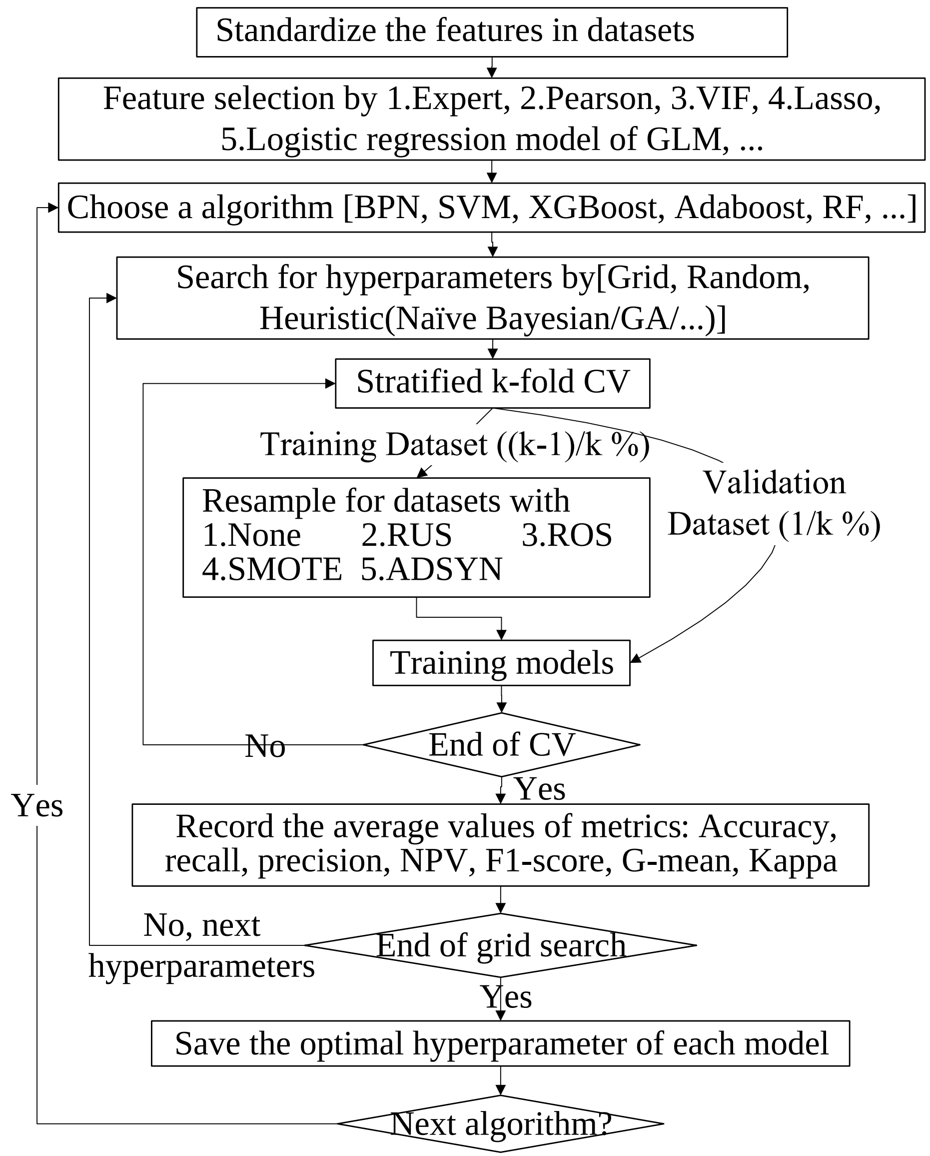
Model Evaluation

Hao-Chun Lu 盧浩鈞
National Yang Ming Chiao Tung University

The Whole Process

- Define the metrics
- Normalization
- Feature selection
- Hyperparameter optimization
- Cross-validation
- Oversampling



Model Evaluation

1 Metrics for Evaluation

2 Model Evaluation

3 Oversampling

4 Learning Curve of Model

5 Hyperparameter Optimization

1 Metrics for Evaluation

- How can we measure **accuracy**?
- Metric[metric] (度量)(or measurement):對一個系統、元件或流程所具有的某個既定的屬性給予一個量化程度的測量
- Qualitative feature: nominal, ordinal scale (discrete) variables
  - 2 classifications: Binary classification, multiclass classification
  - Can be a class, category, code, or state
  - Classification if **y** belong to this scale.
  - Using **confuse matrix**
  - Metric: Accuracy rate, Error rate, Sensitivity, Precision, Specificity[ˌspɛsɪˋfɪsətɪ], F-score, AUC-ROC.
- Quantitative feature: interval, ratio scale (continuous) variables
  - The result is a real number
  - Regression if **y** belong to this scale
  - Metric: Mean squared Error(MSE), R square

1 Metrics in Python

- Metrics in Python
  https://scikit-learn.org/stable/modules/model_evaluation.html
  The scoring parameter: str, defining model evaluation metric
  What's the string in python
- Classification problem
  - 'accuracy': metrics.accuracy_score
  - 'recall' ('recall_micro', 'recall_macro', 'recall_weighted'): metrics.recall_score [sensitivity]
  - 'precision' ('precision_micro', 'precision_macro', 'precision _weighted') : metrics.precision_score (PPV)
  - 'f1' ('f1_micro', 'f1_macro', 'f1_weighted'): metrics.f1_score
  - 'roc_auc' ('roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted', 'roc_auc_ovo_weighted',) : metrics.roc_auc_score
  - Specificity (NPV) 沒提供, 要自己寫
  - Kappa, Gmean: for multiclasses 要找別的package
- Regression problem
  - 'neg_mean_squared_error' : metrics.mean_squared_error (正常MSE乘上負號, 以維持越大越好)
  - 'r2' : metrics.r2_score (R square)

1.1 Metrics for Binary Classification 2

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Accuracy rate** = $\frac{TP+TN}{All}$ 正確率
  - percentage of test set tuples that are correctly classified
- **Error rate** (misclassification rate) = *1* – Accuracy, or $\frac{FN+FP}{All}$

1.1 Metrics for Binary Classification 1

- [**2** classes] Confusion Matrix :

	Predicted class		Total
Actual class	C_1	$\neg C_1$	
C_1	**True Positives (TP)**	**False Negatives (FN)**	**Positive tuples (P)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**	**Negative tuples (N)**

  - Positive tuples: tuple of the main class of **interest**
  - Negative tuples: all other tuples
  - Example

Predicted class / **Actual class**	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**(流失客戶)	7000
buy_computer = no	**412**(浪費時間)	**2588**	3000
Total	7366	2634	10000

1.1 Metrics for Binary Classification 3

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

For ***imbalance*** class

- **Recall = Sensitivity敏感性 = True Positive Rate (TPR)**
  - measure of completeness完整性
  - [2 classes] = $\frac{TP}{TP+FN} = \frac{TP}{P}$
  - % of positive tuples did the classifier label as positive (**真正有病**且被預測出來的%)(**真正買家**且被預測出來的%)
- **Precision精確性 =** Positive predictive values (**PPV**)
  - [2 classes] = $\frac{TP}{TP+FP} = \frac{TP}{P\prime}$
  - % of tuples that the classifier labeled as positive are actually positive?
  - 醫學(PPV):我說有病就有病的%
  - 商業,我說會買就會買的%

1.2 Imbalance Class

Predicted class Actual Class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**(延誤就醫)	300	30.00 (recall)
cancer = no	**140**(白受罪)	**9560**	9700	98.6 (*specificity*)
Total	230 (39.1%) Precision	9770	1000 0	96.4 (*accuracy*)

- When the main class of interest is rare
  - Fraud detection(詐騙): the number of fraud class (positive tuples) is much less than the number of not fraud class (negative tuples)
  - In medical data: the interesting class may be a rare class, such as “cancer=yes”
  - ***Majority of negative class*** vs ***minority of positive class***
- High accuracy rate 96.4%
  - ***majority of negative class*** and ***minority of positive class***
  - low recall rate (90/300=30%) & low precision rate (90/230 = 39.13%)
  - it is **no use** for this clase

1.2 Imbalance Class - Recall vs. Precision

P. A.	C=Y	C=N	Total	
C=Y	280	20	300	93(recall)
C=N	420	9280	9700	95.7(speci)
Total	700	9300	10000	95.6 *acc*
	40(pre)			

P. A.	C=Y	C=N	Total	
C=Y	50	250	300	16(recall)
C=N	5	9695	9700	
Total	55	9945	10000	97.45 *acc*
	91(pre)			

- There tends to be an **inverse relationship between recall and precision**.
  - It is possible to increase one at the cost of reducing the other.
- High recall(93%) and low precision(40%)[420無效醫療]
  - wide the conditions (一點點懷疑就認定是cancer)
- Low recall(16%) and high precision(91%)[250錯失醫療良機]
  - narrow the conditions (要認定是cancer很嚴格)

1.2 Imbalance Class – F1

- ***F* measure** : **Combine Recall & Precision**
- ***F* measure** is a popular metric for imbalanced classification because F-score seeks to the balance of Recall and PPV.
- ***F_1*** or ***F-score***: Equal weight to precision and recall.
  - $F=\frac{(1+1)*Precision\times Recall}{(1*Precision+Recall)}$
- ***$F_ß$***: Weighted measure of precision and recall
  - Assigns **ß** times as much weight **to recall** as to precision (**ß↑→ Recall↑**)
  - $F_ß=\frac{(1+\beta^2)\times Precision\times Recall}{(\beta^2\times Precision+Recall)}$

- Example(embedded excel)

	Can=Y	Can=N	Totel		
Can=Y	280	20	300	93.33%	Recall
Can=N	420	9280	9700	95.67%	Specificity
Totel	700	9300	10000	95.60%	Accuracy
precision	40.00%				
F	56.00%				
F2	0.736842				
F0.5	0.451613				
	Can=Y	Can=N	Totel		
Can=Y	50	250	300	16.67%	Recall
Can=N	5	9695	9700	99.95%	Specificity
Totel	55	9945	10000	97.45%	Accuracy
precision	90.91%				
F	28.17%				
F2	0.199203				
F0.5	0.480769				

1.2 Imbalance Class - Recall vs. Precision vs F1

- TP, TN, FP, and FN are useful in assessing the **costs and benefit** associated with a classification model.
- Medicine(醫療):FN(延誤治療,病死) vs. FP(浪費醫療,沒病當有病,被告)
  - Trend to recall: FN↓ (寬鬆用藥開刀Recall↑ Precision↓)
  - Trend to precision: FP↓ (嚴格用藥開刀Recall↓ Precision↑)
- Loan decisions(貸款): FN(lost business,沒業績) vs. FP(不履行者, 呆帳)
  - Trend to recall: FN↓ (寬鬆授信Recall↑ Precision↓),
  - Trend to precision:FP↓ (嚴格授信Recall↓ Precision↑)

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P’	**N’**	**All**

- Target marketing mailout (郵購): which is high cost?
  - Mailouts to households that do not respond (大量寄Recall↑ Precision↓ , 項多不回應數增多).
  - Lost business from not mailing to households that would have responded (精打細算寄,本來會買的的家庭卻沒寄到Recall↓ Precision↑)
  - Cost vs. Benefit

- Balance accuracy = (Recall + Specificity)/2
- To combine recall and precision into a single measure → F measures.

1.2 Imbalance Class – TNR, NPV, G-mean

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

For ***imbalance*** class

- **Specificity 明確性= True Negative Rate (TNR)**
  - [2 classes] = $\frac{TN}{FP+TN} = \frac{TN}{N}$
  - % of negative tuples did the classifier label as negative (**真正無病**且被預測出來的%)(**真正不買**且被預測出來的%)
- **Negative Predictive Value(NPV)**
  - [2 classes] = $\frac{TN}{FN+TN} = \frac{TN}{N'}$
  - 醫學,我說沒有病就沒有病的%
  - 商業,我說不會買就不會買的%
- ***G-Mean*** : **Combine Recall & Specificity**
  - Sensitivity and Specificity can be combined into a single score that **balances both concerns**, called the geometric mean or G-Mean.
  - G-Mean=$\sqrt{Recall * Specificity}$

1.3 Metrics for Imbalance Classes - kappa 1

- ***Cohen's kappa*** is a metric that represent the level of agreement between **two annotators on a classification problem**. One is **actual class**, another one is the **classifier**. (用來衡量actual class與classifier 2者在classification的認同度level of agreement, level of agreement越高kappa越高)
- Kappa value meaning

Kappa value	meaning
>0.8	Excellent, 2者認同度很高
0.6-0.8	Good
0.4-0.6	Common
<0.4	Poor, 2者認同度很低

- Level of agreement → $\mathrm{k} = (p_o - p_e)/(1 - p_e)$
  - p_o: ***observed accuracy*** (2者認真考量下 毫無爭議的數值)
  - $p_e = \sum_i p_i$: ***expected accuracy*** (2者隨機考量下 的數值)

1.3 Metrics for Imbalance Classes - kappa 2

- $k = (p_o - p_e)/(1 - p_e)$
  - p_o: ***observed accuracy***
  - $p_e = \sum_i p_i$: ***expected accuracy***
- Example
  - p_o = (90+90+1)/210 = 0.8619
    不管Actual class或是classifier都認同的
  - $p_e = p_0 + p_1 + p_2$
    p_0= actual class認同率 * classifier認同率 (class 0隨便選的至少認同率)
    = (90+1+0)/210 * (90+5+5)/210 = 0.2021
    (雙方認同率越接近, p_i越小)
    以下依此類推
    p_1=(1+90+9)/210*(5+90+9)/210=0.2303
    p_2=(0+9+1)/210 *(5+9+1)/210 =0.07
    p_e = 0.4352
  - K=(0.8619-0.4352)/(1-0.4352)=0.75

Predicted class / Actual class	Class 0	Class 1	Class 2	Total
Class 0	90	5	5	100
Class 1	1	90	9	100
Class 2	0	9	1	10
Total	91	104	15	210

1.4 Metrics for Multclass Classification – Recall, Precision

Predicted class / Actual class	Class 0	Class 1	Class 2	Total
Class 0	2	0	0	2
Class 1	1	1	1	3
Class 2	0	2	1	3
Total	3	3	2	8

- Example: 3 classes
- [*3* classes] **confusion matrix:** $CM_{i,i}$ denotes the # of tuples in class *i* that were labeled by the classifier as class *j*

  $CM_{0,0}$=2, $CM_{0,1}$=0, $CM_{0,2}$=0

  $CM_{1,0}$=1, $CM_{1,1}$=1, $CM_{1,2}$=1

  $CM_{2,0}$=0, $CM_{2,1}$=2, $CM_{2,2}$=1

- [m classes] **Accuracy rate** = $\frac{\sum_{i=1}^{m} CM_{i,i}}{All}$=(2+1+1)/8=0.5 (same to binary classfication)
- [m classes] **Recall$_i$**= $\frac{TP\ for\ class\ i}{P\ for\ class\ i}$, **Precision$_i$**= $\frac{TP\ for\ class\ i}{P\prime\ for\ class\ i}$

  Class 0→ Recall$_0$ = 2/2=1, Precision$_0$ = 2/3=0.667, F1$_0$=0.8

  Class 1→ Recall$_1$ = 1/3=0.333, Precision$_1$ = 1/3=0.333, F1$_1$=0.333

  Class 2→ Recall$_2$ = 1/3=0.333, Precision$_2$ = 1/2=0.5, F1$_2$=0.4

1.4 Metrics for Multclass Classification - Specificity

- Example: 3 classes
  - Real: 0 0 1 1 1 2 2 2
  - Predictive: 0 0 0 1 2 1 1 2

Predicted class / Actual class	Class 0	Class 1	Class 2	Total
Class 0	~~2~~	~~0~~	~~0~~	~~2~~
Class 1	~~1~~	1	1	3
Class 2	~~0~~	2	1	3
Total	~~3~~	3	2	8

- [m classes] Specificity= $\frac{All-(P'+P-TP\ for\ class\ i)}{All-P\ for\ class\ i}$

Class 0→Specificity$_0$=(8-2-3+2)/(8-2)=(1+2+1+1)/(3+3)=0.833, G-mean$_0$=0.8

Class 1→ Specificity$_1$ = (8-3-3+1)/(8-3)=0.6 , G-mean$_1$=0.447

Class 2→ Specificity$_2$ = (8-2-3+1)/(8-3)=0.8 , G-mean$_2$=0.516

- [m classes] **NPV**= $\frac{All-(P'+P-TP\ for\ class\ i)}{All-P'\ for\ class\ i}$

Class 0→ NPV$_0$ = (8-2-3+2)/(8-3)=(1+2+1+1)/(3+2)=1

Class 1→ NPV$_1$ = (8-3-3+1)/(8-3)=0.6

Class 2→ NPV$_2$ = (8-2-3+1)/(8-2)=0.667

1.4 Average value for multiple classification 1

- There are 3 average types of recall, precision, specificity, and F1 for multiple classes: **Micro average, Macro average, Weighted average**
  (不然metric有那麼多classes, 不知道要看哪一個class的metric)
- Micro average: 賦予每個sample(tuple) 相同的權重
  Macro average: 賦予每個class同的權重
  Weighted average: 在Macro上, 再考慮 the number of each class不同的加權算法
- Micro average: Accuracy = Recall = Precision = F1(所以不建議使用)
  - 把each class的TP加總 / (總element number) [整體算]
  - Recall = $\frac{TP_1+\cdots+TP_n}{P_1+\cdots+P_n} = \frac{\sum_{i=1}^{m} CM_{i,i}}{All}$
  - Precision = $\frac{TP_1+\cdots+TP_n}{P'_1+\cdots+P'_n} = \frac{\sum_{i=1}^{m} CM_{i,i}}{All}$
  - **Accuracy =** Recall = Precision = F1 = $\frac{\sum_{i=1}^{m} CM_{i,i}}{All}$ = 4/8 = 0.5

1.4 Average value for multiple classification 2

- Macro average
  - Imbalanced data 較看不出差異)
  - All class的metrics加總 / (number of classes) [先算each class在平均]
  - Precision$_{mac}$ = $\frac{\sum_{i=1}^{m} Precision_i}{m}$ =(0.667+0.333+0.5)/3=0.5
  - Recall$_{mac}$ = $\frac{\sum_{i=1}^{m} Recall_i}{m}$=(1+0.333+0.333)/3=0.556
  - F1$_{mac}$ = $\frac{\sum_{i=1}^{m} F1_i}{m}$ =(0.8+0.333+0.4)/3=0.511
  - Specificity$_{mac}$ = $\frac{\sum_{i=1}^{m} Specificity_i}{m}$ =(0.833+0.6+0.8)/3=0.744
  - G-mean$_{mac}$ = $\frac{\sum_{i=1}^{m} G{-}mean_i}{m}$ =(0.9129+0.4472+0.5164)/3=0.625
  - NPV$_{mac}$ = $\frac{\sum_{i=1}^{m} NPV_i}{m}$ =(1+0.6+0.667)/3=0.756

1.4 Average value for multiple classification 3

- Weighted average (考慮每個class tuple不同): Accuracy = Recall
  - 站在Macro average上, 再考慮 the # of each class不同的加權算法
  - $\text{Precision}_w = \frac{\sum_{i=1}^{m} Precision_i * (\#\, of\, tuples\, in\, class\, i)}{\#\, of\, tuples}$
    =(0.667*2+0.333*3+0.5*3)/8=0.479
  - $\text{Accurate=Recall}_w = \frac{\sum_{i=1}^{m} Recall_i * (\#\, of\, tuples\, in\, class\, i)}{\#\, of\, tuples}$
    =(1*2+0.333*3+0.333*3)/8=0.5
  - $\text{F1}_w = \frac{\sum_{i=1}^{m} F1_i * (\#\, of\, tuples\, in\, class\, i)}{\#\, of\, tuples}$ =(0.8*2+0.333*3+0.4*3)/8=0.511
  - $\text{Specificity}_w = \frac{\sum_{i=1}^{m} NPV_i * (\#\, of\, tuples\, in\, class\, i)}{m}$=(0.83*2+0.6*3+0.8*3)/8=0.733
  - $\text{G-mean}_w = \frac{\sum_{i=1}^{m} G-mean_i * (\#\, of\, tuples\, in\, class\, i)}{m}$=(0.83*2+0.6*3+0.8*3)/8=0.5895
  - $\text{NPV}_w = \frac{\sum_{i=1}^{m} NPV_i * (\#\, of\, tuples\, in\, class\, i)}{m}$ =(1*2+0.6*3+0.667*3)/8=0.725

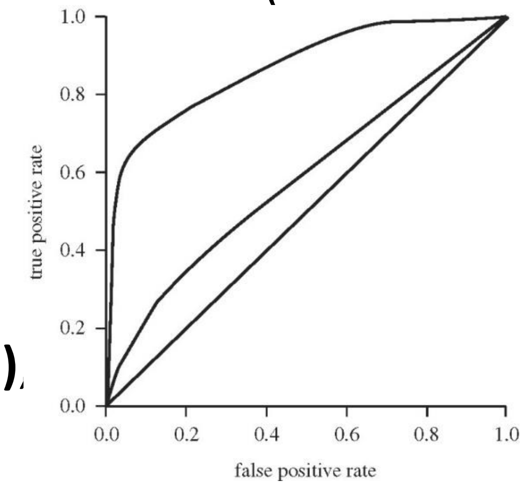
1.5 Practice [06-Model Evaluation.xlsx & 06-1-1-Metrics.py]

Confuse matrix for Multiple Classification

- 06-Model Evaluation.xlsx [Multiclass] & 06-1-1-Metrics.py [20 min]
  - 06-Model Evaluation.xlsx [Multiclass] Calculate the measurements in Excel & compare the result of this python code
  - 06-1-1-Metrics.py: Change the parameter and check all measurements!
    - y_pred = [0,0,1,1,1,2,2,2]
    - y_true = [0,0,0,2,1,1,1,2]
- 06-Model Evaluation.xlsx [Imbalance-class] & 06-1-1-Metrics.py [20 min]
  - 06-Model Evaluation.xlsx [Imbalance-class] Calculate the measurements in Excel & compare the result of this python code
  - 06-1-1-Metrics.py: Change the parameter and check all measurements!
    - y_true = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2]
    - y_pred = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,2]

1.6 AUC-ROC

- Can use in binary, multiclass, and multilabel classifications
- **AUC-ROC :** Area under the Receiver Operating Characteristic Curve (AUC-ROC) from prediction scores.
  - 在目前threshold(對正確機率的要求)下
  - All samples對Positive的正確率(**true positive rate)**, Recall (TPr=TP/P))→ y (Vertical axis)
    (越高越好, 會使curve往上)
  - All samples對Negative的錯誤率(**false positive rate)**, 偽陽性率(FPr=FP/N) → x (Horizontal axis)
    (越低越好,會使curve往右移)
  - 將(x,y)畫出來
- AUC-ROC同時考慮**高正確率(Recall, TPr)** & **低錯誤率(FPr)**
- Under the same FPr,
  higher TPr is better than the lower TPr
  在相同錯誤**率**(N class)下, 已累計正確**率**(P class)有多少?
  因為都是"**率**"所以**不會因為imbalanced data而產生比例問題**
  (Curve越往上衝越好)



A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

1.6 Draw the AUC-ROC 1

- Ex. 10 testing tuples that 5 tuples are positive(1) and 5 tuples are negative(0)

1. **Sorts** the test tuples by the decreasing order of the probability of the predicted class.
2. Tuple 1: threshold =0.9. → Tuple 1 is P. Others are N (tuples 2 - 10).
   → Actual class label of tuple 1 is P
   → TP=1, FP=0, FN=5-1=4, TN=5. TPr=TP/P=0.2, and FPr=FP/N=0.
   → Have the point (0.2,0) for the ROC curve.
3. Tuple 2: threshold = 0.8. → Tuples 1- 2 are P. Others are N.
   → Actual class label of tuples are (P, P).
   → TP=2, FP=0, FN=5-2=3, TN=5.
   → Have the point (0.4,0).
4. Tuple 3: threshold = 0.7.
   Tuples 1-3 are P. Others are N.
   → Actual class label of tuples = [P, P, N].
   → TP=2, FP=1, FN=5-2=3, and TN=5-1=4.
   Have the point (0.4,0.2).

actual ↗ sorting

Tuple #	Class	Prob.	TP	FP	TN	FN	TPR	FPR
1	P	0.90	1	0	5	4	0.2	0
2	P	0.80	2	0	5	3	0.4	0
3	N	0.70	2	1	4	3	0.4	0.2
4	P	0.60	3	1	4	2	0.6	0.2
5	P	0.55	4	1	4	1	0.8	0.2
6	N	0.54	4	2	3	1	0.8	0.4
7	N	0.53	4	3	2	1	0.8	0.6
8	N	0.51	4	4	1	1	0.8	0.8
9	P	0.50	5	4	0	1	1.0	0.8
10	N	0.40	5	5	0	0	1.0	1.0

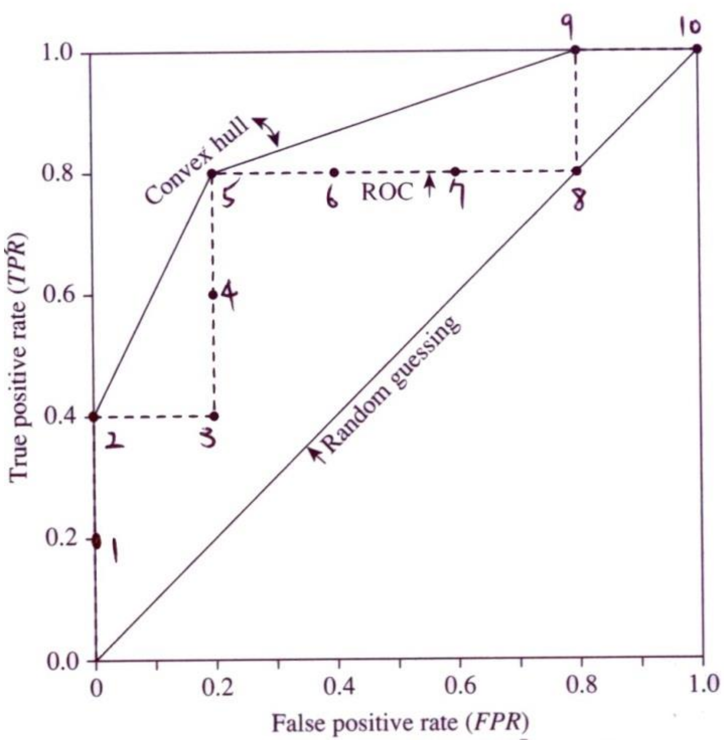
	Y	N	
Y	1	4	5
N	0	5	5

$TPR = \frac{TP}{P} = 0.2$

$FPR = \frac{FP}{N} = 0$

1.6 Draw the AUC-ROC 2

4. Continue the process until all tuples have been calculated.
   (0,0) → (0.2, 0) → (0.4, 0) → (0.4, 0.2) → (0.6, 0.2) → (0.8, 0.2) → (0.8, 0.4) → (0.8, 0.6) → (0.8, 0.8) → (1, 0.8) → (1,1). [program會簡化這些點]
5. So the ACU-ROC = [對角線]與 [ROC curve] 包的面積*2
   (0.5 - 0.12(沒有填滿的部分))*2 = 0.38*2 = 0.76



actual ↗sorting

Tuple #	Class	Prob.	TP	FP	TN	FN	TPR	FPR
1	P	0.90	1	0	5	4	0.2	0
2	P	0.80	2	0	5	3	0.4	0
3	N	0.70	2	1	4	3	0.4	0.2
4	P	0.60	3	1	4	2	0.6	0.2
5	P	0.55	4	1	4	1	0.8	0.2
6	N	0.54	4	2	3	1	0.8	0.4
7	N	0.53	4	3	2	1	0.8	0.6
8	N	0.51	4	4	1	1	0.8	0.8
9	P	0.50	5	4	0	1	1.0	0.8
10	N	0.40	5	5	0	0	1.0	1.0

1.6 ROC AUC – multiclass classification 1

- 假設class number = 3, class 0, 1, 2其樣本數分別為 n_0, n_1, n_2 ($n = n_0+n_1+n_2$)
- multi_class = {‘ovr’, ‘ovo’} →multi-class不能畫圖
  - ‘ovr’: One-vs-rest 共執行n次, based on Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861-874.
    - 提供None / macro / micro / weighted
    - 1st : 把test set分為class 0, others, 用binary classification計算R_0
    - 2nd : 把test set分為class 1, others, 用binary classification計算R_1
    - 3rd : 把test set分為class 2, others, 用binary classification計算R_2
    - None: 分別印出 R_0, R_1, R_2
    - Macro average: $(R_0+R_1+R_2)/3$
    - Micro average: $TPr = (TP_1+TP_2+TP_3)/(TP_1+TP_2+TP_3+FN_1+FN_2+FN_3)$
      $FPr = (FP_1+FP_2+FP_3)/(FP_1+FP_2+FP_3+TN_1+TN_2+TN_3)$
    - Weighted average = $(R_0*n_0+R_1*n_1+R_2*n_2)/n$
    - Micro-averaged OvR ROC is dominated by the more frequent class, since the counts are pooled.
    - Macro-averaged alternative better reflects the statistics of the less frequent classes, and then is more appropriate when performance on all the classes is deemed equally important.

1.6 ROC AUC – multiclass classification 2

- 假設class number = 3, class 0, 1, 2其樣本數分別為 n_0, n_1, n_2 ($n = n_0+n_1+n_2$)
  - 'ovo': One-vs-one, 共執行n(n-1)/2次, based on Hand, D.J., Till, R.J. (2001). A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. Machine Learning, 45(2), 171-186.
    - 只提供macro/weighted
    - For class pair (0,1): 把test set分為class 0, class 1, 用binary classification計算R_{01} = (R for (0,1) + R for (1,0))/2
    - For class pair (0,2): 把test set分為class 0, class 2, 用binary classification計算R_{02} = (R for (0,2) + R for (2,0))/2
    - For class pair (1,2): 把test set分為class 1, class 2, 用binary classification計算R_{12} = (R for (1,2) + R for (2,1))/2
    - Macro average = $(R_{01}+R_{02}+R_{12})/3$
    - Weighted average = $(R_{01}*n_{01}+R_{02}*n_{02}+R_{12}*n_{12})/n$, $n_{ij}=(n_i+n_j)/2$

1.6.1 ROC AUC – python code 1

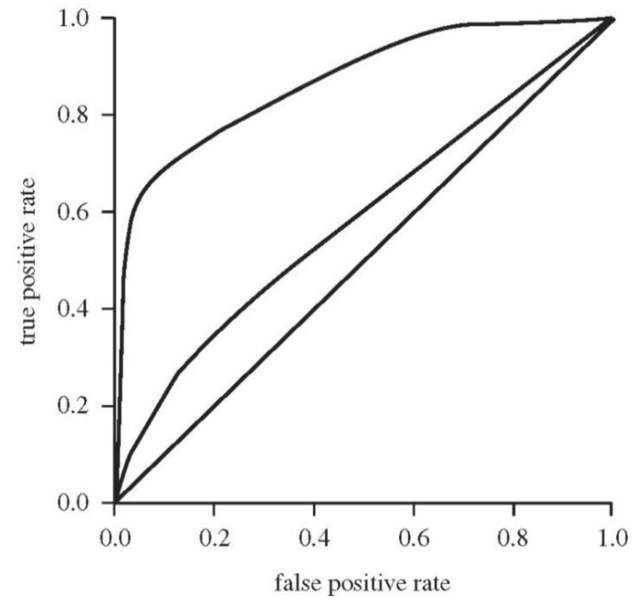
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics, model_selection, datasets
from sklearn.svm import SVC
# 1. 學習驗證用
#    事先準備好 test set 的 real y 與 算出來的機率p
#    只能用於 binary classification
y = np.array([1,   1,   0,   1,   1,    0,    0,    0,    1,   0])
p = np.array([0.9, 0.8, 0.7, 0.6, 0.55, 0.54, 0.53, 0.51, 0.5, 0.4])

#    用 y, p 算出 fpr, tpr, thresholds, pos_label用來指定 Positive是 0 還是 1
fpr, tpr, thresholds = metrics.roc_curve(y, p, pos_label=1)

#    用 fpr, tpr 算出 auc roc
roc  = metrics.auc(fpr, tpr)
print('auc roc={}\n'.format(roc))
plt.figure()
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc, pos_label=1)
display.plot()
```

1.6.1 ROC AUC – python code 2

```
# 2. 實際做法
# 2.1 Load dataset
X, y = datasets.load_iris(return_X_y=True) #鳶尾花數據集：150筆花朵樣本
#X, y = datasets.load_breast_cancer(return_X_y=True)

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, random_state=0)

# 2.2 使用 training set to train the model
clf = model = SVC(kernel='rbf', C=10, probability=True)
clf.fit(X_train, y_train)

# 2.3 使用 test set to calculate the metric
#    roc = metrics.roc_auc_score( y_test, 計算出來的prob)
#    disp = metrics.plot_roc_curve( model, X_test, y_test) only for binary classification
if len(clf.classes_)==2:
   roc = metrics.roc_auc_score(y_test, clf.predict_proba(X_test)[:,1])#雙類別只要Pos之 prob.
   metrics.RocCurveDisplay.from_estimator(estimator=clf, X=X_test, y=y_test, pos_label=1)
else:                                        #多類別要全部class之prob.
   roc = metrics.roc_auc_score(y_test, clf.predict_proba(X_test), average='macro', multi_class='ovo')
```

1.6.2 Practice [06-1-2-ROC.py]

- 06-1-2-ROC.py [20 min]
  - The diagonal line (TPR=FPR for each tuple) represents random guessing.
  - The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model.
  - The area under the ROC curve is a measure of the accuracy of the model.
  - 1.學習驗證用 (要配合自己手算)
  - 2.實際做法
    - Try the 2 classifications:
      X, y = datasets.load_breast_cancer (binary classification)
      X, y = datasets.load_iris (multiclass classification)
    - average=None / 'macro' / 'mocro' / 'weighted', multi_class='ovr'/ 'ovo'



1.7 Metrics for Continuous value

- Regression → the result is continuous value.
- Mean squared error (MSE)
  - MSE= $(1/\mathrm{N})\ \sum_{n=1}^{N}\left(T^{(n)}-O^{(n)}\right)^2$
  - MSE will be influenced by the range of $O^{(n)}$
  - MSE 的大小沒有意義, 只能比相同dataset下, 哪一個model好
- R square
  - $R^2 = 1 - SSE/SST$
    - SSE(Sum of Squared Error) = $\sum_{n=1}^{N}\left(T^{(n)}-O^{(n)}\right)^2$
      預測值T與實際值O的誤差平方 → N*MSE **(Bias)**
    - SST(Sum of Squared Total = $\sum_{n=1}^{N}\left(O^{(n)}-\bar{O}\right)^2$
      實際值O與平均值$\bar{O}$的誤差平方 **(Variance)**
    - SST越小(真實變動越小), SSE也要越小(較好預測)
  - R^2 越接近1越好
    - 1完全相關(SSE=0)
    - 0完全無關(SSE=SST, 預測出來跟取mean一樣)
    - <0完全沒有用處(SSE>SST, 預測效果比用mean來猜測還要糟糕！)
    - 社會科學: R^2常見為0.5~0.6 (>0.7 算不錯)
    - 生物或農業: R^2大於0.9也是不多
    - 儀器效正: R^2通常為0.999

1.8 Additional Issues

- Current assumption: each training tuple can belong to only one class
  - It is more probable to assume that each tuple may belong to ***more than one class***. → accuracy is not appropriate!
  - It is useful to return a ***probability class distribution*** than returning a class label.
  - **Probability → softmax function → class**
- Additional Aspects:
  - **Accuracy**
    - classifier accuracy: predicting class label
  - **Speed**
    - time to construct the model (training time)
    - time to use the model (classification/prediction time)
  - **Robustness**: handling **noise and missing** values
  - **Scalability**: efficiency in large amounts of data
  - **Interpretability**
    - understanding and insight provided by the model

Model Evaluation

1 Metrics for Evaluation

2 Model Evaluation

3 Oversampling

4 Learning Curve of Model

5 Hyperparameter Optimization

2.1 Model Evaluation 1

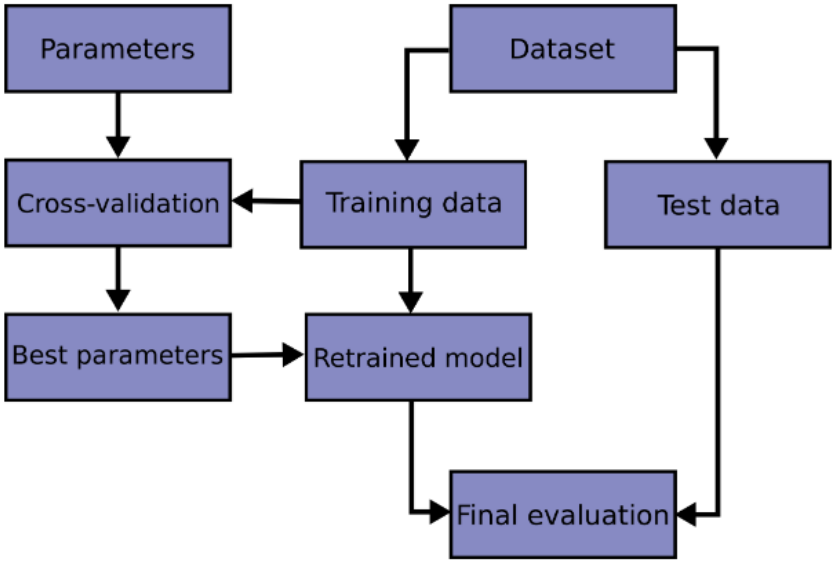
- Dataset will be divided 3 parts: training set, validation set, test set.
  - Training set: Construct Model
  - Validation set: will be used many times
    - Machine learning: **tuning model hyperparameters** with **cross-validation**.
    - Deep leaning (Neural network): (i) evaluating model with short period, (ii) check overfitting
  - Test set: only use one time.
    - Evaluate the final model
  - **Test set/validation set** that are **not used to train the model**.
  - What’s the different between validation set and test set?
    - 小考 vs 期末考

2.1 Model Evaluation 2

1. Construct the Model
   - Training set: construct the model
   - Validation set: evaluate the model.
   - **Learning Curve**: to check the size of training set is **enough** or not?
     (x-axis → the size of training set)
2. Monitor(evaluation) and tune the Model
   - Machine learning: **tuning model hyperparameters** with **cross-validation**.
     - If the measurements are acceptable: Obtain the appropriate **hyperparameters of the model.**
   - Deep leaning (Neural network): (i) evaluating model with short period, (ii) check **overfitting** based on **the trend chart of loss function**.
     (x-axis → the training iteration)
     - If overfitting is happening: stop the training process (drop the other epochs)
3. **Final test the Model**: Get the measurements of the final model
   - Build the model with the **training + validation sets** (**refit**).
   - Evaluate the model with the test set.
4. Use the model to classify/predict the unknown data tuples

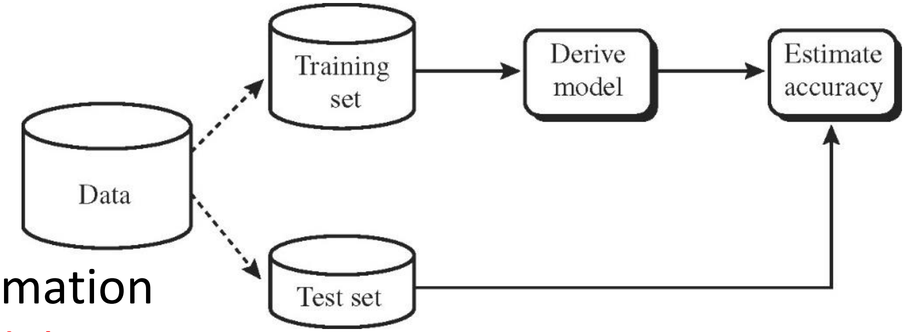
2.1 Model Evaluation 3

- Evaluation the Model
- Training and test sets: evaluating the Final Model
  - Holdout method(保留法) → no use!!!
- Training, validation, and test sets
  - Deep leaning (neural network based approach)
  - No CV since the computing time is too long
- Training, validation, and test sets with CV: tuning model hyperparameters
  - Machine learning (not in neural network)
  - *k*-fold cross-validation(k-CV, *k*-疊交叉驗證法)
  - Stratified K-Fold cross-validation
  - Leave-one-out cross-validation (LOC-CV)
- Comparing classifiers: Confidence intervals



2.2 Holdout

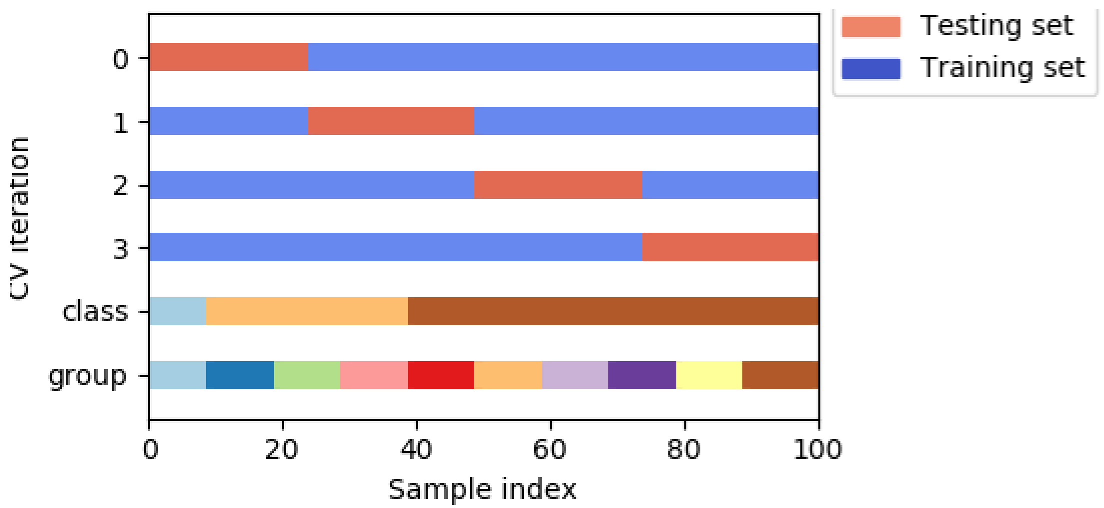
- **Holdout method** (保留法): Given data is randomly partitioned into two independent sets
  - Large dataset.
  - Training set (e.g., 2/3) for model construction
  - Test set (e.g., 1/3) for accuracy estimation
  - Can use in Evaluating the Final Model
  - Drawback: It is suited for **large** data sets




```
#   train_test_split 傳入 X,y 傳回分好的 training & test
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, \
                                        test_size=0.3, random_state=0)
clf.fit(X_train, y_train)
```

2.3 *k*-fold cross-validation 1

- ***k*-fold cross-validation (*k*-CV)**
  - It is suited for **medium** datasets.
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size: D→ D_1, D_2, …, D_k.
  - At *i*-th iteration, use **D_i as validation set** and others as training set for *i*=1,…,*k*.
  - Each tuple is used the same # of times for training and validation.
  - Accuracy of training = (overall # of correct classifications from the k iterations) / (# of tuples in the initial data).
- Example for *k*=4: KFold is not affected by classes

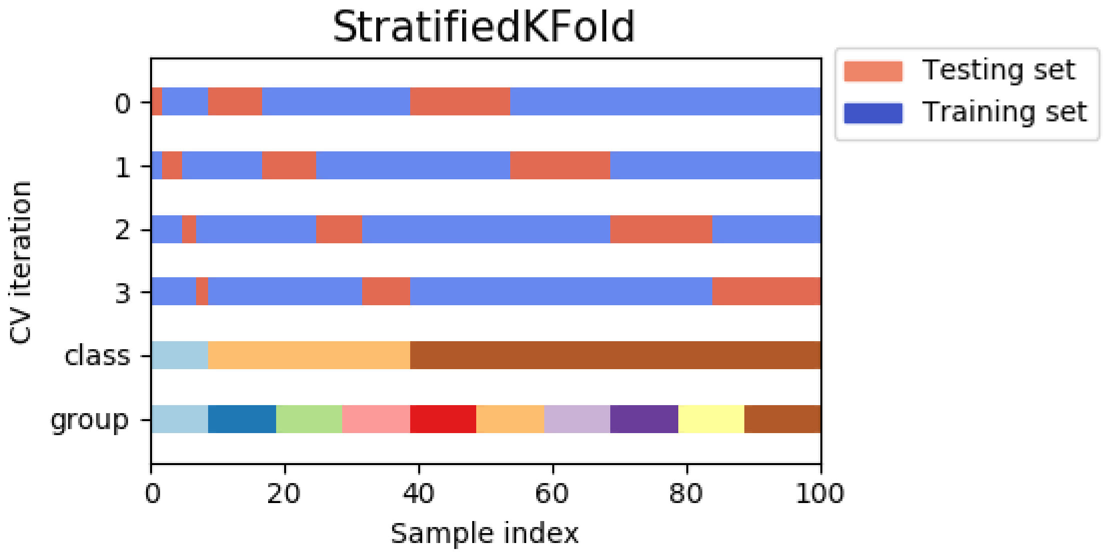


2.3 *k*-fold cross-validation 2

- The number of ***k*** in k-CV**:**
- When k is large
  - (Strengths) The bias of estimator is low (每個fold之tuples少)
  - (Weaknesses) The variance of estimator is high (fold數多)
  - (Weaknesses) Expensive computing cost.
- When k is small
  - (Weaknesses) The bias of estimator is high
  - (Strengths) The variance of estimator is low
  - (Strengths) Efficient computing.
- *k* = 10 is most popular

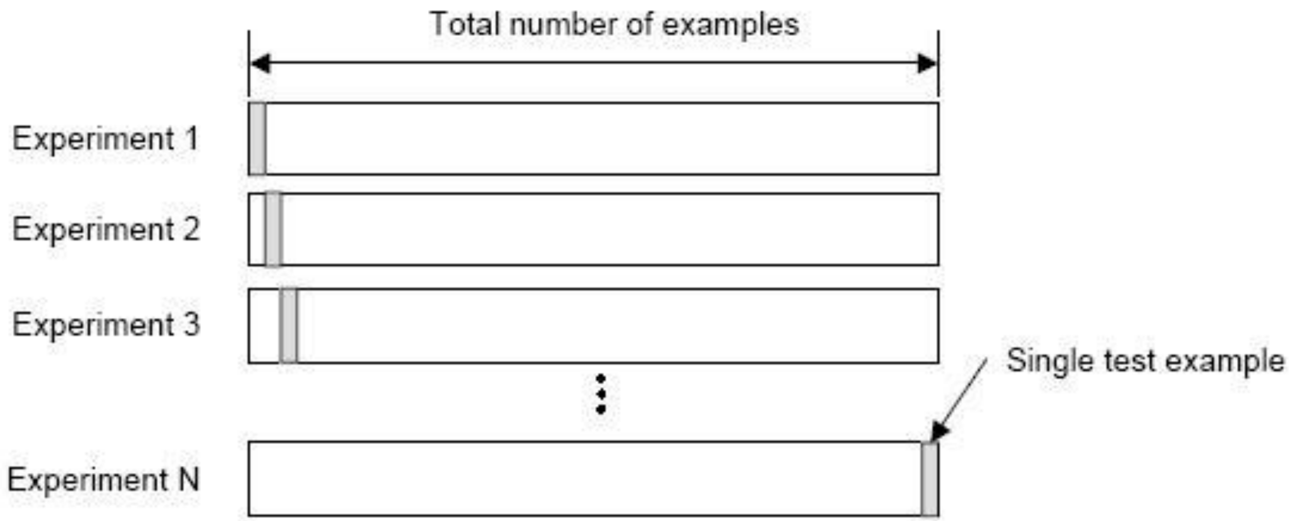
2.4 Stratified K-Fold

- Stratified K-Fold: 類似K-Fold, 但是它是分類別sampling, 確保training set/validation set中, 每個類別的比例相同
- Example for *k*=4: Stratified K-Fold is affected by classes



2.5 LOC cross-validation

- **Leave-one-out cross-validation (LOC-CV)**: a special case of *k*-CV
  - It is suited for **small datasets**.
  - *k* = # of tuples: Only 1 tuple is "left out" at a time for the test set.



2.6 CV Python Code 1

```
# 2. Cross-validation (CV), most popular method for validation
#   用了CV metric往下降是正常, 比較不會overfitting
#   cross_val_score/cross_validate 傳入model,X,y 然後function會搞定一切,
#   只傳回訓練好的 metric result
#   cross_val_score 1次只用1個metric
CV1 = cross_val_score(clf, wine.data, wine.target, cv=5, scoring='accuracy')
print("CV with k=5 [Accuracy]\n%s Mean=%f\n" % (CV1, np.mean(CV1)))
print("CV with k=5 [F1 Weighted]\n%s Mean=%f\n" % (CV2, np.mean(CV2)))

#   cross_validate 1次可用多個metrics
scoring = ['accuracy', 'f1_weighted']
CV = cross_validate(clf, wine.data, wine.target, cv=5, scoring=scoring, \
        return_train_score=False)

for cv1 in CV:
  print('%s %s Mean=%f' % (cv1, CV[cv1], np.mean(CV[cv1])))
```

2.6 CV Python Code 2

```
# 3  使用 CV 切分器 (必須自己訓練)
#    對 wine dataset 使用 StratifiedKFold 實戰
#    StratifiedKFold 傳入X,y後, 傳回分好 tuple 的 indexes
CV=5
res1 = list(np.zeros([CV]))
#   實戰時打開 shuffle=True 且設定 random_state=0
sskf = StratifiedKFold(n_splits=CV, shuffle=True, random_state=0)
i1=0
for train, validation in sskf.split(wine.data, wine.target):
    clf.fit(wine.data[train,:], wine.target[train])
    # 傳回預測結果
    y_pred = clf.predict(wine.data[validation])
    # 傳入 y lable與預測結果, 計算 accuracy, 在此可以使用各種 metric
    res1[i1]=accuracy_score(wine.target[validation], y_pred)
```

2.7 Practice [06-2-Evalution.py] - CV

- 06-2-Evalution.py [30 min]
- Load wine dataset, using SVM as classifier
- 1. Holdout: 70%, 30%. [test set] [自己訓練]
- 2. Cross-validation (CV), most popular method for validation
  - Change the “random_state=0 or 1 or 5” and compare it again!
  - What’s the differences between cross_val_score(single metric) & cross_validate(multiple metric)
  - cross_validate: fit_time, score_time, metric1, metric2, …
- 3.1 使用 CV 切分器
  - K-Fold, StratifiedKFold, LeaveOneOut ,看一下 tuple的順序
- 3.2 使用 StratifiedKFold 切分器 實戰[自己訓練]
  - 算出 accuracy, F1, ROC

Model Evaluation

1 Metrics for Evaluation

2 Model Evaluation

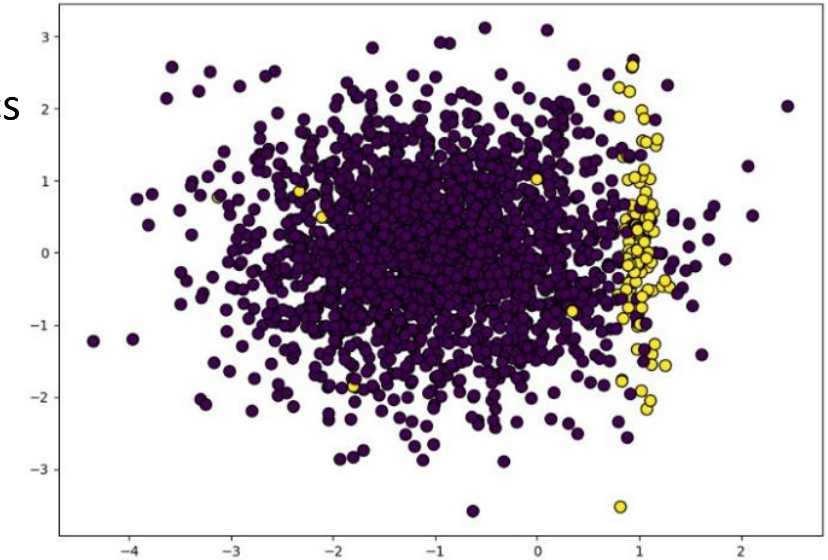
3 Oversampling

4 Learning Curve of Model

5 Hyperparameter Optimization

3.1 Oversampling - ROS

- Imbalanced data
  - Difficult to train for obtain the good metrics
  - Cannot use accuracy
  - No tuples for splitting as training, validation, and test
- Oversampling is a popular technique for treating imbalanced data to avoid the above issues
  - It increases the # of samples of the smaller-sized categories so that the sample sizes are consistent across all categories.
- Random oversampling (ROS): **Randomly sample** the tuples in the categories of smaller sample sizes.




```
Samples in class 0: 1893
Samples in class 1: 107
```

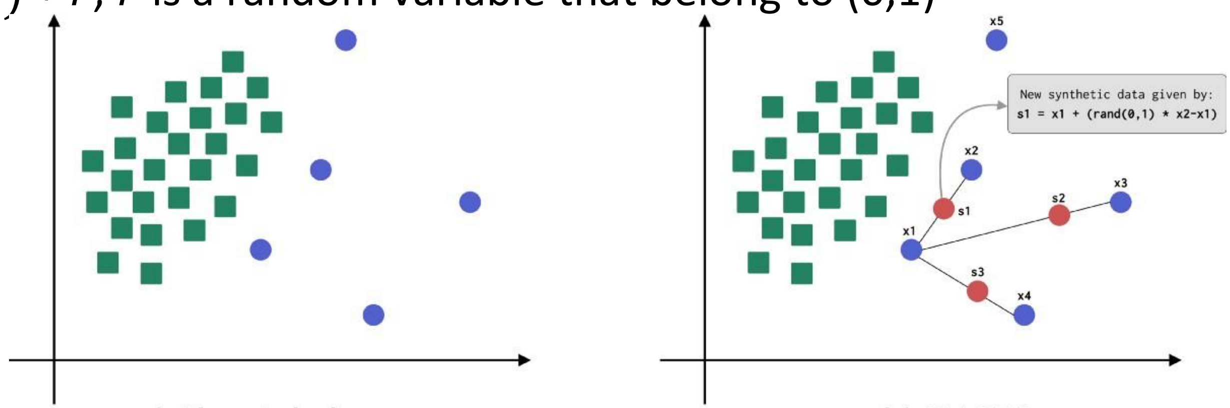
=RandomOverSampler(sampling_strategy='not majority', random_state=0)

sampling_strategy: float, str, dict or callable, default='auto'

'not majority': resample all classes but the majority class

3.2 Oversampling – SMOTE 1

- Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, **16**, 321-357. → Synthetic minority oversampling technique (SMOTE)
- Training set: T= P+N. |T|=pnum+nnum=num
  P={$\mathbf{p}_1$, $\mathbf{p}_2$,…,$\mathbf{p}_{pnum}$} are the minority class.
  N={$\mathbf{n}_1$, $\mathbf{n}_2$,…,$\mathbf{n}_{nnum}$} are the majority class.
- [SMOTE] : All features are continuous
  Each sample $\mathbf{p}_j \in P$ (j=1,…,pnum) will generate s=(nnum-pnum)/pnum synthetic.[Fixed]
  For each sample $\mathbf{p}_j \in P$, SMOTE selects its ***k* minority class** nearest neighbors PN_j.
  Repeat s times :
  - Select one sample $\mathbf{p}' \in PN_j$ and create a new individual $\mathbf{p}''$
  - $\mathbf{p}''$:= **linear combination of $\mathbf{p}_j$** and $\mathbf{p}'$.
    $\mathrm{p}''_i = \mathrm{p}_{j,i} + (\mathrm{p}_{j,i} - \mathrm{p}'_i) * r$, r is a random variable that belong to (0,1)
- **Syntheic→**
  must standardize the features first
  Why?



a) Class Imbalance b) SMOTE

3.2 Oversampling – SMOTE 2

- [SMOTENC] : Some features are nominal
  - For each sample $\mathbf{p}_j \in P$, how to select its ***k* minority class** nearest neighbors PN_j.
    - Median computation: Compute the **median** of **standard deviations of all continuous features** for the minority class.
    - Nearest neighbor computation: If the nominal features differ between a sample $\mathbf{p}_j$ and its nearest neighbors p'_j, then this median is added in the Euclidean distance for penalty
    - Example: $\mathrm{a1}$ $\mathrm{a2}$ $\mathrm{a3}$ $\mathrm{a4}$ $\mathrm{a5}$ $\mathrm{a6}$

      $\mathbf{p}_j$ = (1, 2, 3, A, B, C)

      Std= (0.5, 1, 1.3, n/a, n/a, n/a) # So median = 1

      $\mathbf{p}' \in PN_i$ = (4, 6, 5, A, D, E)

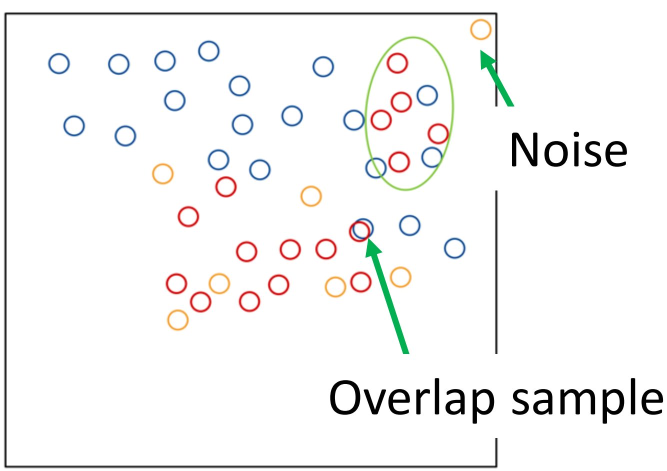
      Euclidean Distance between $\mathbf{p}_j$ and $\mathbf{p}'$

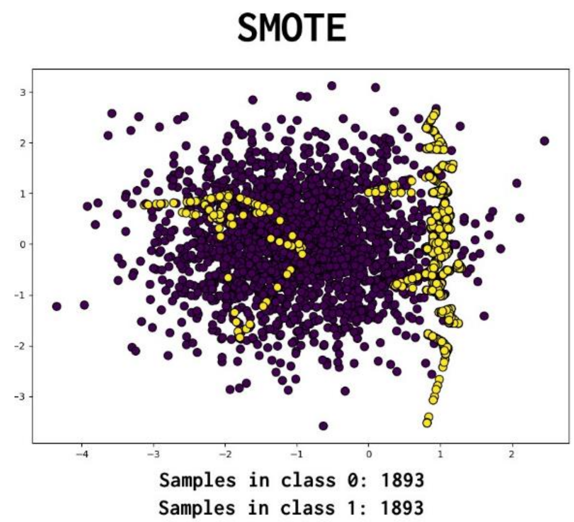
      = ((4-1)^2 + (6-2)^2 + (5-3)^2 + 1^2(median^2) + 1^2(median^2))$^{0.5}$

      median^2 + median^2 → use median to penalize the difference of nominal features
  - The nominal feature is given the value occurring in the majority of the k-nearest neighbors. (多數決? 是否取 $\mathbf{p}_j$ and $\mathbf{p}'$)

3.2 Oversampling - SMOTE 3

- 2 issues in SMOTE
- Synthetic from a noise sample
  - $\mathbf{p}_j$ and $\mathbf{p}'$ is a noise sample
  - The synthetic example is useless (反而干擾)
  - How to check $\mathbf{p}_j$ and $\mathbf{p}'$?
- Overlapping sample
  - If $\mathbf{p}_j$ and $\mathbf{p}'$ are too close, the new generating sample will overlap with either $\mathbf{p}_j$ and $\mathbf{p}'$.
  - How to select its k-nearest neighbor samples → Distance > a threadhold





3.2 Oversampling - SMOTE 4

for Continuous features.

=SMOTE(sampling_strategy='auto' , k_neighbors=5, random_state=None)

for nominal & continuous features.

=SMOTENC(categorical_features, *, sampling_strategy='auto', k_neighbors=5, random_state=None)

categorical_feature: sarray-like of shape (n_cat_features,) or (n_features,)

[1,2,…] 哪些indexes是nomial features

sampling_strategy: float, str, dict or callable, default='auto'

'not majority': resample all classes but the majority class

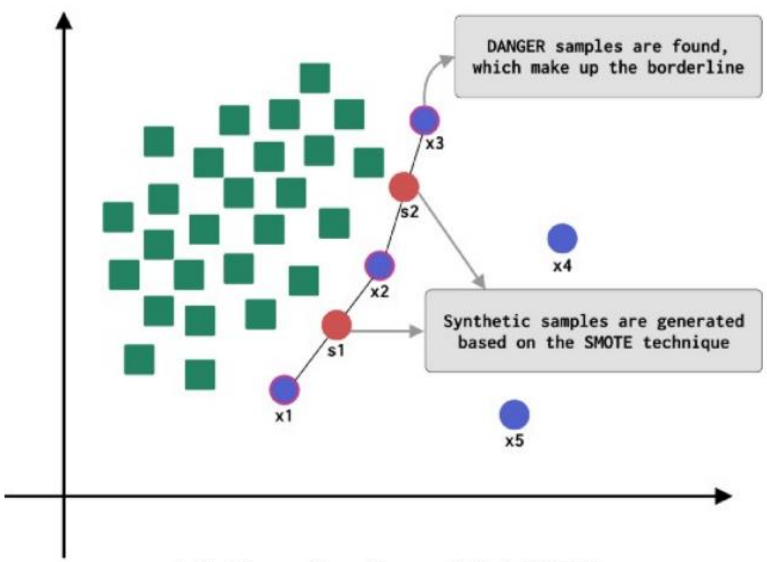
k_neighbors: int or obj, default=5 (find the # of nearest samples **of the minority sample**, which are used to generate the synthetics)

int: the # of neighbors to use

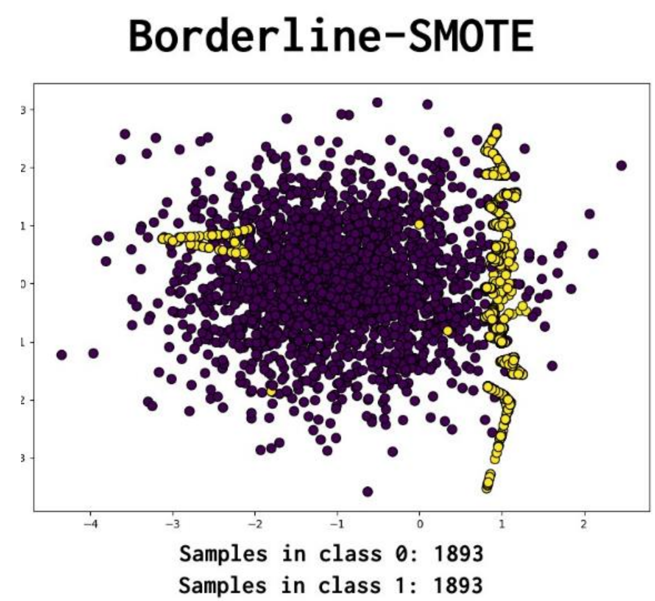
obj: an instance of a compatible nearest neighbors algorithm

3.3 Oversampling – Borderline SMOTE 1

- H. Han, W. Wen-Yuan, M. Bing-Huan. Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. *Advances in intelligent computing*, 878-887, 2005, Springer.
- Unlike the SMOTE, Borderline-SMOTE focuses on generating synthetic data by considering **only samples that make up the border that divides one class from another**.
- Borderline-SMOTE detects which samples are on the border of the class space and applies the SMOTE technique to these samples.

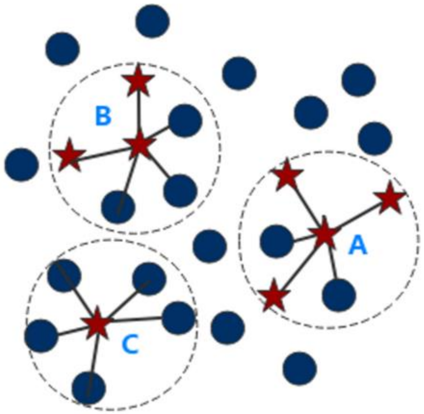


b) Borderline-SMOTE



3.3 Oversampling – Borderline SMOTE 2

- Borderline SMOTE
  - Divide the samples of minority class into 3 types: Safe, Danger, Noise
    →Only oversampling the **Danger** type.
  - Training set: T= P+N
    P={$\mathbf{p}_1$, $\mathbf{p}_2$,…,$\mathbf{p}_{pnum}$} are the minority class.
    N={$\mathbf{n}_1$, $\mathbf{n}_2$,…,$\mathbf{n}_{nnum}$} are the majority class.



- Obtain the **DANGER** individuals
  - For each $\mathbf{p}_j$ (i=1,…,pnum), select ***m*** nearest neighbors.
  - There are ***m'*** majority class samples among the ***m*** nearest neighbors
  (a) **0≤*m'*<0.5*m* :** $\mathbf{p}_j$ is **safe**
  (b) **0.5*m* ≤*m'*<*m*** : $\mathbf{p}_j$ to be easily misclassified majority class, $\mathbf{p}_j$ is **DANGER**
  (c) **m'= *m***: $\mathbf{p}_j$ is **noise**
  DANGER = {$\mathbf{p}'_1$, $\mathbf{p}'_2$,…, $\mathbf{p}'_{dnum}$}, 0≤ dnum≤ pnum
- How to find nearest neighbors?
  1. 採用歐幾里得空間分布#觀察資料設計圈的大小
  2. 採用KNN尋找週圍個數判斷

3.3 Oversampling – Borderline SMOTE 3

- Synthetic new sample: Borderline-SMOTE1
  - Need to generate s*dnum synthetic minority examples
  - For each **p**'$_d$(d=1,…,dnum), select **s** nearest neighbors ($\mathbf{DN}_d$) from its **k** minority class nearest neighbors.
  - [s times] Create a new example **p**'' based on the **linear combination of p**'$_d$ and **p**'$\in \mathrm{DN}_d$.

    $\mathrm{p}''_i = p'_{d,i} + (p'_{d,i} - \mathrm{p}'_i) * r$, r is a random variable that belong to (0,1), for all i =1,…,n; d=1,…,dnum.
- Synthetic new sample: Borderline-SMOTE2
  - For each **p**'$_d$(d=1,…,dnum), select **s** nearest neighbors ($\mathbf{DN}_d$) from its **k** nearest neighbors that **can be minority or majority classes**.
  - If **p**'$\in \mathrm{DN}_d$ is come from N, r is a random variable that belong to (0,0.5), thus the new example **p**' is closer to P.
- ***m > k > s***
  - ***m***: the nearest neighbors of **p**$_j$, which use to decide the DANGER
  - ***k***: the nearest neighbors $\mathbf{DN}_d$, which are the candidates for generating the synthetic minority examples
  - ***s***: the # of synthetic examples for each p'$_d$

3.3 Oversampling – Borderline SMOTE 4

=BorderlineSMOTE(sampling_strategy='auto', random_state=None, k_neighbors=5, m_neighbors=10, kind='borderline-1')

sampling_strategy: float, str, dict or callable, default='auto'

'auto'='not majority': resample all classes but the majority class

*m*_neighbors: int or object, default=10: 對minority sample挑選周圍*m*個 nearest neighbors來決定是否為DANGER

*k*_neighbors: int or obj, default=5: 對DANGER sample, 要挑選周圍k個nearest neighbor candidates來產生synthetic examples.

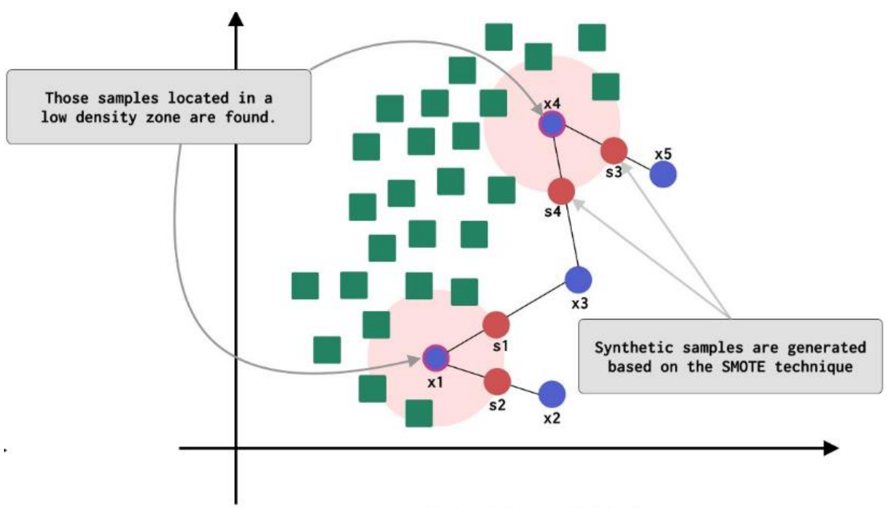
kind{"borderline-1", "borderline-2"}, default='borderline-1':

borderline-1: *k* nearest neighbor candidates come from P

borderline-2: *k* nearest neighbor candidates come from {P, N}

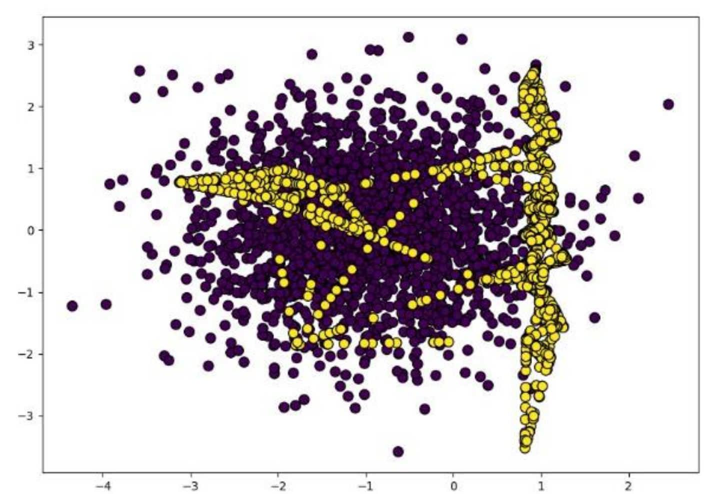
3.4 Oversampling - ADASYM 1

- He, H., Bai, Y., Garcia, E.A., Li, S. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. **IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence**). Pp.1-8. June, 2008. Hong Kong, China.
- The difference between **ADASYN** and **SMOTE**:
  Depend on the ratio ri = (# of majority)/(k nearest neighbors) to generate synthetics **in the lower density areas of the minority class**.
- 對sample with minority class而言, neighbors majority samples越多, generate synthetics越多 → 是低密度區卻又深入敵方



b) ADASYN

ADASYN



Samples in class 0: 1893
Samples in class 1: 1884

3.4 Oversampling – ADASYM 2

1. d = pnum/nnum. (imbalance比例)   [Ex. 0.3 = 3/10]
   if d < dth (threadhold for the max. tolerated degree of class imbalance ratio), then do ADASYM.
2. G = (nnum-pnum)β, β ∈[0,1] [要generate幾個 synthetics] [Ex. G=7, β=1]
3. For each $\mathbf{p}_j$(i=1,…,pnum), select ***k*** nearest neighbors (N_j) based on Euclidean distance.
   $r_j = \frac{\Delta_j}{k}$, Δ_j the # of samples ∈ N in N_j, $r_j \in [0,1]$. ($\mathbf{p}_j$被majority samples包圍的比重)
   - Majority sample↑ → $\triangle_j$ ↑ → r_j↑
   - Ex. r_1=1/5, r_2=2/5, r_3=0/5
4. $\hat{r}_j = \frac{r_j}{\sum_{i=1}^{pnum} r_i}, j = 1, \ldots, pnum$ [normalize to $\sum_{i=1}^{pnum} \hat{r}_j = 1$].
   - Ex. $\hat{r}_1$=1/3,$\hat{r}_2$=2/3
5. For each $\mathbf{p}_j$ will generate $\hat{r}_j$*G synthetics. [$\hat{r}_j$*G times]
   Create a new example **p''** based on the **linear combination of** $\mathbf{p}_j$ and **p'**∈ PN_j.
   $p''_i = p_{j,i} + (p_{j,i} - p'_i) * r$, r is a random variable that belong to (0,1), for all i =1,…,n; j=1,…,pnum.
   - Ex. $\mathbf{p}_1$ generate 7*1/3=2.333=2, $\mathbf{p}_2$ generate 7*2/3=4.666=5.

缺點:

- When $\triangle_j$ → 1, then $\mathbf{p}_j$ may being a noise. → Generating noise examples.

3.4 Oversampling – ADASYN 3

=ADASYN(sampling_strategy='auto' , n_neighbors=5, random_state=None)

sampling_strategy: float, str, dict or callable, default='auto'

'auto'='not majority': resample all classes but the majority class

n_neighbors: int or obj, default=5 (The nearest neighbors used to define the neighborhood of samples to use to generate the synthetic samples.) [P+N]

int: the # of neighbors to use

obj: an instance of a compatible nearest neighbors algorithm

Adaptive bouderline SMOTE:

1.Consider the nominal features

2.Standardize the continuous features.

3.G = (nnum+pnum)

4.For each $\mathbf{p}_j$(i=1,…,pnum), select ***k*** nearest neighbors (N_j) based on Euclidean distance.

r_j: the # of majority samples∈N_j.

- 0.4*k< r_j<k-1 → Boundary samples → Could develop a adaptive hyperparameters
- Boundary= {$\mathbf{p}'_1$, $\mathbf{p}'_2$,…, $\mathbf{p}'_{bnum}$}, 0≤ bnum≤ pnum

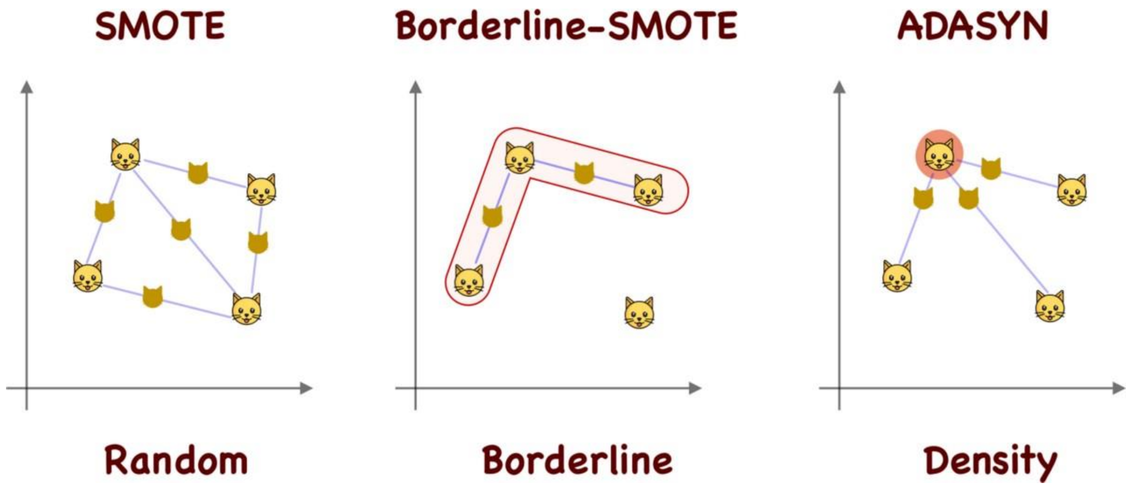
5.Adaptive control the number of synthetics for each $\mathbf{p}'_j$

$\hat{r}_j$ = r_j/ sum of r_j(j∈ Boundary samples) [normalize to sum of $\hat{r}_j$=1].

6.For each $\mathbf{p}'_j$ will generate $\hat{r}_j$*G synthetics. [$\hat{r}_j$*G times]

3.4 Oversampling - Overview

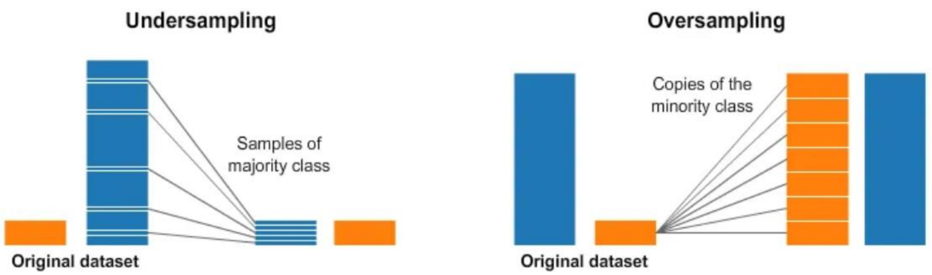
- **SMOTE** and **ADASYN** may not generate the good result. Why?
  →focuses on the samples are the **noise samples**
- Setting the class weight is an another good method.



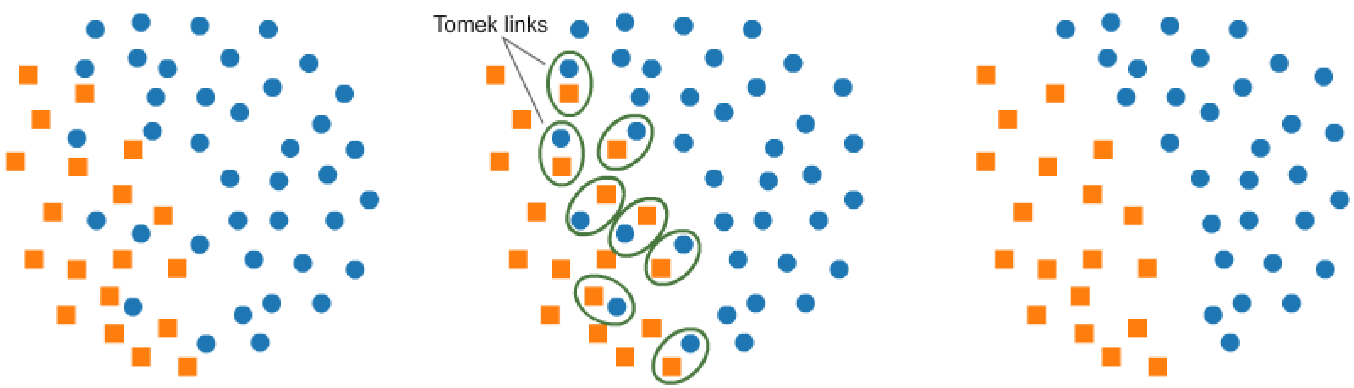
- How to obtain the k (minority class) nearest neighbors
  Base: Calculate the distance of (xi, xj), i=1,...,num; j=i+1,...,num.
  Then, (xj, xi)=(xi, xj) --> Make the matrix of distance for all pairs (xi, xj)
- k nearest neighbors of xi
  - Sort the (xi,xj), j=1,...,num
  - Pick the first K entries from the sorted (xi,xj)
- k minority class nearest neighbors of xi
  - Sort the (xi,xj) where xj are minority examples, j=1,...,pnum-1
  - Pick the first K entries from the sorted (xi,xj)

3.5 Over- & under- sampling – SMOTETomek 1

- Batista, G.E., Bazzan, A.L., Monard, M.C. Balancing Training Data for Automated Annotation of Keywords : a Case Study. WOB 2003, 10-18.
- Over-sampling using SMOTE and cleaning using Tomek links.



- Tomek link: For an Tomek link ($\boldsymbol{p}_i$,$\boldsymbol{n}_j$) where $\boldsymbol{p}_i \in P$ and $\boldsymbol{n}_j \in N$ There is no $\boldsymbol{x}_k \in \{P, N\}$ such that: d($\boldsymbol{p}_i$,$\boldsymbol{x}_k$)<d($\boldsymbol{p}_i$,$\boldsymbol{n}_j$) or d($\boldsymbol{n}_j$,$\boldsymbol{x}_k$)<d($\boldsymbol{p}_i$,$\boldsymbol{n}_j$).
- Concept: 找出邊界鑑別度不高的marjority samples, 認為Tomek are noises應該剔除. **Both majority and minority class examples** that form a Tomek link **are removed** because minority class example is **artificially created** and the data sets are currently **balanced**.



3.5 Over- & under- sampling – SMOTETomek 2

- Tomek link: For an Tomek link ($\boldsymbol{p}_i$,$\boldsymbol{n}_j$) where $\boldsymbol{p}_i \in P$ and $\boldsymbol{n}_j \in N$
  There is no $\boldsymbol{x}_k \in \{P, N\}$ such that: d($\boldsymbol{p}_i$,$\boldsymbol{x}_k$)<d($\boldsymbol{p}_i$,$\boldsymbol{n}_j$) or d($\boldsymbol{n}_j$,$\boldsymbol{x}_k$)<d($\boldsymbol{p}_i$,$\boldsymbol{n}_j$).
  - For each pair ($\boldsymbol{p}_i$,$\boldsymbol{n}_j$) (i=1,…,pnum, j=1,…,nnum):
    For each $\boldsymbol{x}_k$ (k=1,…,num):
    if ($\boldsymbol{p}_i$, $\boldsymbol{x}_k$)<($\boldsymbol{p}_i$,$\boldsymbol{n}_j$) or ($\boldsymbol{x}_k$,$\boldsymbol{n}_j$)<($\boldsymbol{p}_i$,$\boldsymbol{n}_j$),
    then ($\boldsymbol{p}_i$,$\boldsymbol{n}_j$) is not a Tomek link.
- =SMOTETomek(*, sampling_strategy='auto', random_state=None, smote=None, tomek=None)

3.6 Over- & under- sampling – SMOTEENN 1

- G. Batista, R.C. Prati, M.C. Monard. 2004. A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, 6(1), 20-29. → Over-sample using SMOTE followed by under-sampling using Edited Nearest Neighbors.
- 與 Tomek Links的觀念相同, 透過某種方式來剔除鑑別度低的samples. 只是改成了對majority class samples尋找 k-nearest neighbors, 如果有1/2以上(當然, 門檻可以自己設定)都不屬於majority class samples, 就將其剔除. 通常這些樣本也會出現在少數樣本之中. In general, k=3.
- The algorithm of ENN
  - For each $\mathbf{x}_j$ (j=1,…,num), select ***k*** nearest neighbors ($\mathbf{N}_j$).
  - If the class of $\mathbf{x}_j$ and the majority class of $\mathbf{N}_j$ is different, then the $\mathbf{x}_j$ and $\mathbf{N}_j$ are deleted.

  不管$\mathbf{x}_j \in P$ or $\mathbf{x}_j \in N$ 都要做.
- =SMOTEENN(*, sampling_strategy='auto', random_state=None, smote=None, enn=None)

3.7 Python – Oversampling 1

```
# conda install -c conda-forge imbalanced-learn
from imblearn.over_sampling RandomOverSampler, SMOTE, ADASYN, BorderlineSMOTE,
SVMSMOTE
from imblearn.combine import SMOTEENN, SMOTETomek
…
# 0. Set the hyperparameters
random_state_alg  = 42
random_state_over = 3 # 更動 random_state_over=2, 答案完全不一樣
CV                = 5
max_k_num = 10
Show_dstail = 2 # 2 全秀Unbalanced+Balanced, 1:只秀Unbalance, 0:只秀k-num

# 1. Import the data
X   = pd.read_excel('06-Imbalance-BreastCancer.xlsx', sheet_name = 'Breast Cancer-10')
X.drop(['6th Stage','differentiate','Regional Node Examined'], axis=1, inplace=True) #有 T
Stahe & N Stage就不用'6th Stage'
y  = X.pop('Class') # For classification
X  = np.array(X)
```

3.7 Python – Oversampling 2

```
# 2. Create Objects (alggorithm, CV, Over-sampling)
clf1 = DecisionTreeClassifier(random_state = random_state_alg)
clf2 = DecisionTreeClassifier(class_weight='balanced', random_state = random_state_alg)
sskf = StratifiedKFold(n_splits=CV, shuffle=False) # 設定KFold CV
OverStr = ('None    ','ROS     ','SMOTE   ','SMOTENC ', 'BDLSMOTE','SVMSMOTE','ADASYN  ',
           'SMOTEENN','SMOTETomek')
Over = np.zeros((9), dtype=object)
# 3. Main Loop: knum -> cv -> over
for k_num in range(5, max_k_num+1):
  Over[1] = RandomOverSampler(sampling_strategy='not majority', random_state=3)
  Over[2] = SMOTE(sampling_strategy='not majority', k_neighbors=k_num,
          random_state=3)
  Over[3] = SMOTENC(categorical_features=[1], sampling_strategy='not majority',
          k_neighbors=k_num, random_state=3)
  Over[4] = BorderlineSMOTE(sampling_strategy='not majority', k_neighbors=k_num,
          random_state=3)
  Over[5] = SVMSMOTE(sampling_strategy='not majority', k_neighbors=k_num,
          random_state=3, m_neighbors=10, svm_estimator=None, out_step=0.5)
  Over[6] = ADASYN(sampling_strategy='not majority', n_neighbors=k_num,
          random_state=3)
  Over[7] = SMOTEENN(sampling_strategy='not majority', smote=Over[3])
  Over[8] = SMOTETomek(sampling_strategy='not majority', smote=Over[2])
```

3.7 Python – Oversampling 3

```
res = list(np.zeros((2,2,CV))) # 1=(X,class_weight), 2=(AUC ROC, F1), 3=CV
# Main loop 每種oversampling run一次
iBestOver, BestAUC = -1, -1 # Bast over-sampleing for AUC ROC with no class_weight
for oIdx, Over1 in enumerate(Over):
  # 開始CV
  for cvIdx, (train, validation) in enumerate(sskf.split(X, y)):
    # 設定dataset
    if oIdx==0:
      X_imb, y_imb = X[train,:], y[train]
    else: # 將得到的 training fold oversampling
      X_imb, y_imb = Over1.fit_resample( X[train,:], y[train])

    # training & 用原始dataset做validation & 計算 metrics
    clf1.fit(X_imb, y_imb)
    y_pred1 = clf1.predict(X[validation,:])
    # 傳入 y lable與預測結果, 計算 accuracy,f1(在此可以使用各種 metric)
    res[0][0][cvIdx]=roc_auc_score(y[validation], y_pred1)
    res[0][1][cvIdx]=     f1_score(y[validation], y_pred1)
```

3.7 Python – Oversampling 4

```
if Show_dstail>=2:
        clf2.fit(X_imb, y_imb)
        y_pred2 = clf2.predict(X[validation,:])
        res[1][0][cvIdx]=roc_auc_score(y[validation], y_pred2)
        res[1][1][cvIdx]=    f1_score(y[validation], y_pred2)

  if np.mean(res[0][0])>BestAUC:
    BestAUC = np.mean(res[0][0])
    iBestOver = oIdx
  if Show_dstail>=1:
    print('[%s]  Unbalanced ROC AUC:%f f1:%f' % (OverStr[oIdx], np.mean(res[0][0]),
np.mean(res[0][1])))
    if Show_dstail>=2:
      print('[Size=%3d] Balanced ROC AUC:%f f1:%f\n' % (len(X_imb), np.mean(res[1][0]),
np.mean(res[1][1])))

 print('[k-num=%d] The best ROC AUC:%f [%s]\n' % (k_num, BestAUC,
         OverStr[iBestOver]))
```

3.8 Practice [06-3-Over.py]

- 06-3-Over.py [30 min]
- 1. Load breast_cancer, using DecisionTree as classifier
- 2. Create the object – classifier, oversampling
- 3. Main-loop: k_num -> cv -> over
  - k_num
    - Cross-validation (CV)
      - Oversampling the training fold
        - Training the model by the oversampling set
        - 用原始dataset做 validation
        - 計算metrics & save
- 2 # 更動 random_state_over=2, 答案完全不一樣
- 討論結果
  - 1.有 over-sampling後, class_weight有沒有效果?
  - 2.SMOTENC有考慮那些是nomial features, 哪些nomial features效果是最好的?
  - 3.Over-sampling hyperparameter: k_num (the # of neighbors for target sample that generate the synthetic example) 是否為hyperparameter?

Model Evaluation

1 Metrics for Evaluation

2 Model Evaluation

3 Oversampling

4 Learning Curve of Model

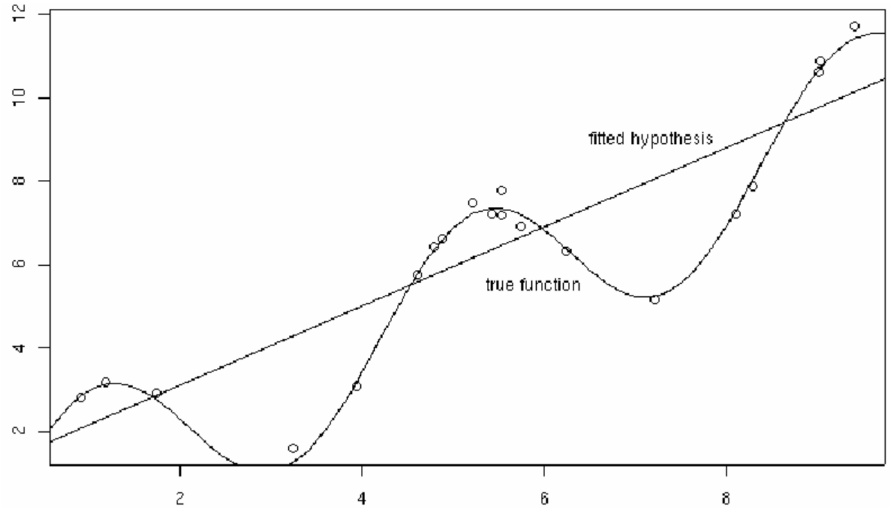
5 Hyperparameter Optimization

4.1 Variance & Bias 1

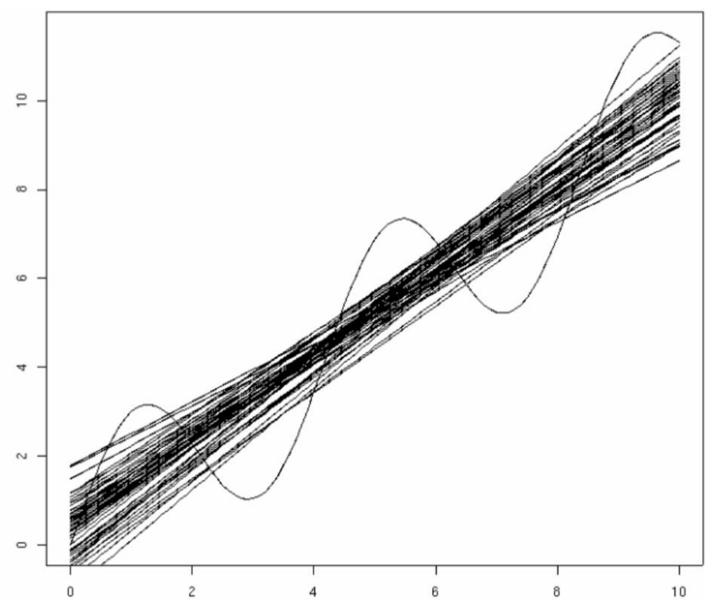
- $Y_i = f(x_i) + \varepsilon_i = f + \varepsilon_i$
  - f: true function that always complete unknown
  - $\varepsilon_i \sim N(0, \sigma)$: ε_i is a random variable (noise)
- $\hat{f}_D(x_i) = \hat{f}_D$: estimated model from certain training set *D*
  - Different training dataset *D* → different estimated model $\hat{f}_D$
  - $\hat{f}_D$ has some error when tested on some test data
  - $\hat{f}_D$ → 有很多版本
- How to measure the error of $\hat{f}_D$: the mean squared error (MSE)
  - $E\left[\left(Y_i - \hat{f}_D\right)^2\right] = \sigma^2 + \mathrm{Var}\left[\hat{f}_D\right] + (\mathrm{Bias}[\hat{f}_D])^2$
  - σ^2: irreducible error, the error from the ε_i
  - Var[$\hat{f}_D$]=$E[(\hat{f}_D - E\left[\hat{f}_D\right])^2]$: the amount by which $\hat{f}_D$ varies as we change training sets (不同$\hat{f}_D$間的variance)
    - $E[\hat{f}_D]$: the expect value of the different models $\hat{f}_D$
  - $(\mathrm{Bias}[\hat{f}_D])^2 = (f - E[\hat{f}_D])^2$: bias, the squared error between of f and $E[\hat{f}_D]$→ true function與E[estimated model] 的差平方

4.1 Variance & Bias 2

- One of $\hat{f}_D(x_i)$: from certain training dataset *D*



- 20 個 $\hat{f}_D(x_i)$: from 20 training datasets *D*



4.1 Decomposition of Variance & Bias 1

- For a independent variable x (feature)
  - Many observation x_i → many Y_i, ε_i, and $\hat{f}_D$ where Y_i and ε_i are random variables
  - Different observation x_i from training dataset *D*→different estimated model $\hat{f}_D$
- $Y_i = f + \varepsilon_i$
  - f: true function that always complete unknown
  - $\varepsilon_i \sim N(0, \sigma)$: ε_i is a random variable (noise)
- $E[c] = c$ (常數mean還是本身)
- $\mathrm{V}ar[c] = 0$ (常數variance為0)
- $E[Y_i] = E[f + \varepsilon_i] = E[f] + E[\varepsilon_i] = f + 0 = f$
- $Var[Y_i] = Var[f + \varepsilon_i] = Var[f] + Var[\varepsilon_i] = 0 + \sigma^2 = \sigma^2$

4.1 Decomposition of Variance & Bias 2

- $E\left[(Y_i - \hat{f}_D)^2\right] = \sigma^2 + (Bias[\hat{f}_D])^2 + Var\left[\hat{f}_D\right]$
- $Var[x] = E[(x - E[x])^2] = E[x^2 - 2xE[x] + E[x]^2]$

$$= E[x^2] - 2E[x]E[E[x]] + E[E[x]^2]$$

$$= E[x^2] - 2E[x]E[x] + E[x]^2$$

$$= E[x^2] - E[x]^2$$

- $E[\left(Y_i - \hat{f}_D\right)^2] = Var[Y_i - \hat{f}_D] + \left(E[Y_i - \hat{f}_D]\right)^2$

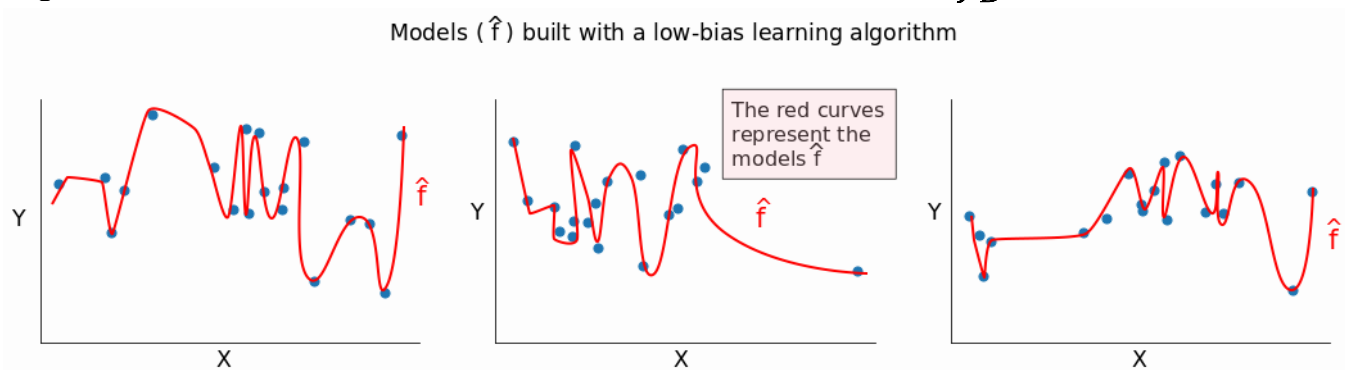
$$= Var[Y_i] + Var[-\hat{f}_D] + \left(E[Y_i] - E[\hat{f}_D]\right)^2$$

$$= \sigma^2 + Var(\hat{f}_D)] + \left(f - E[\hat{f}_D]\right)^2$$

$$= \sigma^2 + E\left[(\hat{f}_D - E[\hat{f}_D])^2\right] + (Bias[\hat{f}_D])^2$$

4.1 Trade-off of Variance and Bias

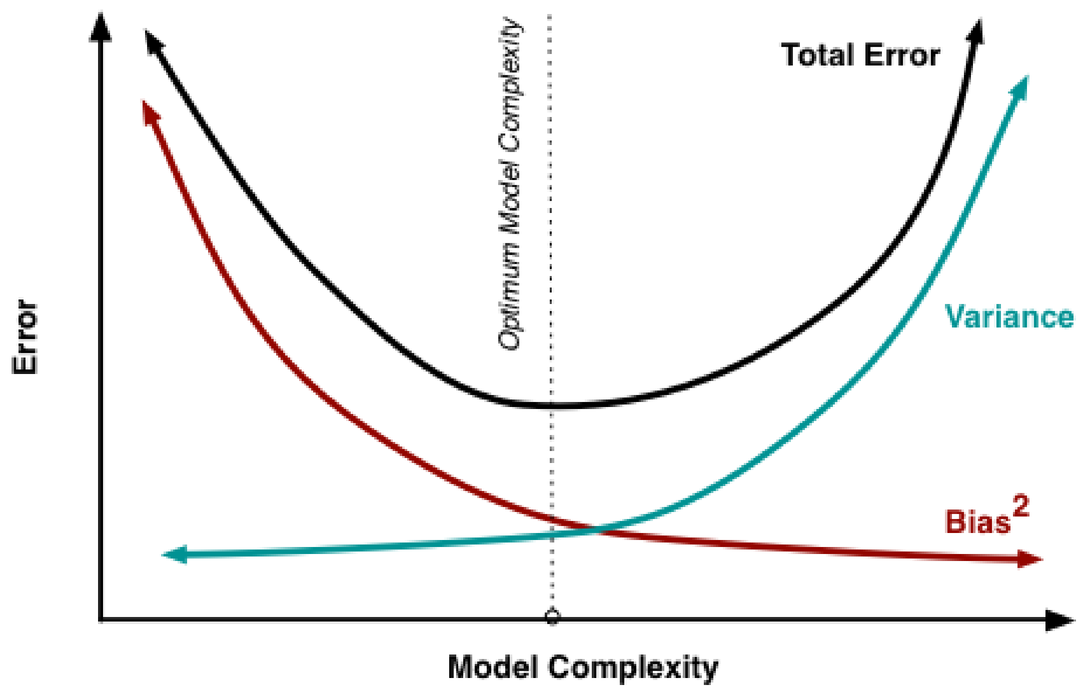
- Impossible to keep both bias and variance at their minimum for all model.
- Low-biased method:
  - Different training sets → different models
  - High variance between the different models $\hat{f}_D$



- Low-variance method
  - High bias between the true function (f) and the estimated models $E(\hat{f}_D)$

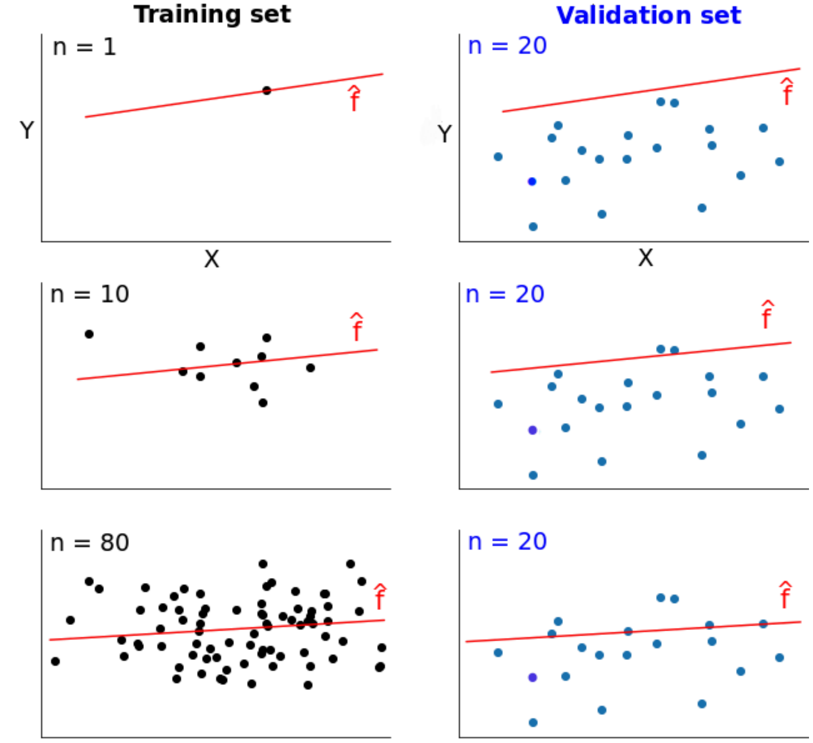
4.1 Trade-off of Variance and Bias

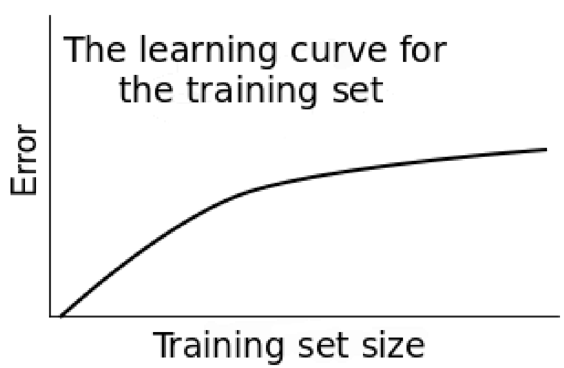
- Low bias=準, Low variance=穩
- Low bias: avoid building a model that's too simple.
- Low variance: avoid building an overly complex model.
- It is impossible to keep both bias and variance at their minimum.
- In practice, however, we need to accept a trade-off.

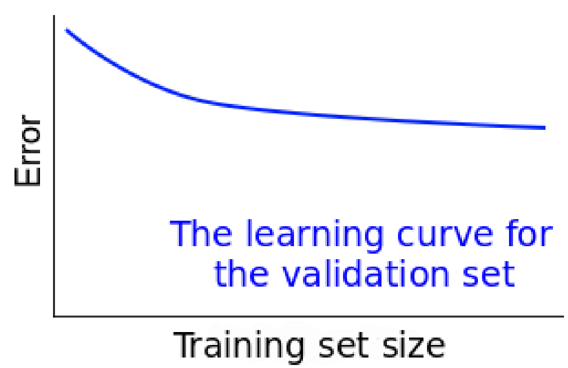


4.2 Learning Curve - Training size

- 2 errors for different sets
  - Error of validation set:
    $$E\left[\left(Y_i - \hat{f}_D(x_{valid})\right)^2\right]$$
  - Error of training set:
    $$E\left[\left(Y_i - \hat{f}_D(x_{train})\right)^2\right]$$
- Increasing the size of training set
  - Error of training set: ↑
  - Error of validation set: ↓
- Learning curves: plot the 2 error scores with different training sets

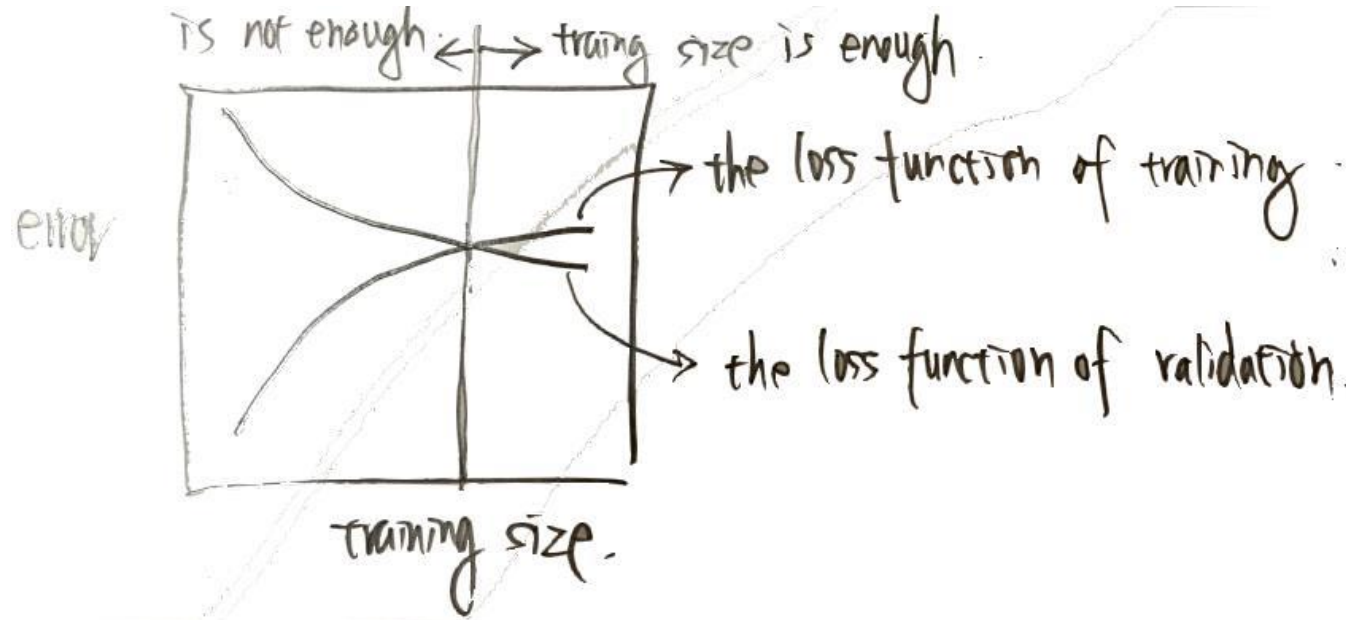






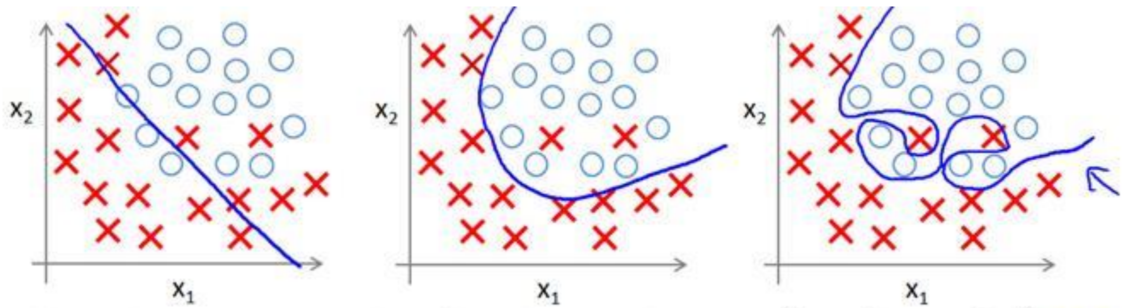
4.2 Learning Curve 2 - Training size

- Learning curve: in this figure, the Y-axis can be **accuracy rate or error rate!**
- If the training set is not enough:
  the loss function of training set < the loss function of validation set
- If the training set is enough:
  the loss function of training set >= the loss function of validation set



4.3 Underfitting or overfitting

- Underfitting　　Fitting　　Overfitting



Underfitting: $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ (g = sigmoid function)

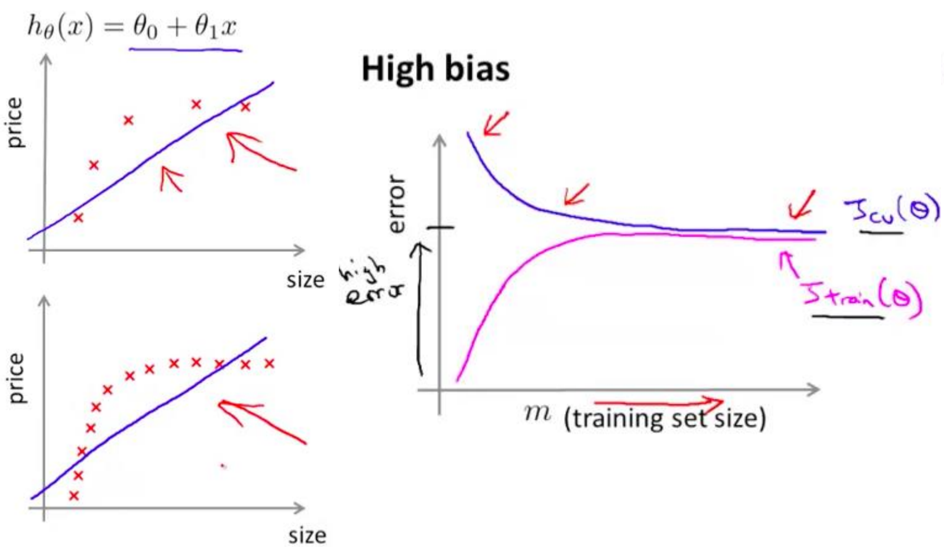
Fitting: $g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

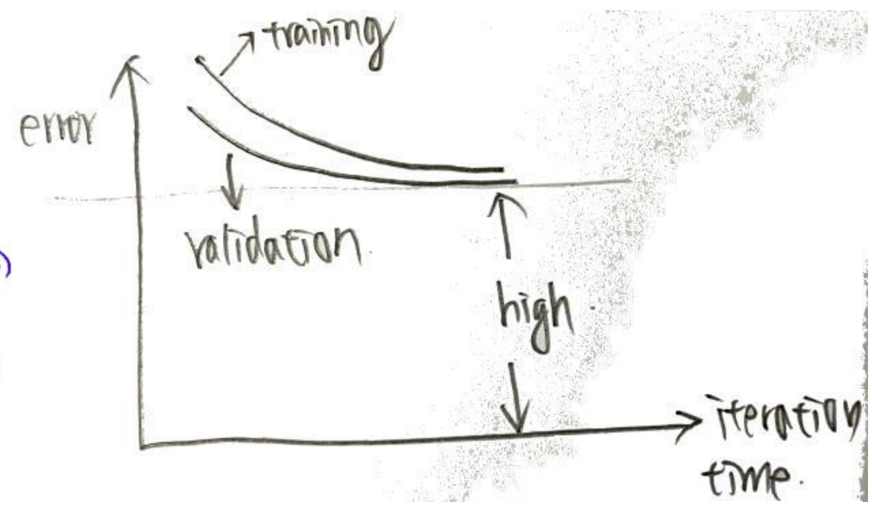
Overfitting: $g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

- Underfitting: training set, validation set
  → 精度都很低 for training & validation sets
  → model is too simple
- Overfitting: training set
  → 精度很高 for training set(連錯誤都被訓練成model)
  →實際上誤差很大 for validation set
  - model is too complexity! 複雜到可以fitting到all training data, 但只針對training data有用, 對其他data無用

4.3 Underfitting – viewpoint of training size

- Error from **high bias**
- The errors of training & validation sets are **closer**
- High **errors** for both training & validation sets
- Even getting more training set, the error rate **can not be reduced**
- More training set is not help.
  - The model is **too simple**!
- Solution
  - Change the algorithm
  - Increase the # of features

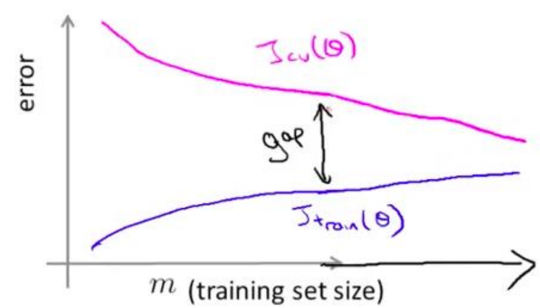




4.3 Overfitting - viewpoint of training size

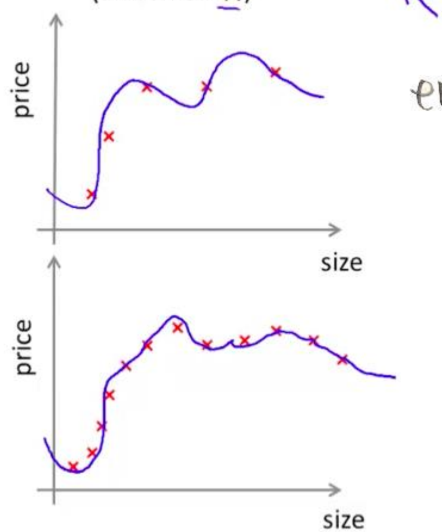
- Error from **high variance**
- The gap of the 2 errors is large
- When the loss function of training set are lower than the loss function of validation set, the process can be terminated!
- More training set is help.
  - The data is **too small**!
- Solution
  - Increase the training set
  - Reduce the # of features
  - L1 & L2 regularization

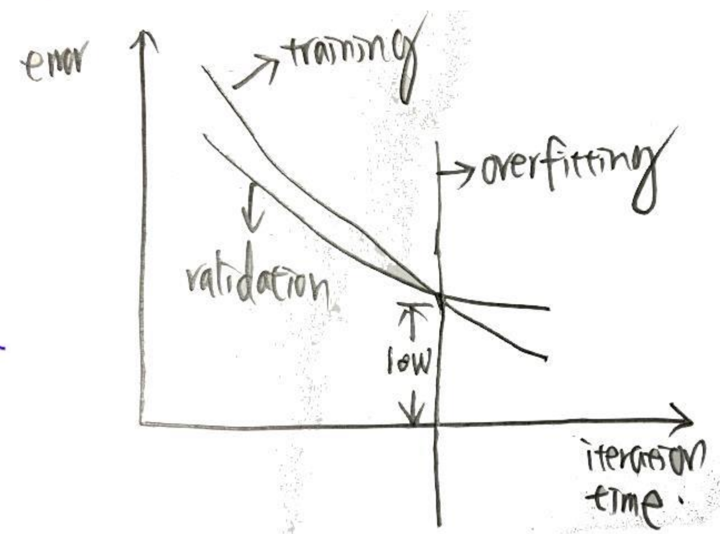
High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help.

$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$
(and small λ)





4.4 Python – learning_Curve 1

from sklearn.model_selection import learning_curve
train_sizes, train_scores, validation_scores = learning_curve(parameters…)

- Parameters for learning_curve
  - estimator: object type
    - training model that implements the “fit” and “predict” methods
  - X:array, shape(n_samples, n_features) (X, features array)
  - Y:array, shape(n_samples) (y, target label)
  - Train_sizes:array, shape (n_ticks,)
    - 可以給trainingset裡筆數(最多只能X set的8成)或比例, 如[0.1, 0.3, 0.6, 1.]
  - Cv:int, (the # of cv, regression類使用**KFold**, classification類使用**StratifiedKFold**)
  - scoring=None,
    - 不同estimator有不同的計算分事的方法, 請參考
      # https://scikit-learn.org/stable/modules/model_evaluation.html
      All scorer objects follow the convention that higher return values are better than lower return values
  - shuffle=False,
    - 在切分cv前是否先打亂dataset裡面tuple的順序
  - verbose=0
    - 數字越高, 越可以看裡面計算的細節

4.4 Python – learning_Curve 2

from sklearn.model_selection import learning_curve
train_sizes, train_scores, validation_scores = learning_curve(parameters…)

- Returen
  - train_sizess:array,shape(n_ticks,)
    - 傳回每次learn training set的tuple #
  - train_scores:array,shape(n_ticks, n_cv_folds)
    - 傳回歷次learn的各個CV之train分數
  - test_scores :array,shape(n_ticks, n_cv_folds)
    - 傳回歷次learn的各個CV之validation分數

```
# 1 Run learning_curve
#   Check the train_sizes, train_scores, validation_scores
train_sz = [1, 100, 2000, 5000, 11200]
train_sizes, train_scores, validation_scores = learning_curve( \
    LinearRegression(), # Underfitting
    X[features], X[target], train_sizes = train_sz, \
    cv = 5, scoring = 'neg_mean_squared_error', verbose=0)
print('Training size=%s' %(train_sz))
print('Training:%s' % (-train_scores.mean(axis=1)))
print('Validation:%s' % (-validation_scores.mean(axis=1)))
```

4.4 Practice [06-4-LearnCurve.py]

- Electrical energy output of a power plant from Turkish
  - x: AT, Ambiental Temperature 環境溫度 / V, Exhaust Vacuum 排氣真空
  - AP, Ambiental Pressure 環境壓力/ RH, Relative Humidity 相對濕度
  - Y: PE, Electrical Energy Output
- 06-4-LearnCurve.py [30 min]
  - Simple example for LearningCurve
  - 1. Run learning_curve
    - Change the parameters: train_sizes, cv, scoring
      - scoring請參考metric
    - Verify the return
      - train_sizes, train_scores, validation_scores, train_scores_mean, validation_scores_mean
  - 2. Draw the learning curve
    - Which is underfitting? [RandomForestRegressor() or LinearRegression()]
    - Which is overfitting? [RandomForestRegressor() or LinearRegression()]
    - Chanage the parameter: max_leaf_nodes = 1000
      RandomForestRegressor(..., max_leaf_nodes = 100)

Model Evaluation

1 Metrics for Evaluation

2 Model Evaluation

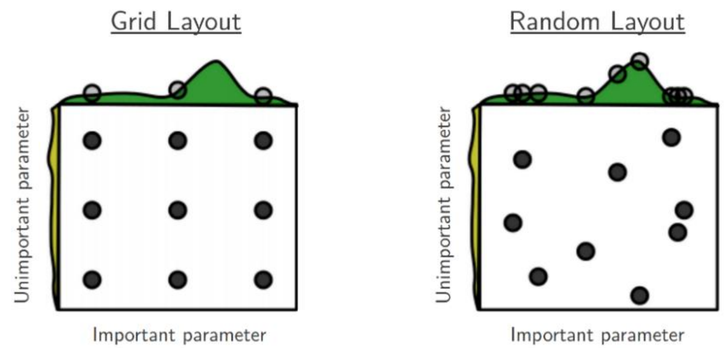
3 Oversampling

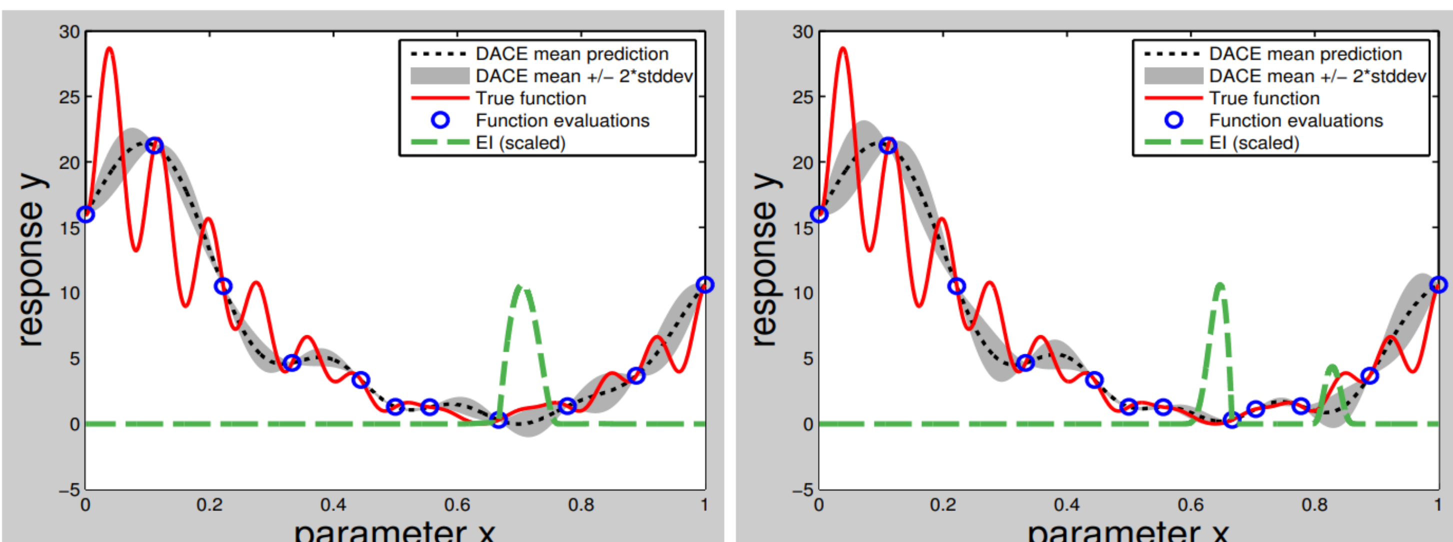
4 Learning Curve of Model

5 Hyperparameter Optimization

5 Hyperparameter Optimization

- https://www.youtube.com/watch?v=bcy6A57jAwI (13:17, 有英文字幕)
- Grid search : Fixed point
- Random search : Random point
- Auto search: hyperopt / optuna
  - o evaluation samples
  - — Red line: true loss function
  - --- DACE mean prediction of approximate model of ture loss fun.(surrogate)
  - --- Use acquisition functions (取得函數, 比如EI) to obtain the max probability of next sampling point





5.1 Grid Search 1

```
from sklearn.model_selection import GridSearchCV
GridSearchCV(estimator, param_grid, *, scoring=None, refit=True,
                 cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan,
                 return_train_score=False)
```

- Parameters
  - Estimator: object
    Either estimator needs to provide a **score function**, or **scoring** must be passed.
  - Param_grid: **dict** or list
    Suggest use **dict**. For example:
    Hparams = {'criterion'   :['gini', 'entropy'],
                 'min_samples_leaf' :[1,3]}
    key := 原本algorithm hyperparameter的name
    value := a list for search this hyperparameter.
  - n_jobs: **int**, default=None
    -1 means using all processors, 可以設為-1加快速度.

5.1 Grid Search 2

from sklearn.model_selection import GridSearchCV

- Parameters
  - Scoring: **str, callable, list**, tuple or **dict**, default=None,
    - 不同estimator有不同的metric, 請參考
      # https://scikit-learn.org/stable/modules/model_evaluation.html
    - However, scikit-learn has 2 problems:
      (1) binary classification時pos_label內定皆為1(有時候你需要pos_label=0)
      (2) No **NVP** in scikit-learn.
      Therefore, we can use the following example:

```
scoring={'accuracy': 'accuracy', # 簡單使用
         'recall': make_scorer(recall_score, pos_label=0, greater_is_better=True,
                   needs_proba=False, needs_threshold=False),
                   #特別指明pos_label=0
         'NPV': make_scorer(NPV, pos_label=1, average='binary',
                greater_is_better=True, needs_proba=False,
                needs_threshold=False)
                # 可以自己寫一個 NPV的 metric function
        }
```

5.1 Grid Search 3

from sklearn.model_selection import GridSearchCV

- Parameters
  - Cv: int, cross-validation generator or an iterable, default=None
  - Refit: bool, **str**, or callable, default=True
    - Refit an estimator using the best found parameters on the whole dataset. (最後再用 best 對whole dataset跑一次)
    - For multiple metric evaluation, this needs to be a str denoting the scorer that would be used to find the best parameters for refitting the estimator at the end. (當有使用multiple metrics時, 比需指名用哪個metric最為rank以找到best hyperparameter)
  - verbose: int

    Controls the verbosity: the higher, the more messages.

5.1 Grid Search 4

from sklearn.model_selection import GridSearchCV

- Attributes : For multi-metric evaluation, this is present only if refit is specified.
  - best_index_: int
    The best index of refit metric → np.argmax(CV.cv_results_['mean_test_refit'])
  - best_score_:float
    The best score for refit metric → np.max(CV.cv_results_['mean_test_refit'])
  - best_params_:dict
    The best hyperparameter values → .cv_results_['params'][CV. best_index_]
- Return attritube
  - cv_results_: dict of numpy (storing all experiment results, 最重要attribute)
    - .cv_results_['params'] : list of dict for hyperparameter
      list for all search hyperparameters
    - .cv_results_['mean_test_xxx'] : list of mean value of XXX metric
      xxx is the string of metric,
      if single metric → .cv_results_['mean_test_score']
      [mean] → the mean of cv results
    - .cv_results_['std_test_xxx'] : list of std value of XXX metric
    - 要看有幾個hyperparameters可以用
      len(CV.cv_results_['params'])

5.2 Random Search 1

from sklearn.model_selection import RandomizedSearchCV

class sklearn.model_selection.RandomizedSearchCV(estimator, param_distributions, *, n_iter=10, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None, error_score=nan, return_train_score=False)

- Parameters
  - Estimator: object → is the same as GridSearchCV
  - param_distributions: **dict** or list of dicts (Suggest use **dict**.)
    key := 原本algorithm hyperparameter的name
    value := a **list** or **distribution** for search this hyperparameter.
    - 列舉式Enumerate→ List
      Ex. 'criterion': [‘gini’, ‘entropy’]
    - integer → List or range(start=1, stop=10, step=2)
      Ex. 'max_depth': range(2, 11, 2) or 'max_depth': [2,4,6,8,10]
    - float → distribution
      uniform(loc=, scale=), uniform distribution on [loc, loc + scale]
      norm[loc=, scale=], normal distribution with mean(loc) and s.d.(scale)

5.2 Random Search 2

from sklearn.model_selection import RandomizedSearchCV

- Parameters
  - n_iter: **int**, default=10
    要random search幾次
  - n_jobs, scoring, cv, refin, verbose: are the same as GridSearchCV
- Attributes (are the same as GridSearchCV)
  - best_score_:float
    For multi-metric evaluation, this is present only if refit is specified.
  - best_params_:dict
    For multi-metric evaluation, this is present only if refit is specified.
  - best_index_: int
    The best index of ".cv_results_['params']"
  - cv_results_: dict of numpy (storing all experiment results, 最重要attribute)
    - 同GridSearchCV

5.3 Practice [06-5-1 Hyperparameter.py]

- 06-5-1 Hyperparameter.py [20 min]
- (a) 3 classes
  - print(CV.cv_results_) & check the detail information
  - print(CV2.cv_results_) & check the detail information
  - What's .cv_results_['params']?
  - What's .cv_results_['mean_test_xxx']?
  - Using the confuse matrix to calculate (a)accuracy (b)recall (c)PPV (d)NPV (e) f1

- (b) binary classification
  - Using "dbset = datasets.load_breast_cancer()" for binary classification class_n = 2
  - What's pos_label?
  - Try it again!!!

5.4 optuna 1

- Install optuna firstly!
  pip install optuna plotly # plotly是畫圖用的
- The terminology in Optuna
  - **optuna.trial** object:
    - A object **to track the input variables (hyperparameters) and its results of that evaluation**.
    - Each time the objective function is called to perform an evaluation
      → A new **trial** object will be internally created
      → Being a parameter (**optuna.trial** object) for the objective function
      → Let you perform the evaluation & record the result in the objective function
    - This allows for **easy tracking and management of the results of each trial**, which is important for guiding the optimization algorithm towards the optimal solution.
  - **optuna.study** object: An optimization session, which is a set of trials

5.4 optuna 2

1. Define a objective function
   - Define the dict of hyperparameters所有可能的solution space
   - Pick a set of hyperparameters by trial.suggest_xxx
   - Perform an evaluation & compute the metrics
   - Return the metrics

   The hypermeters & its results(metrics) will be stored in the passing trial object
2. Create sampler & pruner object → optuna.create_study
   study = optuna.create_study(sampler=x, pruner=x, direction='maximize')
3. Use optimize() to perform the auto search (傳入定義的 objective function)
   study.optimize(objective, n_trials =30)
4. Optuna 視覺化分析

5.4.1 Define objective 1

1. Create a objective function, uses **trial** object to set each Hyperparameter
   - Using **trial.suggest_XXX** to set up each Hyperparameter
     - = trial.suggest_categorical('criterion',['gini', 'entropy'])
       → for categorical parameters
     - = trial.suggest_int('max_depth', 2, 21)
       → for integer parameters
     - = trial.suggest_float('C', 0.1, 100)
       → for floating point parameters

     How to pick the hyperparameters will be influenced by trial.suggest_xxx
   - Evaluation the performance of the set of Hyperparameter (ex.corss_validate)
   - Calculate the spectfic metrics and return it.
     fitness value 可以越小越好 or 越大越好, 主要是在 optuna.create_study(direction='maximize') 設定 direction='maximize' or 'minimize'

5.4.1 Define objective 2

```
#先create 好 object, 不要在 objective裡面create, 浪費記憶體又慢
estimator = DecisionTreeClassifier()
def objective(trial):
  # 1.1 先定義space: hyperparameters所有可能的solution space, 可以視為domain
  DTreeParamA = {# 類別型
            'criterion' :trial.suggest_categorical('criterion',['gini', 'entropy']),
            # 整數型
            'max_depth'      :trial.suggest_int('max_depth', 2, 21)}

  # 1.2 將傳進來的params, 用.set_params() 來重設estimator
  estimator.set_params(**DTreeParamA)

  # 1.3 使用 cross_validate 來做cv=5 的cross-validation
  cvfit = cross_validate(estimator, X_train, y_train, scoring='accuracy', cv=5)
  metric = np.mean(cvfit['test_score'])

  # 1.4 是否要啟動prune (剪枝)
  trial.report(metric, step=0)
  if trial.should_prune():
     raise optuna.TrialPruned()
  return metric
```

5.4.2 Sampling Algorithm 1

2.1 設定sampling algorithm

- Samplers basically continually **narrow down the search space** using **the records of suggested parameter values** and **evaluated objective values**

```
#alg=optuna.samplers.GridSampler(seed=42)   # Grid search
#alg=optuna.samplers.RandomSampler(seed=42) # Random search
alg=optuna.samplers.TPESampler(seed=42) # Tree-structured Parzen Estimator algorithm.
#alg=optuna.samplers.CmaEsSampler(seed=42) # CMA-ES based algorithm
#alg=optuna.samplers.NSGAIISampler(seed=42)# Nondominated Sorting Genetic
                                             Algorithm II
#alg=optuna.samplers.MOTPESampler(seed=42) # Multiobjective Tree-structured Parzen
Estimator
#alg=optuna.samplers.QMCSampler(seed=42) # A Quasi Monte Carlo sampling algorithm
#alg=optuna.samplers.BruteForceSampler(seed=42) # Brute force algorithm.
#alg=optuna.samplers.intersection_search_space()# the intersection of parameter
distributions that have been suggested in the completed trials of the study so far.
#alg=optuna.samplers.IntersectionSearchSpace()  # provides the same functionality of
                                 intersection_search_space with a much faster way
```

- The default sampler is TPESampler.

5.4.2 Sampling Algorithm 2

- ✅ :Supports. ▲:Works, but inefficiently. ✕ :Causes an error, or has no interface.
- Time complexity O(x)
  d is the dimension of the search space.
  n is the number of finished trials.
  m is the number of objectives.
  p is the population size.

	TPESampler	CmaEsSampler	NSGAIISampler	QMCSampler	BoTorchSampler	BruteForceSampler
Float parameters	✅	✅	▲	✅	✅	✅ (❌ for infinite domain)
Integer parameters	✅	✅	▲	✅	✅	✅
Categorical parameters	✅	▲	✅	▲	✅	✅
Pruning	✅	▲	❌	✅	▲	✅
Multivariate optimization	✅	✅	▲	▲	✅	▲
Conditional search space	✅	▲	▲	▲	▲	✅
Multi-objective optimization	✅	❌	✅ (▲ for single-objective)	✅	✅	✅
Batch optimization	✅	✅	✅	✅	▲	✅
Distributed optimization	✅	✅	✅	✅	▲	✅
Constrained optimization	✅	❌	✅	❌	✅	❌
Time complexity (per trial) (*)	$O(dn \log n)$	$O(d^3)$	$O(mp^2)$ (***)	$O(dn)$	$O(n^3)$	$O(d)$
Recommended budgets (#trials) (**)	100 – 1000	1000 – 10000	100 – 10000	as many as one likes	10 – 100	number of combinations

5.4.2 Pruning Algorithm 1

2.2 設定pruning algorithm

- Automatically stop unpromising trials at the early stages of the training (據當前結果的XX數停止不良試驗)

pruner=optuna.pruners.MedianPruner() # Prune with median. For RandomSampler, MedianPruner

pruner=optuna.pruners.NopPruner() # No pruning

pruner=optuna.pruners.PatientPruner() # Prune with tolerance

pruner=optuna.pruners.PercentilePruner() # Prune with specified percentile

pruner=optuna.pruners.SuccessiveHalvingPruner() # Prune with Asynchronous Successive Halving algorithm

pruner=optuna.pruners.HyperbandPruner() #SuccessiveHalvingPruner的更激進版本, 並根據中間結果修剪試驗. For TPESampler, HyperbandPruner is the best.

pruner=optuna.pruners.ThresholdPruner() #當目標函數的值超過或是低於給定閾值時，該剪枝器停止試驗

5.4.2 Pruning Algorithm 2

- To turn on the pruning feature, you need to call report() and should_prune() **after each step of the iterative training**.
  - **.report()** : to report the intermediate objective values to the pruning.
  - **.should_prune()**: need to prune or not?
    (系統只會跟你說該prune了, 但這動作必須自己發動, 歡喜做甘願受)

```
def objective(trial):
  # 1.1 先定義space: hyperparameters所有可能的solution space, 可以視為domain
  # 1.2 將傳進來的params, 用.set_params() 來重設estimator
  # 1.3 使用 cross_validate 來做cv=5 的cross-validation
  cvfit = cross_validate(estimator, X_train, y_train, scoring='accuracy', cv=5)
  metric = np.mean(cvfit['test_score'])
  # 1.4 是否要啟動prune
  #      須要用 trial.report(metric, step=)來跟prune報告此 trial的metric result
  #      其中step表示此 trial 的第幾個step (from 0), 在ML中step幾乎為0,
  #      在Neural Network, step即為iteration(表示做到第幾個step in this epoch)
  trial.report(metric, step=0)
  if trial.should_prune():          # check是否需要prune
     raise optuna.TrialPruned() # 必須自發性啟動prune
  return metric
```

5.4.2 Setup Sampler & Pruner

- For RandomSampler, MedianPruner is the best.
- For TPESampler, HyperbandPruner is the best.

2. 設定sampling algorithm

```
# 2.1 設定 sampler (搜尋方法)
alg=optuna.samplers.TPESampler(seed=42) # Tree-structured Parzen
#   2.2 設定 pruners (修剪方法)
pruner=optuna.pruners.HyperbandPruner()

study = optuna.create_study(sampler=alg, pruner = pruner,
direction='maximize')
```

3. 使用 optimize() 搜尋, 要傳入定義的 fitness function

```
study.optimize(objective, n_trials =30)
print('\n\nSampler is {}'.format(study.sampler.__class__.__name__))
print('Pruner is {}'.format(study.pruner.__class__.__name__))
print('Best Accuracy={}\nHyperparameters = {}\n'.format(study.best_value,
study.best_params))
```

5.4.3 Visualization 1

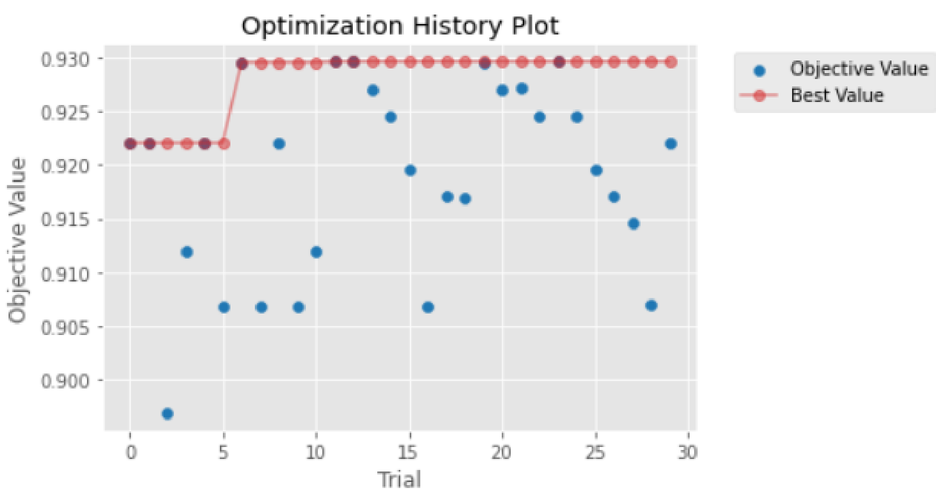
4. Optuna 視覺化分析

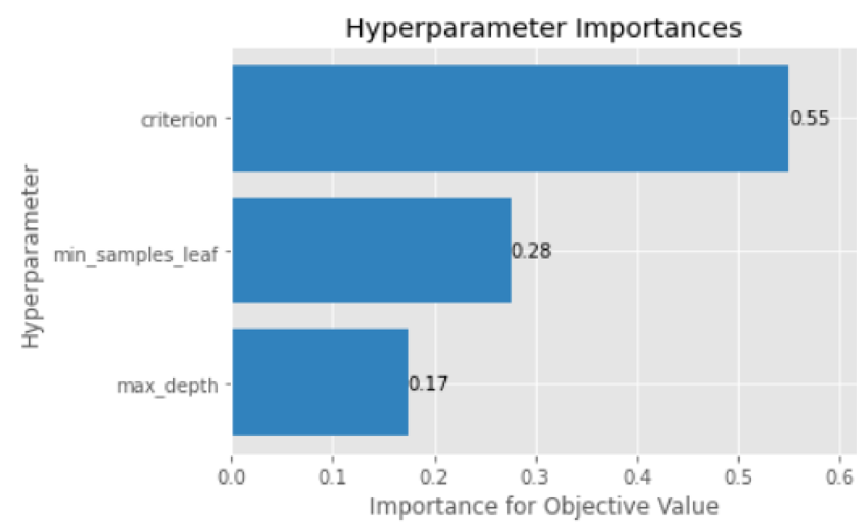
   # The trend-chart of evaluating process (objective values)

   fig = optuna.visualization.matplotlib.plot_optimization_history(study)

   # The importance of each hyperparameter

   fig = optuna.visualization.matplotlib.plot_param_importances(study)



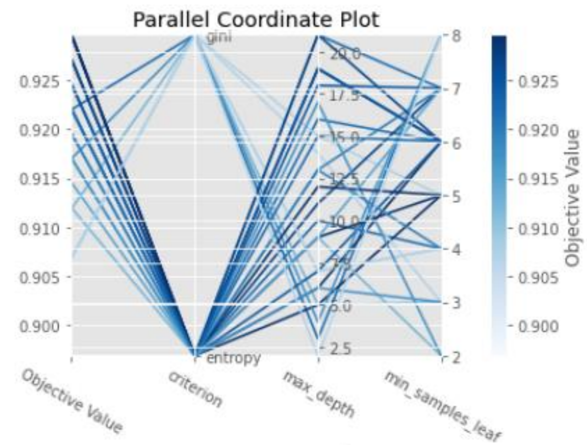


5.4.3 Visualization 2

3. Optuna 視覺化分析

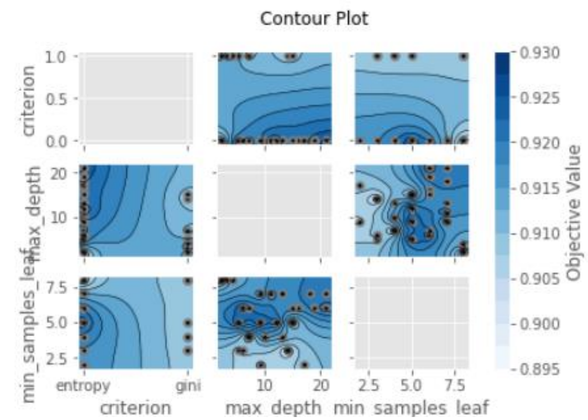
 # the relationship between objective and each hyperparameter

fig = optuna.visualization.matplotlib.plot_parallel_coordinate(study)



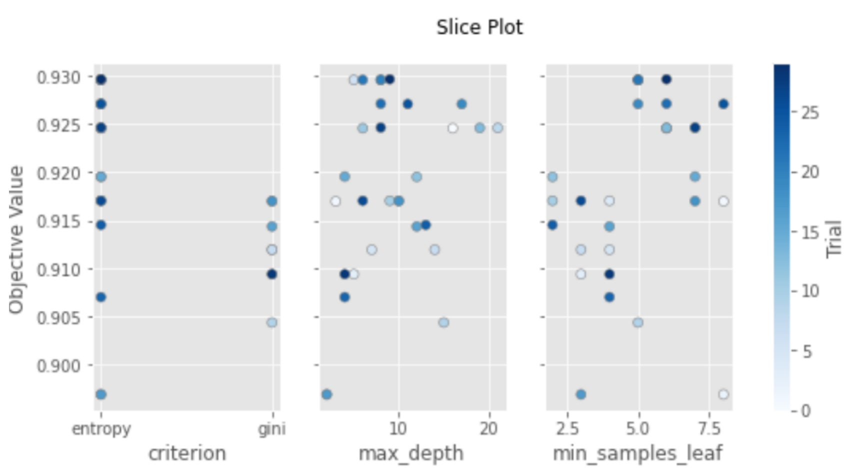
hyperparameter中, 兩兩間的關係

fig = optuna.visualization.matplotlib.plot_contour(study)



視覺化個別參數→隨著trial的進行,看超參的變化

fig = optuna.visualization.matplotlib.plot_slice(study)



5.5 Practice [06-5-3 optunaSearch.py]

- 06-5-3 optunaSearch.py [20 min]
- (a) binary classification

  dataset = datasets.load_breast_cancer()

  class_n = 2
- (b) 3 classification

  dataset = datasets.load_iris() #鳶尾花數據集：150筆花朵樣本

  class_n = 3 # how many classes
- Try different sampling algorithms
- Try using prune or not & different purners