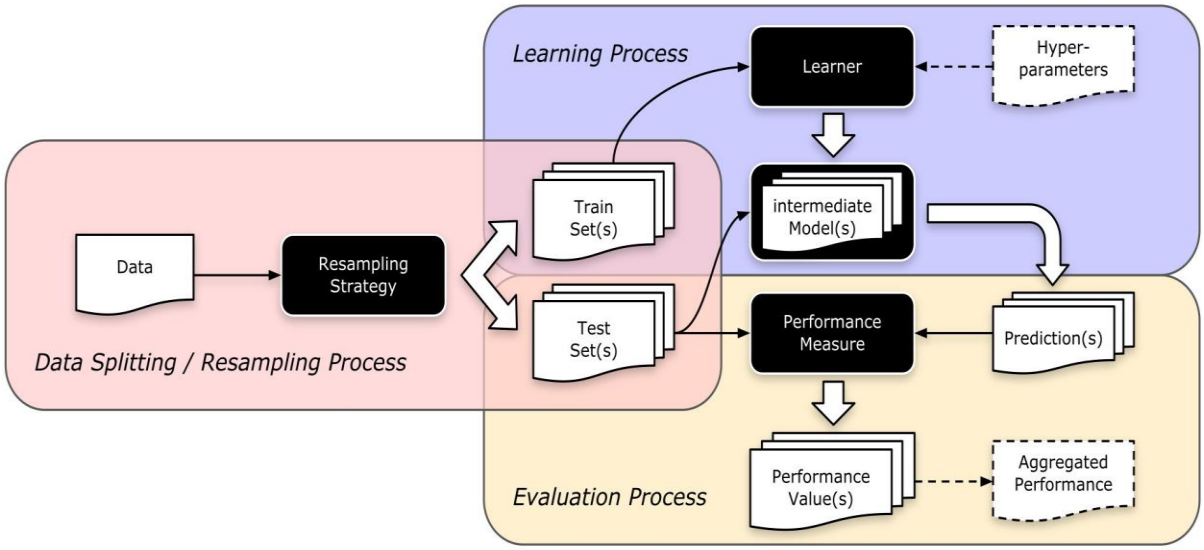
Setelah kita memutuskan ukuran kinerja, langkah selanjutnya adalah mengadopsi strategi yang menentukan cara menggunakan data yang tersedia untuk memperkirakan kinerja generalisasi. Sayangnya, menggunakan data yang sama untuk melatih dan menguji model adalah strategi yang buruk karena akan menghasilkan estimasi performa yang terlalu optimis. Misalnya, model overfitted mungkin sangat cocok dengan data yang dilatihnya, tetapi mungkin tidak dapat digeneralisasikan dengan baik ke data baru. Menilai kinerjanya menggunakan data yang sama dengan yang dilatihnya akan secara menyesatkan menyarankan model yang berkinerja baik. Oleh karena itu, merupakan praktik umum untuk menguji model pada data independen yang tidak digunakan untuk melatih model. Namun, kita biasanya melatih model yang diterapkan pada semua data yang tersedia, yang tidak menyisakan data untuk menilai kinerja generalisasinya. Untuk mengatasi masalah ini, strategi estimasi kinerja yang ada menahan sebagian dari data yang tersedia untuk tujuan evaluasi. Kumpulan tes yang disebut ini berfungsi sebagai data yang tidak terlihat dan digunakan untuk memperkirakan kinerja generalisasi.

Strategi sederhana yang umum adalah metode holdout, yang secara acak mempartisi data menjadi satu set pelatihan dan pengujian menggunakan rasio pemisahan yang telah ditentukan sebelumnya. Set pelatihan digunakan untuk membuat model perantara, yang tujuan utamanya adalah untuk memperkirakan kinerja menggunakan set tes. Estimasi performa ini kemudian digunakan sebagai proksi untuk performa model akhir yang dilatih pada semua data yang tersedia dan diterapkan dalam praktik. Idealnya, set pelatihan harus sebesar semua data yang tersedia sehingga model perantara mewakili model akhir dengan baik. Jika data pelatihan jauh lebih kecil, model perantara mempelajari hubungan yang kurang kompleks dibandingkan dengan model akhir, menghasilkan perkiraan kinerja yang bias secara pesimistis. Di sisi lain, kita juga menginginkan

sebanyak mungkin data uji untuk memperkirakan kinerja generalisasi secara andal.

Namun, kedua tujuan tersebut tidak mungkin jika kita hanya memiliki akses ke data dalam jumlah terbatas. Untuk mengatasi masalah ini, strategi resampling berulang kali membagi semua data yang tersedia menjadi beberapa set pelatihan dan pengujian, dengan satu pengulangan sesuai dengan apa yang disebut iterasi resampling di mlr3. Model perantara kemudian dilatih pada setiap set pelatihan dan set tes yang tersisa digunakan untuk mengukur kinerja di setiap iterasi resampling. Performa generalisasi akhirnya diestimasi oleh performa rata-rata selama beberapa iterasi resampling (lihat Gambar untuk ilustrasi). Metode resampling memungkinkan penggunaan lebih banyak titik data untuk pengujian, sekaligus mempertahankan set pelatihan sebesar mungkin. Secara khusus, mengulangi proses pemisahan data memungkinkan penggunaan semua titik data yang tersedia untuk menilai kinerja, karena setiap titik data dapat dipastikan menjadi bagian dari set pengujian setidaknya dalam satu iterasi resampling. Jumlah iterasi resampling yang lebih tinggi dapat mengurangi varians dan menghasilkan estimasi performa yang lebih andal.

Hal ini juga mengurangi risiko perkiraan kinerja yang sangat dipengaruhi oleh pembagian yang tidak menguntungkan yang tidak mencerminkan distribusi data asli dengan baik, yang merupakan masalah yang diketahui dari metode holdout. Namun, karena strategi resampling membuat beberapa model perantara yang dilatih pada bagian berbeda dari data yang tersedia dan rata-rata kinerjanya, mereka mengevaluasi kinerja algoritma pembelajaran yang menginduksi model ini, daripada kinerja model akhir yang digunakan dalam praktik. Oleh karena itu penting untuk melatih model perantara pada hampir semua titik data dari distribusi yang sama sehingga model perantara dan model akhir serupa. Jika kita hanya memiliki akses ke data dalam jumlah terbatas, yang terbaik yang dapat kita lakukan adalah menggunakan kinerja algoritma pembelajaran sebagai proxy untuk kinerja model akhir. Pada praktikum ini, kita akan belajar bagaimana memperkirakan performa generalisasi dari seorang Pelajar dengan menggunakan package mlr3.



Impor Library

install package berikut jika belum di pasang sebelumnya.

```
install.packages("tidyverse")
install.packages("mlr3verse")
install.packages("mlr3tuning")
install.packages("paradox")
install.packages("kknn")
install.packages("ggpubr")
```

Lalu panggil library yang digunakan:

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.0     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
```

```
## v ggplot2   3.4.1     v tibble    3.1.8
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(mlr3verse)
```

```
## Loading required package: mlr3
```

```
library(mlr3tuning)
```

```
## Loading required package: paradox
```

```
library(smotefamily)
```

Impor Data di R

download data: Diabetes Data

```
setwd("c:/Users/ardik/OneDrive/Pythonenv/Praktikum SSD")
diabetes <- read.csv("diabetes.csv",stringsAsFactors = TRUE)
diabetes <- diabetes %>% mutate(across(where(is.integer),as.numeric))
glimpse(diabetes)
```

```
## Rows: 752
## Columns: 9
## $ Pregnancies              <dbl> 6, 1, 8, 1, 0, 5, 3, 10, 8, 4, 10, 10, 1, 5, ~
## $ Glucose                  <dbl> 148, 85, 183, 89, 137, 116, 78, 115, 125, 110~
## $ BloodPressure            <dbl> 72, 66, 64, 66, 40, 74, 50, 0, 96, 92, 74, 80~
## $ SkinThickness            <dbl> 35, 29, 0, 23, 35, 0, 32, 0, 0, 0, 0, 0, 23, ~
## $ Insulin                  <dbl> 0, 0, 0, 94, 168, 0, 88, 0, 0, 0, 0, 0, 846, ~
## $ BMI                      <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35.~
## $ DiabetesPedigreeFunction <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.2~
## $ Age                      <dbl> 50, 31, 32, 21, 33, 30, 26, 29, 54, 30, 34, 5~
## $ Outcome                  <fct> Case, Control, Case, Control, Case, Control, ~
```

Kita akan menerapkan metode holdout dengan secara manual mempartisi data yang terdapat dalam objek Task (sebelumnya disebut Tugas) ke dalam satu set pelatihan (untuk melatih model) dan satu set pengujian (untuk memperkirakan kinerja generalisasi). Sebagai awal cepat untuk melakukan resampling dan benchmarking dengan package mlr3, kita menunjukkan contoh singkat bagaimana melakukannya dengan fungsi convenience resample() dan benchmark() . Secara khusus, kita menunjukkan cara memperkirakan kinerja generalisasi learner pada tugas yang diberikan dengan metode holdout menggunakan resample() dan cara menggunakan benchmark() untuk membandingkan dua learner pada task.

kita mendefinisikan objek Task dan Learner sebagai berikut:

```
task_diabetes = TaskClassif$new(id="diabetes",
                                backend = diabetes,
                                target = "Outcome",positive ="Case")
```

Argumen utama dalam fungsi TaskClassif$new adalah sebagai berikut:

`id` yang merupakan nama dari task (bisa diisi dengan nama apapun) `backend` adalah data yang ingin dimodelkan dengan catatan peubah respon-nya harus berupa peubah numerik `target` adalah nama kolom yang dijadikan peubah respon

Berikut adalah model-model yang akan digunakan beserta argumen di dalam fungsi learn:

Regresi Logistik - “classif.log_reg” - library(stats) Linear Discriminant Analysis - “classif.lda” - library(MASS)

Sebagai catatan, untuk model-model yang digunakan dalam mlr3 berasal dari package-package lain sehingga package-package tersebut perlu install terlebih dahulu. Kemudian, untuk model klasifikasi (respon biner atau multiclass) selalu diawali dengan kata “classif.”. Fungsi lrn juga memungkinkan untuk memasukan argumen-argumen dari package asalnya (termasuk hiperparameter).

Pada praktikum ini kita menggunakan model Regresi Logistik dan Linear Discriminant Analysis.

Learner

```
learner1 = lrn("classif.log_reg", predict_type = "prob")
learner1
```

```
## <LearnerClassifLogReg:classif.log_reg>
## * Model: -
## * Parameters: list()
## * Packages: mlr3, mlr3learners, stats
## * Predict Types:  response, [prob]
## * Feature Types: logical, integer, numeric, character, factor, ordered
## * Properties: loglik, twoclass
```

```
learner2 = lrn("classif.lda", predict_type = "prob")
learner2
```

```
## <LearnerClassifLDA:classif.lda>
## * Model: -
## * Parameters: list()
## * Packages: mlr3, mlr3learners, MASS
## * Predict Types:  response, [prob]
## * Feature Types: logical, integer, numeric, factor, ordered
## * Properties: multiclass, twoclass, weights
```

Berdasarkan output diatas argumen-argumen yang bisa digunakan dalam `classif.log_reg` dan `classif.lda` ada di kolom id. Selanjutnya, kolom class menunjukkan tipe data argumen tersebut. Kolom lower, upper dan levels merupakan isi/nilai dari argumen tersebut. Informasi ini bisa digunakan untuk melakukan tuning hyperparameter.

untuk memeriksa pengukuran dari klasifikasi ini dapat digunakan perintah berikut:

```
msr_tbl = as.data.table(mlr_measures)
msr_tbl[1:5, .(key, label, task_type)]
```

```
##             key                          label task_type
## 1:          aic   Akaika Information Criterion      <NA>
## 2:          bic Bayesian Information Criterion      <NA>
## 3:  classif.acc        Classification Accuracy   classif
## 4:  classif.auc       Area Under the ROC Curve   classif
## 5: classif.bacc              Balanced Accuracy   classif
```

```
msr_tbl[1:5, .(key, packages, predict_type, task_properties)]
```

```
##             key          packages predict_type task_properties
## 1:          aic              mlr3     response
## 2:          bic              mlr3     response
## 3:  classif.acc mlr3,mlr3measures     response
## 4:  classif.auc mlr3,mlr3measures         prob        twoclass
```

```
## 5: classif.bacc mlr3,mlr3measures     response
```

```
as.data.table(lrn("classif.log_reg")$param_set)
```

```
##              id    class lower upper      levels nlevels is_bounded
##  1:  dispersion ParamUty    NA    NA                 Inf      FALSE
##  2:     epsilon ParamDbl  -Inf   Inf                 Inf      FALSE
##  3:    etastart ParamUty    NA    NA                 Inf      FALSE
##  4:       maxit ParamDbl  -Inf   Inf                 Inf      FALSE
##  5:       model ParamLgl    NA    NA  TRUE,FALSE       2       TRUE
##  6:     mustart ParamUty    NA    NA                 Inf      FALSE
##  7:      offset ParamUty    NA    NA                 Inf      FALSE
##  8: singular.ok ParamLgl    NA    NA  TRUE,FALSE       2       TRUE
##  9:       start ParamUty    NA    NA                 Inf      FALSE
## 10:       trace ParamLgl    NA    NA  TRUE,FALSE       2       TRUE
## 11:           x ParamLgl    NA    NA  TRUE,FALSE       2       TRUE
## 12:           y ParamLgl    NA    NA  TRUE,FALSE       2       TRUE
##     special_vals        default storage_type          tags
##  1:    <list[0]>                   list       predict
##  2:    <list[0]>          1e-08      numeric train,control
##  3:    <list[0]> <NoDefault[3]>         list         train
##  4:    <list[0]>             25      numeric train,control
##  5:    <list[0]>           TRUE      logical         train
##  6:    <list[0]> <NoDefault[3]>         list         train
##  7:    <list[0]> <NoDefault[3]>         list         train
##  8:    <list[0]>           TRUE      logical         train
##  9:    <list[0]>                   list         train
## 10:    <list[0]>          FALSE      logical train,control
## 11:    <list[0]>          FALSE      logical         train
## 12:    <list[0]>           TRUE      logical         train
```

```
as.data.table(lrn("classif.lda")$param_set)
```

```
##               id    class lower upper                      levels nlevels
## 1:          dimen ParamUty    NA    NA                                 Inf
## 2:         method ParamFct    NA    NA           moment,mle,mve,t       4
## 3:             nu ParamInt  -Inf   Inf                                 Inf
## 4: predict.method ParamFct    NA    NA plug-in,predictive,debiased       3
## 5:  predict.prior ParamUty    NA    NA                                 Inf
## 6:          prior ParamUty    NA    NA                                 Inf
## 7:            tol ParamDbl  -Inf   Inf                                 Inf
##    is_bounded special_vals        default storage_type    tags
## 1:      FALSE    <list[0]> <NoDefault[3]>         list predict
## 2:       TRUE    <list[0]>         moment    character   train
## 3:      FALSE    <list[0]> <NoDefault[3]>      integer   train
## 4:       TRUE    <list[0]>        plug-in    character predict
## 5:      FALSE    <list[0]> <NoDefault[3]>         list predict
## 6:      FALSE    <list[0]> <NoDefault[3]>         list   train
## 7:      FALSE    <list[0]> <NoDefault[3]>      numeric   train
```

Pengukuran

Kode di bawah ini merupakan contoh penggunaan `holdout` (specified using `rsmp("holdout")`) untuk regresi logistik

```
resampling = rsmp("holdout")
rr = resample(task = task_diabetes, learner = learner1, resampling = resampling)
```

```
## INFO  [21:20:01.560] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 1/1)
```

```
rr$aggregate(msr("classif.acc"))
```

```
## classif.acc
##   0.7888446
```

Kode di bawah ini merupakan contoh penggunaan `holdout` (specified using `rsmp("holdout")`) untuk LDA

```
resampling = rsmp("holdout")
rr = resample(task = task_diabetes, learner = learner2, resampling = resampling)
```

```
## INFO  [21:20:01.719] [mlr3] Applying learner 'classif.lda' on task 'diabetes' (iter 1/1)
```

```
rr$aggregate(msr("classif.acc"))
```

```
## classif.acc
##   0.7410359
```

The `benchmark()` function juga menggunakan `resample()` function untuk mengestimasi performa berdasarkan strategi resampling. Sebagai contoh dapat dilihat minimal kode di bawah ini untuk membanding hasil classification accuracy regresi logistik:

```
lrns = c(learner1, lrn("classif.featureless"))
d = benchmark_grid(task = task_diabetes, learners = lrns, resampling = resampling)
bmr = benchmark(design = d)
```

```
## INFO  [21:20:01.842] [mlr3] Running benchmark with 2 resampling iterations
## INFO  [21:20:01.850] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 1/1)
## INFO  [21:20:01.887] [mlr3] Applying learner 'classif.featureless' on task 'diabetes' (iter 1/1)
## INFO  [21:20:01.908] [mlr3] Finished benchmark
```

```
acc = bmr$aggregate(msr("classif.acc"))
acc[, .(task_id, learner_id, classif.acc)]
```

```
##     task_id          learner_id classif.acc
## 1: diabetes     classif.log_reg   0.7888446
## 2: diabetes classif.featureless   0.6733068
```

```
lrns = c(learner2, lrn("classif.featureless"))
d = benchmark_grid(task = task_diabetes, learners = lrns, resampling = resampling)
bmr = benchmark(design = d)
```

```
## INFO  [21:20:02.042] [mlr3] Running benchmark with 2 resampling iterations
## INFO  [21:20:02.050] [mlr3] Applying learner 'classif.lda' on task 'diabetes' (iter 1/1)
## INFO  [21:20:02.083] [mlr3] Applying learner 'classif.featureless' on task 'diabetes' (iter 1/1)
## INFO  [21:20:02.111] [mlr3] Finished benchmark
```

```
acc = bmr$aggregate(msr("classif.acc"))
acc[, .(task_id, learner_id, classif.acc)]
```

```
##     task_id          learner_id classif.acc
## 1: diabetes         classif.lda   0.7729084
## 2: diabetes classif.featureless   0.6254980
```

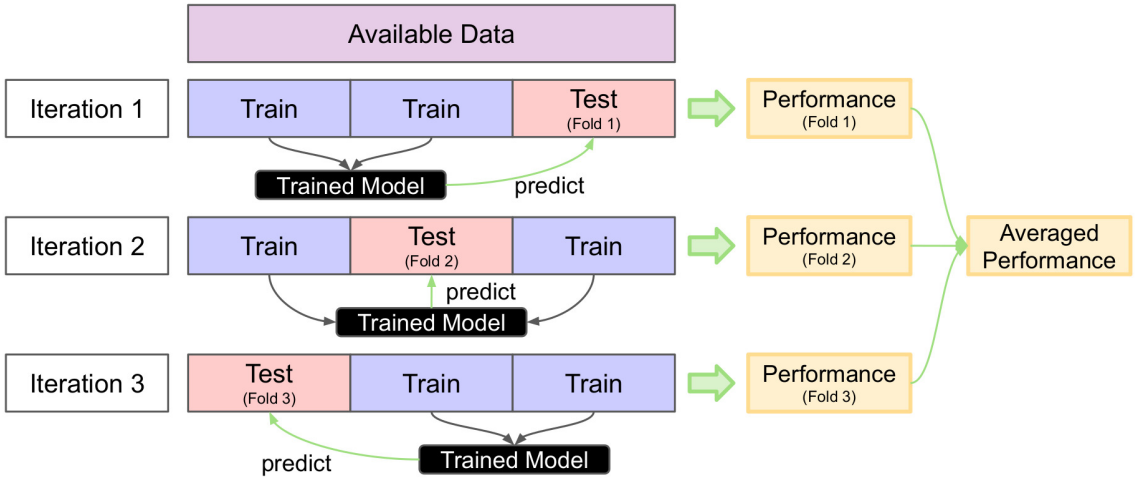
Apa itu Resampling?

Menurut Japkowicz and Shah (2011), Strategi resampling berbeda dengan mempartisi data menjadi training set dan test set. Sebagai contoh metode ini k-fold cross validation secara acak mempartisi data menjadi sejumlah k subset, yang disebut fold seperti yang ditunjukkan pada gambar. Kemudian k model di training pada proses pelatihan data dengan k-1 fold yang mana sisa nya digunakan pada data uji pada k iterasi. Hasil Performa k di estimasi dari setiap fold adalah rata-rata untuk menghitung estimasi performa.

Strategi resampling lainnya yang diketahui adalah subsampling dan bootsraping. subsampling dikenal sebagai repeated holdout pad resampling yang melakukan perulangan holdout metode, dan membuat banyak split train-test dengan mempertimbangkan rasio pengamatan untuk dimasukkan dalam set pelatihan.

Sementara Bootstrap adalah metode berbasis resampling data sampel dengan syarat pengembalian pada datanya dalam menyelesaikan statistik ukuran suatu sampel dengan harapan sampel tersebut mewakili data populasi sebenarnya, biasanya ukuran resampling diambil secara ribuan kali agar dapat mewakili data populasinya.Beberapa observasi dalam training set mungkin muncul lebih dari satu kali, sedangkan observasi lain yang tidak muncul sama sekali digunakan sebagai test set.

Pilihan strategi resampling biasanya bergantung pada tugas spesifik yang ada dan tujuan penilaian kinerja. Properti dan perangkap teknik resampling yang berbeda telah dipelajari secara luas dan dibahas dalam literatur, lihat misalnya, Bengio dan Grandvalet (2003), Molinaro, Simon, dan Pfeiffer (2005), Kim (2009), Bischl et al. (2012).



Gambar di atas menunjukkan 3-fold cross validation.

Strategi Resampling

Query

berikut beberapa perintah query yang digunakan dalam package mlr3

```
as.data.table(mlr_resamplings)
```

```
##           key                       label        params iters
## 1:  bootstrap                 Bootstrap ratio,repeats    30
## 2:     custom                Custom Splits                NA
## 3:  custom_cv Custom Split Cross-Validation                NA
## 4:         cv              Cross-Validation         folds    10
## 5:    holdout                        Holdout         ratio     1
## 6:   insample          Insample Resampling                1
```

```
## 7:         loo                  Leave-One-Out                   NA
## 8: repeated_cv     Repeated Cross-Validation folds,repeats   100
## 9: subsampling                   Subsampling ratio,repeats    30
```

Construction

Setelah kita memilih query apa yang akan digunakan dalam teknik resampling maka selanjutnya adalah membangun resampling melalui fungsi `rsmp()`

```
resampling = rsmp("holdout")
print(resampling)
```

```
## <ResamplingHoldout>: Holdout
## * Iterations: 1
## * Instantiated: FALSE
## * Parameters: ratio=0.6667
```

Secara default, holdout metode akan menggunakan 2/3 data sebagai data training dan 1/3 sebagai data test. kita dapat mengatur lebih spesifik lagi parameter rasui untuk holdout dengan meng-update rasio. sebagai contoh kita membangun objek resampling untuk holdout dengan split 80:20 (lihat kode di bawah ini)

```
resampling = rsmp("holdout", ratio = 0.8)
```

kemudian di update menjadi 50:50

```
resampling$param_set$values = list(ratio = 0.5)
```

Holdout hanya mengestimasi performa dengan menggunakan single set atau satu set. Untuk mendapatkan estimasi kinerja yang lebih andal dengan memanfaatkan semua data yang tersedia, kita dapat menggunakan strategi resampling lainnya. Misalnya, menyiapkan 10-fold cross-validation melalui

```
resampling = rsmp("cv", folds = 10)
```

Secara default, bidang `$is_instantiated` dari objek Resampling yang dibuat seperti yang ditunjukkan di atas disetel ke FALSE. Ini berarti bahwa strategi resampling belum diterapkan pada suatu Task, yang mana, pemisahan train-test tidak terdapat dalam objek Resampling.

Instantiation

Untuk menghasilkan pemisahan uji train untuk task tertentu, kita perlu membuat instance strategi resampling dengan memanggil metode `$instantiate()` dari objek Resampling yang dibangun sebelumnya pada Task. hal ini akan memanifestasikan partisi tetap dan menyimpan indeks baris untuk set pelatihan dan pengujian langsung di objek Resampling. Kita dapat mengakses baris ini melalui metode `$train_set()` dan `$test_set()` :

```
resampling = rsmp("holdout", ratio = 0.8)
resampling$instantiate(task_diabetes)
train_ids = resampling$train_set(1)
test_ids = resampling$test_set(1)
str(train_ids)
```

```
##  int [1:602] 2 3 4 6 7 8 9 10 11 12 ...
```

```
str(test_ids)
```

```
##  int [1:150] 1 5 16 18 26 31 35 44 61 76 ...
```

Instansiasi sangat relevan adalah ketika tujuannya adalah untuk membandingkan beberapa learner secara adil. Di sini, sangat penting untuk menggunakan pemisahan tes-latihan yang sama untuk mendapatkan hasil yang sebanding. Artinya, kita perlu memastikan bahwa semua learner yang akan dibandingkan menggunakan

data pelatihan yang sama untuk membuat model dan bahwa learner menggunakan data pengujian yang sama untuk mengevaluasi kinerja model.

Eksekusi

Memanggil fungsi `resample()` pada Task, learner, dan objek resampling yang dibangun akan mengembalikan objek `ResampleResult` yang memuat semua informasi yang diperlukan untuk memperkirakan performa secara umum. Secara khusus, fungsi tersebut akan menggunakan learner secara internal untuk melatih model untuk setiap rangkaian pelatihan yang ditentukan oleh strategi resampling dan menyimpan prediksi model dari setiap rangkaian pengujian. Kita dapat menerapkan fungsi `print` atau `as.data.table` ke objek `ResampleResult` untuk mendapatkan beberapa informasi dasar:

```
resampling = rsmp("cv", folds = 4)
rr = resample(task_diabetes, learner1, resampling)
```

```
## INFO  [21:20:02.442] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 1/4)
## INFO  [21:20:02.550] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 2/4)
## INFO  [21:20:02.594] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 3/4)
## INFO  [21:20:02.644] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 4/4)
```

```
print(rr)
```

```
## <ResampleResult> of 4 iterations
## * Task: diabetes
## * Learner: classif.log_reg
## * Warnings: 0 in 0 iterations
## * Errors: 0 in 0 iterations
```

```
as.data.table(rr)
```

```
##                 task                    learner         resampling iteration
## 1: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         1
## 2: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         2
## 3: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         3
## 4: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         4
##                 prediction
## 1: <PredictionClassif[20]>
## 2: <PredictionClassif[20]>
## 3: <PredictionClassif[20]>
## 4: <PredictionClassif[20]>
```

Di sini, kita menggunakan 4-fold cross-validation sebagai strategi resampling. Objek `ResampleResult` yang dihasilkan (disimpan sebagai rr) menyediakan berbagai metode untuk mengakses informasi yang disimpan. Dua metode yang paling relevan untuk penilaian kinerja adalah `$score()` dan `$aggregate()`.

Dalam contoh kode di bawah ini, secara eksplisit menggunakan akurasi klasifikasi (`classif.acc`) sebagai ukuran kinerja dan meneruskannya ke metode `$score()` untuk mendapatkan perkiraan kinerja setiap iterasi resampling secara terpisah:

```
acc = rr$score(msr("classif.acc"))
acc[, .(iteration, classif.acc)]
```

```
##    iteration classif.acc
## 1:         1   0.7500000
## 2:         2   0.7925532
## 3:         3   0.7819149
## 4:         4   0.7872340
```

Demikian pula, kita bisa meneruskan objek Measure ke metode `$aggregate()` untuk menghitung skor agregat di semua iterasi resampling. Jenis agregasi biasanya ditentukan oleh objek Ukur. Ada dua pendekatan untuk menggabungkan skor di seluruh iterasi resampling: Yang pertama disebut sebagai rata-rata makro (macro average), yang pertama menghitung ukuran di setiap iterasi resampling secara terpisah, lalu rata-rata skor ini di semua iterasi. Pendekatan kedua adalah rata-rata mikro (micro average), yang mengumpulkan semua prediksi lintas iterasi resampling menjadi satu objek Prediksi dan menghitung ukurannya secara langsung. Akurasi klasifikasi `msr("classif.acc")` menggunakan rata-rata makro secara default, tetapi rata-rata mikro dapat dihitung juga dengan menentukan argumen rata-rata:

```
rr$aggregate(msr("classif.acc"))
```

```
## classif.acc
##   0.7779255
```

```
rr$aggregate(msr("classif.acc", average = "micro"))
```

```
## classif.acc
##   0.7779255
```

Inspeksi

```
rrdt = as.data.table(rr)
rrdt
```

```
##                 task                     learner         resampling iteration
## 1: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         1
## 2: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         2
## 3: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         3
## 4: <TaskClassif[50]> <LearnerClassifLogReg[37]> <ResamplingCV[20]>         4
##                 prediction
## 1: <PredictionClassif[20]>
## 2: <PredictionClassif[20]>
## 3: <PredictionClassif[20]>
## 4: <PredictionClassif[20]>
```

```
rrdt$prediction
```

```
## [[1]]
## <PredictionClassif> for 188 observations:
##     row_ids   truth response  prob.Case prob.Control
##           1    Case     Case 0.70969087    0.2903091
##           2 Control  Control 0.04137551    0.9586245
##           9    Case  Control 0.03613102    0.9638690
## ---
##         750 Control  Control 0.15855495    0.8414450
##         751    Case  Control 0.30657191    0.6934281
##         752 Control  Control 0.06125233    0.9387477
##
## [[2]]
## <PredictionClassif> for 188 observations:
##     row_ids   truth response  prob.Case prob.Control
##          10 Control  Control 0.19345695    0.8065430
##          17    Case  Control 0.16757422    0.8324258
##          25    Case  Control 0.39933294    0.6006671
## ---
##         738    Case     Case 0.63318650    0.3668135
```

```
##         739    Case     Case 0.70765429    0.2923457
##         747 Control  Control 0.07588355    0.9241164
##
## [[3]]
## <PredictionClassif> for 188 observations:
##     row_ids   truth response prob.Case prob.Control
##           5    Case     Case 0.8920442    0.1079558
##           7    Case  Control 0.0818699    0.9181301
##           8 Control     Case 0.6750419    0.3249581
## ---
##         742    Case  Control 0.3417545    0.6582455
##         745 Control  Control 0.1015957    0.8984043
##         748 Control  Control 0.4531973    0.5468027
##
## [[4]]
## <PredictionClassif> for 188 observations:
##     row_ids   truth response prob.Case prob.Control
##           3    Case     Case 0.8234756    0.1765244
##           4 Control  Control 0.0370062    0.9629938
##           6 Control  Control 0.1442955    0.8557045
## ---
##         732    Case     Case 0.7497753    0.2502247
##         733 Control  Control 0.4049310    0.5950690
##         741 Control  Control 0.4823899    0.5176101
```

```
all.equal(rrdt$prediction, rr$predictions())
```

```
## [1] TRUE
```

```
pred = rr$prediction()
pred
```

```
## <PredictionClassif> for 752 observations:
##     row_ids   truth response  prob.Case prob.Control
##           1    Case     Case 0.70969087    0.2903091
##           2 Control  Control 0.04137551    0.9586245
##           9    Case  Control 0.03613102    0.9638690
## ---
##         732    Case     Case 0.74977532    0.2502247
##         733 Control  Control 0.40493101    0.5950690
##         741 Control  Control 0.48238991    0.5176101
```

```
pred$score(msr("classif.acc"))
```

```
## classif.acc
##   0.7779255
```

Resampling with Stratification and Grouping

Di mlr3, kita dapat menetapkan peran khusus ke fitur yang terdapat dalam data dengan mengonfigurasi bidang `$col_roles` yang sesuai dari sebuah Task. Dua peran kolom relevan yang akan memengaruhi perilaku strategi resampling adalah “grup” atau “strata atau stratum”. Pada contoh kali ini digunakan dataset “penguins”.

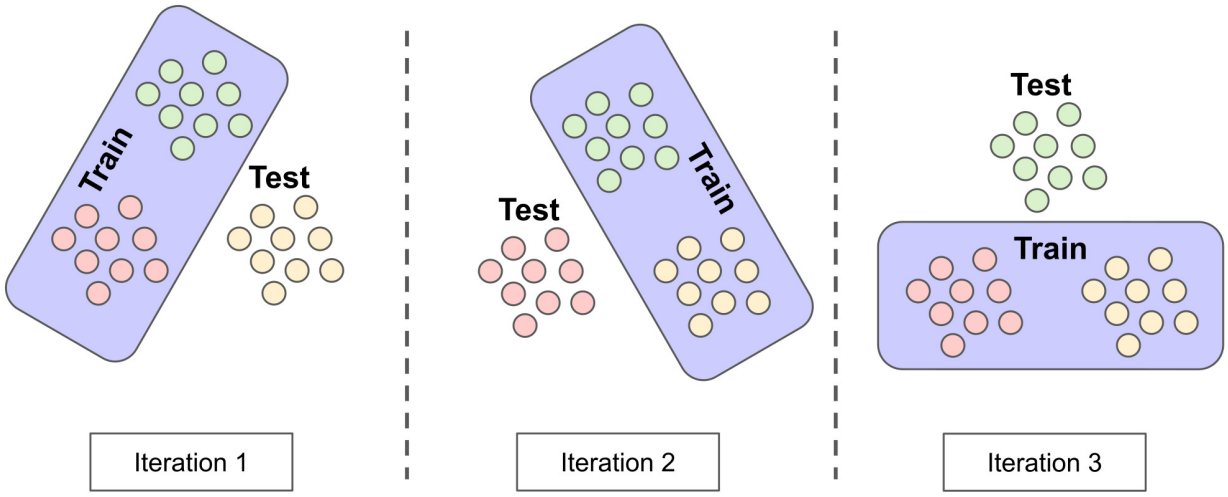


Illustration of the train-test splits of a leave-one-object-out cross-validation with 3 groups of observations (highlighted by different colors)

```
task_grp = tsk("penguins")
task_grp$col_roles$group = "year"
r = rsmp("loo")
r$instantiate(task_grp)

table(task_grp$data(cols = "year"))
```

```
## year
## 2007 2008 2009
##  110  114  120
```

```
table(task_grp$data(rows = r$test_set(1), cols = "year"))
```

```
## year
## 2009
##  120
```

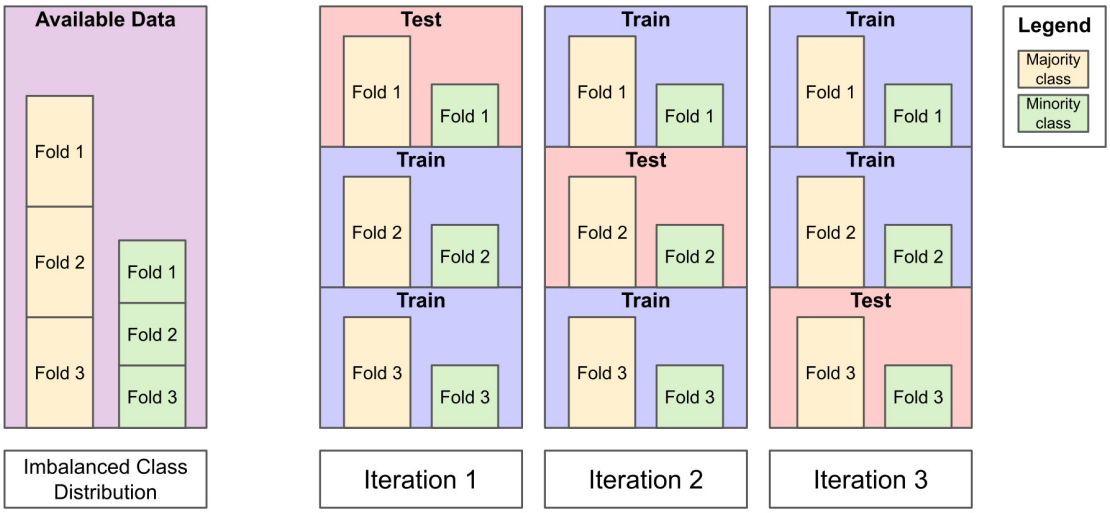
```
table(task_grp$data(rows = r$test_set(2), cols = "year"))
```

```
## year
## 2008
##  114
```

```
table(task_grp$data(rows = r$test_set(3), cols = "year"))
```

```
## year
## 2007
##  110
```

Peran kolom lain yang tersedia di mlr3 adalah “strata”, yang menerapkan pengambilan sampel bertingkat. Stratified sampling memastikan bahwa satu atau lebih fitur diskrit dalam set pelatihan dan pengujian akan memiliki distribusi yang sama seperti pada tugas awal yang berisi semua pengamatan. Ini sangat berguna ketika fitur diskrit sangat tidak seimbang (imbalance dataset) dan kita ingin memastikan bahwa distribusi fitur tersebut serupa di setiap iterasi resampling. Stratifikasi umumnya digunakan untuk tugas klasifikasi yang tidak seimbang di mana kelas fitur target tidak seimbang



Pada contoh di bawah ini kita menerapkam stratum pada dataset diabetes - Sebelum dilakukan stratum:

```
prop.table(table(task_diabetes$data(cols = "Outcome")))
```

```
## Outcome
##      Case   Control
## 0.3484043 0.6515957
```

```
r = rsmp("cv", folds = 3)
r$instantiate(task_diabetes)
prop.table(table(task_diabetes$data(rows = r$test_set(1), cols = "Outcome")))
```

```
## Outcome
##      Case   Control
## 0.3864542 0.6135458
```

```
prop.table(table(task_diabetes$data(rows = r$test_set(2), cols = "Outcome")))
```

```
## Outcome
##      Case   Control
## 0.3505976 0.6494024
```

```
prop.table(table(task_diabetes$data(rows = r$test_set(3), cols = "Outcome")))
```

```
## Outcome
##    Case Control
##   0.308   0.692
```

Menggunakan Stratum:

```
task_diabetes$col_roles$stratum = "Outcome"
r = rsmp("cv", folds = 3)
r$instantiate(task_diabetes)

prop.table(table(task_diabetes$data(rows = r$test_set(1), cols = "Outcome")))
```

```
## Outcome
##      Case   Control
## 0.3492063 0.6507937
```

```
prop.table(table(task_diabetes$data(rows = r$test_set(2), cols = "Outcome")))
```

```
## Outcome
##    Case Control
##   0.348   0.652
```

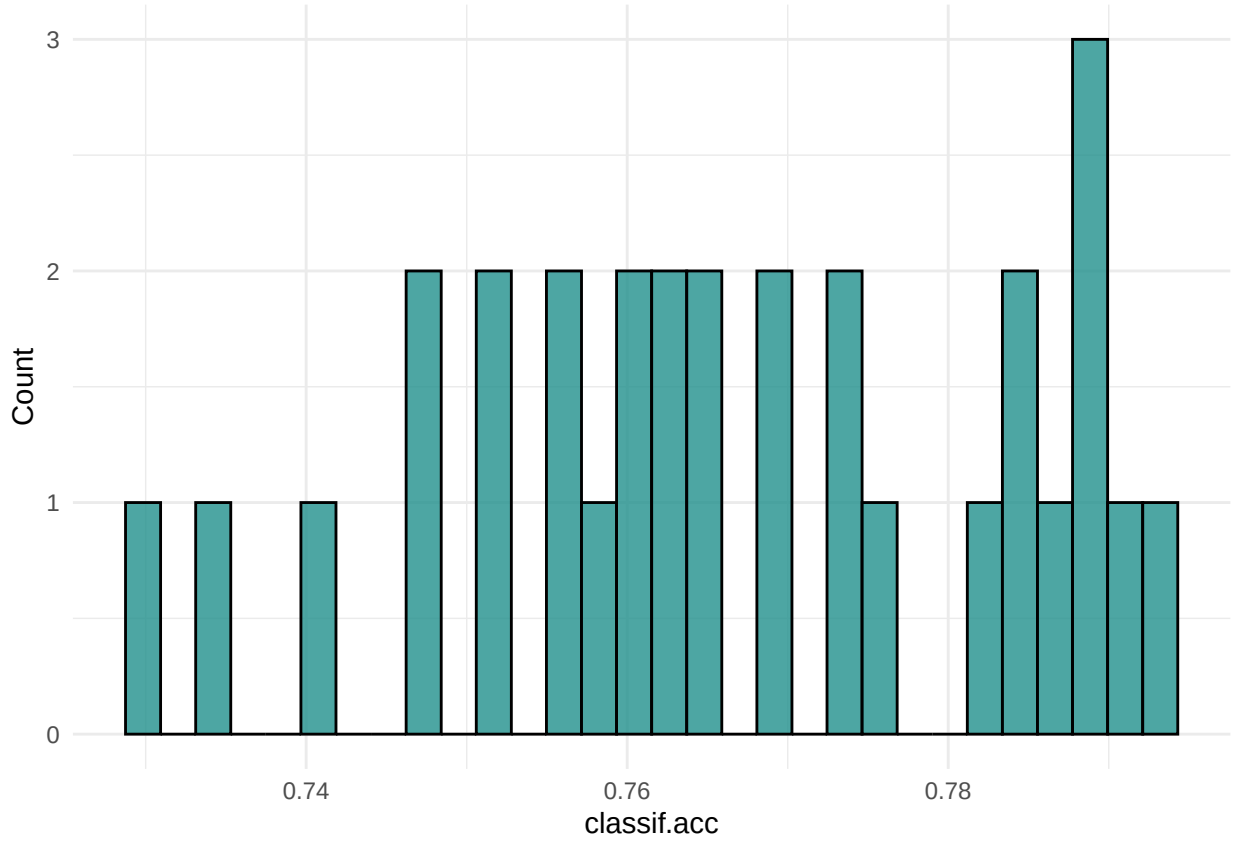
```
prop.table(table(task_diabetes$data(rows = r$test_set(3), cols = "Outcome")))
```

```
## Outcome
##    Case Control
##   0.348   0.652
```

Plotting Resampling

```
## INFO  [21:20:03.507] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 1/30)
## INFO  [21:20:03.558] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 2/30)
## INFO  [21:20:03.605] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 3/30)
## INFO  [21:20:03.642] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 4/30)
## INFO  [21:20:03.675] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 5/30)
## INFO  [21:20:03.707] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 6/30)
## INFO  [21:20:03.739] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 7/30)
## INFO  [21:20:03.780] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 8/30)
## INFO  [21:20:03.805] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 9/30)
## INFO  [21:20:03.832] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 10/30)
## INFO  [21:20:03.867] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 11/30)
## INFO  [21:20:03.899] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 12/30)
## INFO  [21:20:03.931] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 13/30)
## INFO  [21:20:03.962] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 14/30)
## INFO  [21:20:04.004] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 15/30)
## INFO  [21:20:04.036] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 16/30)
## INFO  [21:20:04.068] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 17/30)
## INFO  [21:20:04.103] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 18/30)
## INFO  [21:20:04.139] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 19/30)
## INFO  [21:20:04.185] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 20/30)
## INFO  [21:20:04.219] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 21/30)
## INFO  [21:20:04.257] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 22/30)
## INFO  [21:20:04.292] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 23/30)
## INFO  [21:20:04.325] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 24/30)
## INFO  [21:20:04.361] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 25/30)
## INFO  [21:20:04.404] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 26/30)
## INFO  [21:20:04.445] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 27/30)
## INFO  [21:20:04.492] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 28/30)
## INFO  [21:20:04.525] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 29/30)
## INFO  [21:20:04.557] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 30/30)
```

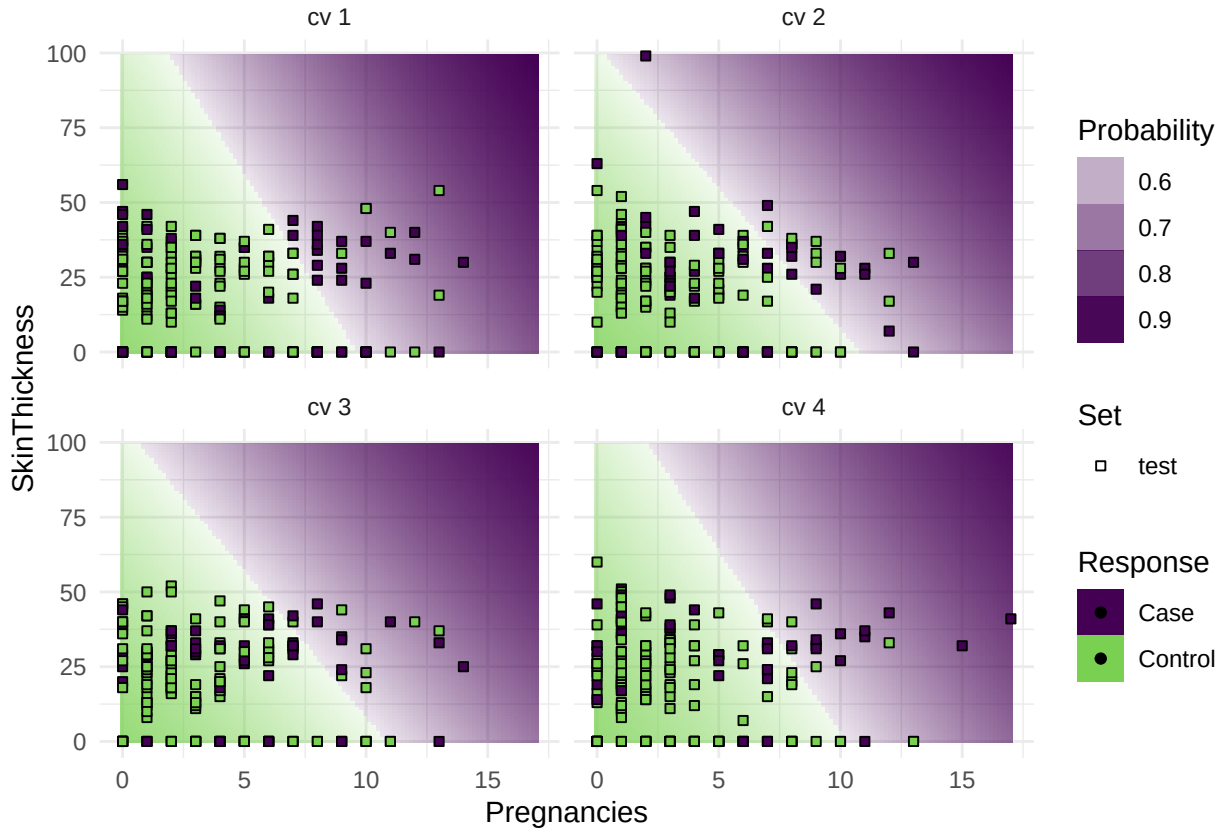

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Histogram berguna untuk mengukur secara visual varian dari hasil kinerja di seluruh iterasi resampling, sedangkan boxplot sering digunakan saat banyak learner dibandingkan secara berdampingan.

Kita juga dapat memvisualisasikan permukaan prediksi 2 dimensi dari masing-masing model di setiap iterasi resampling jika tugas dibatasi pada dua fitur:

```
## INFO  [21:20:05.969] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 1/4)
## INFO  [21:20:06.023] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 2/4)
## INFO  [21:20:06.059] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 3/4)
## INFO  [21:20:06.087] [mlr3] Applying learner 'classif.log_reg' on task 'diabetes' (iter 4/4)
```



ROC Analysis

Analisis ROC (Receiver Operating Characteristic) banyak digunakan untuk mengevaluasi pengklasifikasi biner. Meskipun ada ekstensi untuk pengklasifikasi multikelas (lihat misalnya Hand and Till (2001)), kita hanya akan membahas kasus klasifikasi biner yang jauh lebih mudah di sini. Untuk pengklasifikasi biner yang memprediksi kelas diskrit, kita dapat menghitung confusion matrix yang menghitung besaran berikut (lihat juga Gambar 3.6):

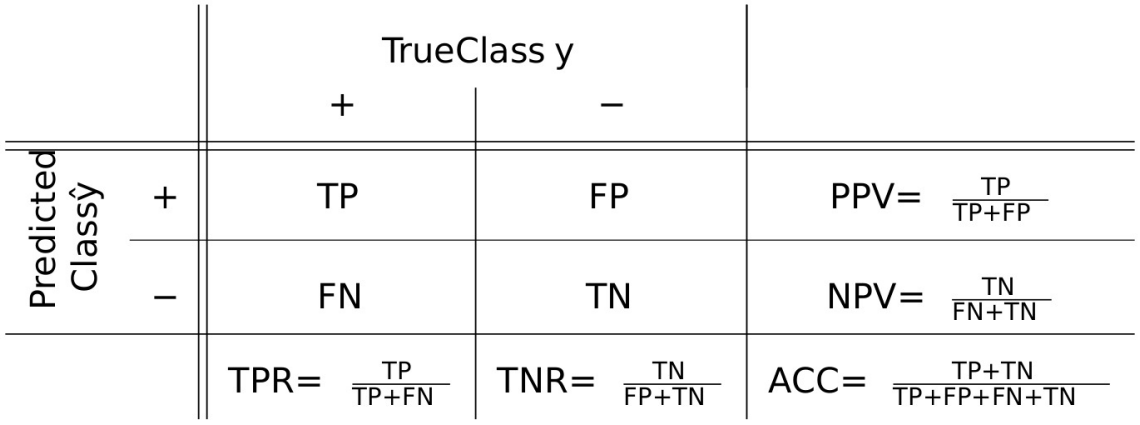
True positives (TP): Contoh yang benar-benar positif dan diklasifikasikan dengan benar sebagai positif. True Negatives (TN): Contoh yang benar-benar negatif dan diklasifikasikan dengan benar sebagai negatif. Positif palsu (FP): Contoh yang sebenarnya negatif tetapi salah diklasifikasikan sebagai positif. Negatif Palsu (FN): Contoh yang sebenarnya positif tetapi salah diklasifikasikan sebagai negatif.

Confusion Matrix-based Measures

Beberapa ukuran kinerja umum yang didasarkan pada confusion matix dan mengukur kemampuan pengklasifikasi untuk memisahkan dua kelas (yaitu, kinerja diskriminasi) meliputi (lihat juga Gambar di bawah ini untuk definisinya berdasarkan TP, FP, TN, dan FN):

- True Positive Rate (TPR), Sensitivity or Recall: How many of the true positives did we predict as positive?
- True Negative Rate (TNR) or Specificity: How many of the true negatives did we predict as negative?
- False Positive Rate (FPR), or 1 - Specificity: How many of the true negatives did we predict as positive?
- Positive Predictive Value (PPV) or Precision: If we predict positive how likely is it a true positive?
- Negative Predictive Value (NPV): If we predict negative how likely is it a true negative?

Akurasi (ACC): Proporsi instance yang diklasifikasikan dengan benar dari jumlah total instance. F1-score: The harmonic mean of precision and recall, which balances the trade-off between precision and recall.




```
splits = partition(task_diabetes, ratio = 0.8)

learner1$train(task_diabetes, splits$train)
pred = learner1$predict(task_diabetes, splits$test)
pred$confusion
```

```
##          truth
## response  Case Control
##   Case       9       5
##   Control   43      93
```

Package `mlr3measures` memungkinkan untuk menghitung tambahan beberapa pengukuran berbasis confusion matrix yang umum menggunakan fungsi Confusion_matrix :

```
mlr3measures::confusion_matrix(truth = pred$truth,
  response = pred$response, positive = task_diabetes$positive)
```

```
##          truth
## response  Case Control
##   Case       9       5
##   Control   43      93
## acc :  0.6800; ce  :  0.3200; dor :  3.8930; f1  :  0.2727
## fdr :  0.3571; fnr :  0.8269; fomr:  0.3162; fpr :  0.0510
## mcc :  0.1997; npv :  0.6838; ppv :  0.6429; tnr :  0.9490
## tpr :  0.1731
```

Jika pengklasifikasi biner memprediksi probabilitas alih-alih kelas diskrit, kita dapat secara sewenang-wenang menetapkan ambang batas untuk memotong probabilitas dan menugaskannya ke kelas positif dan negatif. Dalam hal kinerja klasifikasi, umumnya sulit untuk mencapai TPR yang tinggi dan FPR yang rendah secara bersamaan karena sering terjadi trade-off antara kedua tarif tersebut. Meningkatkan ambang batas untuk mengidentifikasi kasus positif, mengarah ke jumlah prediksi negatif yang lebih tinggi dan prediksi positif yang lebih sedikit. Akibatnya, FPR biasanya lebih baik (lebih rendah), tetapi dengan nilai TPR yang lebih buruk (lebih rendah). Misalnya, dalam kasus khusus di mana ambang batas ditetapkan terlalu tinggi dan tidak ada instance yang diprediksi sebagai positif, matriks konfusi menunjukkan nol positif sejati (tidak ada instance yang benar-benar positif dan diklasifikasikan dengan benar sebagai positif) dan nol positif palsu (tidak ada instance yang sebenarnya negatif tetapi salah diklasifikasikan sebagai positif). Oleh karena itu, FPR dan TPR juga nol karena tidak ada false positive dan nol true positive. Sebaliknya, menurunkan ambang batas untuk

mengidentifikasi kasus positif mungkin tidak pernah memprediksi kelas negatif dan dapat meningkatkan (memperbaiki) TPR, tetapi dengan biaya FPR yang lebih buruk (lebih tinggi). Misalnya, di bawah ini kami menetapkan ambang batas (*Threshold*) ke 0,99 dan 0,01:

```
pred$set_threshold(0.99)
mlr3measures::confusion_matrix(pred$truth, pred$response, task_diabetes$positive)
```

```
##          truth
## response  Case Control
##   Case       0       0
##   Control   52      98
## acc :  0.6533; ce  :  0.3467; dor :  NaN; f1  :  NaN
## fdr :  NaN; fnr :  1.0000; fomr:  0.3467; fpr :  0.0000
## mcc :  0.0000; npv :  0.6533; ppv :  NaN; tnr :  1.0000
## tpr :  0.0000
```

```
pred$set_threshold(0.01)
mlr3measures::confusion_matrix(pred$truth, pred$response, task_diabetes$positive)
```

```
##          truth
## response  Case Control
##   Case      52      98
##   Control    0       0
## acc :  0.3467; ce  :  0.6533; dor :  NaN; f1  :  0.5149
## fdr :  0.6533; fnr :  0.0000; fomr:  NaN; fpr :  1.0000
## mcc :  0.0000; npv :  NaN; ppv :  0.3467; tnr :  0.0000
## tpr :  1.0000
```

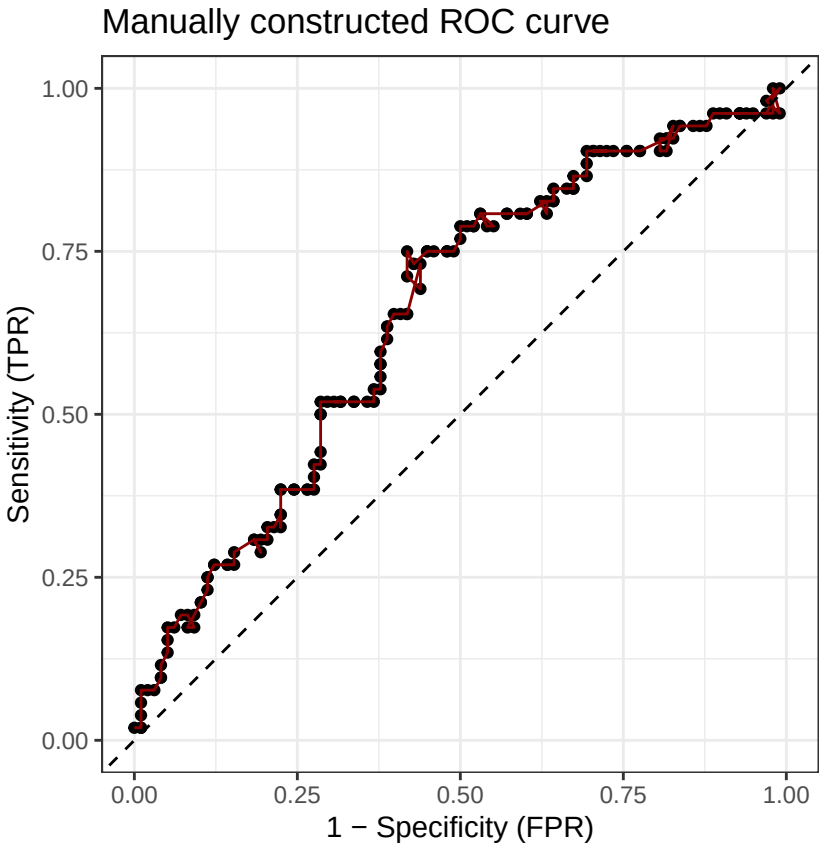
ROC Curve

```
thresholds = sort(pred$prob[,1])

rocvals = data.table::rbindlist(lapply(thresholds, function(t) {
  pred$set_threshold(t)
  data.frame(
    threshold = t,
    FPR = pred$score(msr("classif.fpr")),
    TPR = pred$score(msr("classif.tpr"))
  )
}))

head(rocvals)
```

```
##    threshold       FPR       TPR
## 1: 0.1908231 0.9693878 0.9615385
## 2: 0.1908231 0.9897959 1.0000000
## 3: 0.1908231 0.9693878 0.9807692
## 4: 0.1908231 0.9897959 0.9615385
## 5: 0.1908231 0.9795918 1.0000000
## 6: 0.1908231 0.9795918 0.9615385
```



Ukuran kinerja yang dapat diturunkan dari kurva ROC adalah area di bawah kurva the area under the curve (AUC). Semakin tinggi nilai AUC, semakin baik kinerjanya, sedangkan pengklasifikasi acak akan menghasilkan AUC sebesar 0,5. AUC dapat diartikan sebagai probabilitas bahwa instance positif yang dipilih secara acak diberi peringkat lebih tinggi (dalam artian mendapat probabilitas prediksi yang lebih tinggi untuk menjadi kelas positif) oleh model klasifikasi daripada instance negatif yang dipilih secara acak.

Melanjutkan Teknik Resampling pada Regresi Logistik dan LDA

```
standardize <- po("scale")
# Jika dup_size=1, jumlah amatan kelas  minoritas
#bertambah sebanyak
#1*(jumlah amatan awal)+jumlah amatan awal

smote <- po("smote",dup_size=1)
```

```
standardize$train(list(task_diabetes))[[1]]$data() %>% glimpse
```

```
## Rows: 752
## Columns: 3
## $ Outcome       <fct> Case, Control, Case, Control, Case, Control, Case, Contr~
## $ Pregnancies   <dbl> 0.63448799, -0.84914064, 1.22793944, -0.84914064, -1.145~
## $ SkinThickness <dbl> 0.90765717, 0.53169562, -1.28545187, 0.15573407, 0.90765~
```

```
smote$train(list(task_diabetes))[[1]]$data() %>% count(Outcome)
```

```
##    Outcome   n
```

```
## 1:    Case 524
## 2: Control 490
```

```
reglog <- GraphLearner$new(standardize %>>% smote %>>% lrn("classif.log_reg"))
reglog
```

```
## <GraphLearner:scale.smote.classif.log_reg>
## * Model: -
## * Parameters: scale.robust=FALSE, smote.dup_size=1
## * Packages: mlr3, mlr3pipelines, smotefamily, mlr3learners, stats
## * Predict Types:  [response], prob
## * Feature Types: logical, integer, numeric, character, factor, ordered,
##   POSIXct
## * Properties: featureless, hotstart_backward, hotstart_forward,
##   importance, loglik, missings, multiclass, oob_error,
##   selected_features, twoclass, weights
```

```
lda <- GraphLearner$new(standardize %>>%
smote %>>%                          lrn("classif.lda",method="moment"))
lda
```

```
## <GraphLearner:scale.smote.classif.lda>
## * Model: -
## * Parameters: scale.robust=FALSE, smote.dup_size=1,
##   classif.lda.method=moment
## * Packages: mlr3, mlr3pipelines, smotefamily, mlr3learners, MASS
## * Predict Types:  [response], prob
## * Feature Types: logical, integer, numeric, character, factor, ordered,
##   POSIXct
## * Properties: featureless, hotstart_backward, hotstart_forward,
##   importance, loglik, missings, multiclass, oob_error,
##   selected_features, twoclass, weights
```

Intepretasi Model

Tahap ini biasanya dilakukan untuk melihat bagaimana pengaruh peubah-peubah prediktor terhadap respon menurut masing-masing model. Misalnya saja dalam regresi logistik besarnya nilai koefisien,odds ratio dan p-value bisa menggambarkan bagaimana pengaruh peubah prediktor terhadap respon.

Sebelum kita bisa memperoleh interpretasi dari suatu model dalam mlr3, kita perlu melakukan proses modeling/training terlebih dahulu dengan menggunakan keseluruhan data. Kemudian, Untuk menampilkan nilai koefisien dan p-value pada model regresi logistik, kita bisa menggunakan fungsi summary.

Regresi Logistik

```
# train model dengan keseluruhan data
reglog$train(task = task_diabetes)
summary(reglog$model$classif.log_reg$model)
```

```
##
## Call:
## stats::glm(formula = task$formula(), family = "binomial", data = data,
##     model = FALSE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -1.9560  -1.1047   0.6892   1.1486   1.5155
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    0.03789    0.06479   0.585  0.55862
## Pregnancies    0.47770    0.06715   7.114 1.13e-12 ***
## SkinThickness  0.20018    0.06393   3.131  0.00174 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1404.6  on 1013  degrees of freedom
## Residual deviance: 1343.2  on 1011  degrees of freedom
## AIC: 1349.2
##
## Number of Fisher Scoring iterations: 4
```

Selain fungsi summary, kita juga bisa menggunakan fungsi tidy dari package broom untuk menampilkan hal yang sama. Hanya saja fungsi tidy menampilkan nilai koefisien dan p-value dalam bentuk data.frame

```
broom::tidy(reglog$model$classif.log_reg$model)
```

```
## # A tibble: 3 x 5
##   term          estimate std.error statistic  p.value
##   <chr>            <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)     0.0379    0.0648     0.585 5.59e- 1
## 2 Pregnancies     0.478     0.0671     7.11  1.13e-12
## 3 SkinThickness   0.200     0.0639     3.13  1.74e- 3
```

Kita bisa menambahkan odds ratio dengan menggunakan sintaks berikut:

```
broom::tidy(reglog$model$classif.log_reg$model) %>%
  mutate(OddsRatio = exp(estimate))
```

```
## # A tibble: 3 x 6
##   term          estimate std.error statistic  p.value OddsRatio
##   <chr>            <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
## 1 (Intercept)     0.0379    0.0648     0.585 5.59e- 1      1.04
## 2 Pregnancies     0.478     0.0671     7.11  1.13e-12      1.61
## 3 SkinThickness   0.200     0.0639     3.13  1.74e- 3      1.22
```

```
# menampilkan informasi tambahan tentang model
broom::glance(reglog$model$classif.log_reg$model)
```

```
## # A tibble: 1 x 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1         1405.    1013  -672. 1349. 1364.    1343.        1011  1014
```

Contoh Interpretasi odds ratio:

Age : setiap penambahan Age sebesar 1 tahun maka akan meningkatkan resiko diabetes sebesar 1.2 kali.

Insulin : setiap penambahan Insulin sebesar 1 satuan maka akan menurunkan resiko diabetes sebesar 0.8 kali.

LDA

```
# train model dengan keseluruhan data
lda$train(task = task_diabetes)
```

```
coef_lda <- coef(lda$model$classif.lda$model)
coef_lda
```

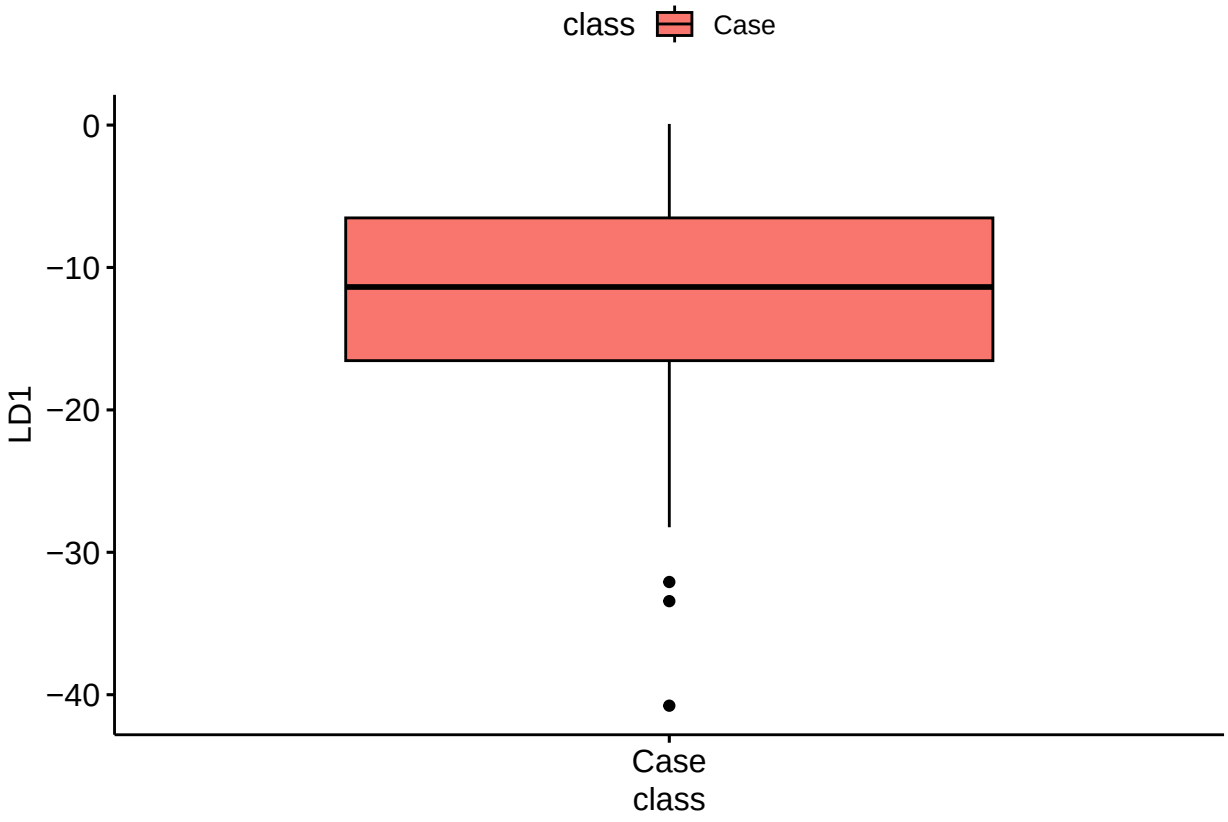
```
##                       LD1
## Pregnancies   -0.9429460
## SkinThickness -0.3935996
```

```
predictedLD <- predict(lda$model$classif.lda$model,newdata = diabetes)
plotLD <- data.frame(predictedLD$x,class=predictedLD$class)
glimpse(plotLD)
```

```
## Rows: 752
## Columns: 2
## $ LD1   <dbl> -19.351918, -12.275591, -7.461825, -9.913994, -13.694242, -4.632~
## $ class <fct> Case, Case, Case, Case, Case, Case, Case, Case, Case, Case, Case~
```

```
plotLD %>% count(class)
```

```
##   class   n
## 1  Case 752
```



Selanjutnya lakukan teknik resampling yang sudah di jelaskan sebelumnya dan lakukan prediksi terhadap masing-masing kelas.