딥러닝/클라우드

Chapter 7

Feature selection
Model stacking

Sejong Oh
Bio Information Technology Lab.

Contents

- Feature selection
- Voting Classifier
- Bagging meta-estimator
- Model stacking

1. Feature selection

- Feature = variable = column in datasets
- More information leads better classification performance ?

당뇨병진단

gender	age	height	weight	f_color	Label
					Pos
					Neg

- 1000 features ?
  - Requires selection of good features

1. Feature selection

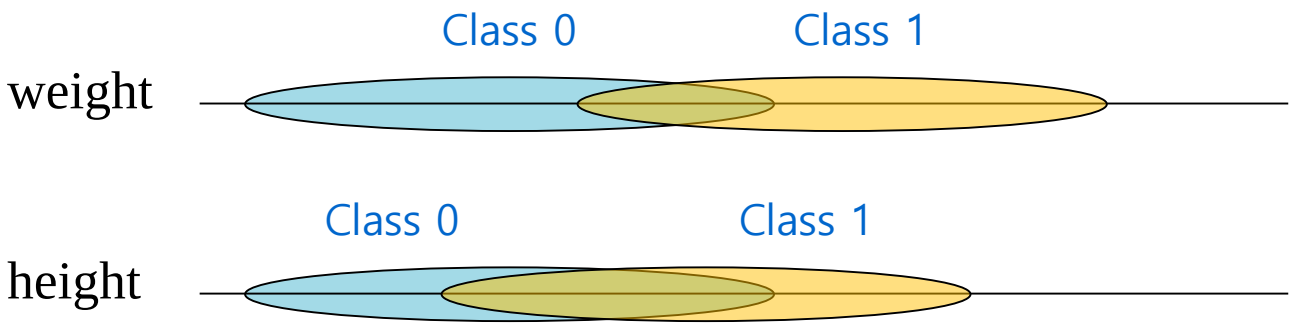
- Feature selection
  - the process where you automatically or manually select
  - those features which contribute most
  - to your prediction variable or output in which you are interested in.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

  - Evaluate features and choose good feature subset

1. Feature selection

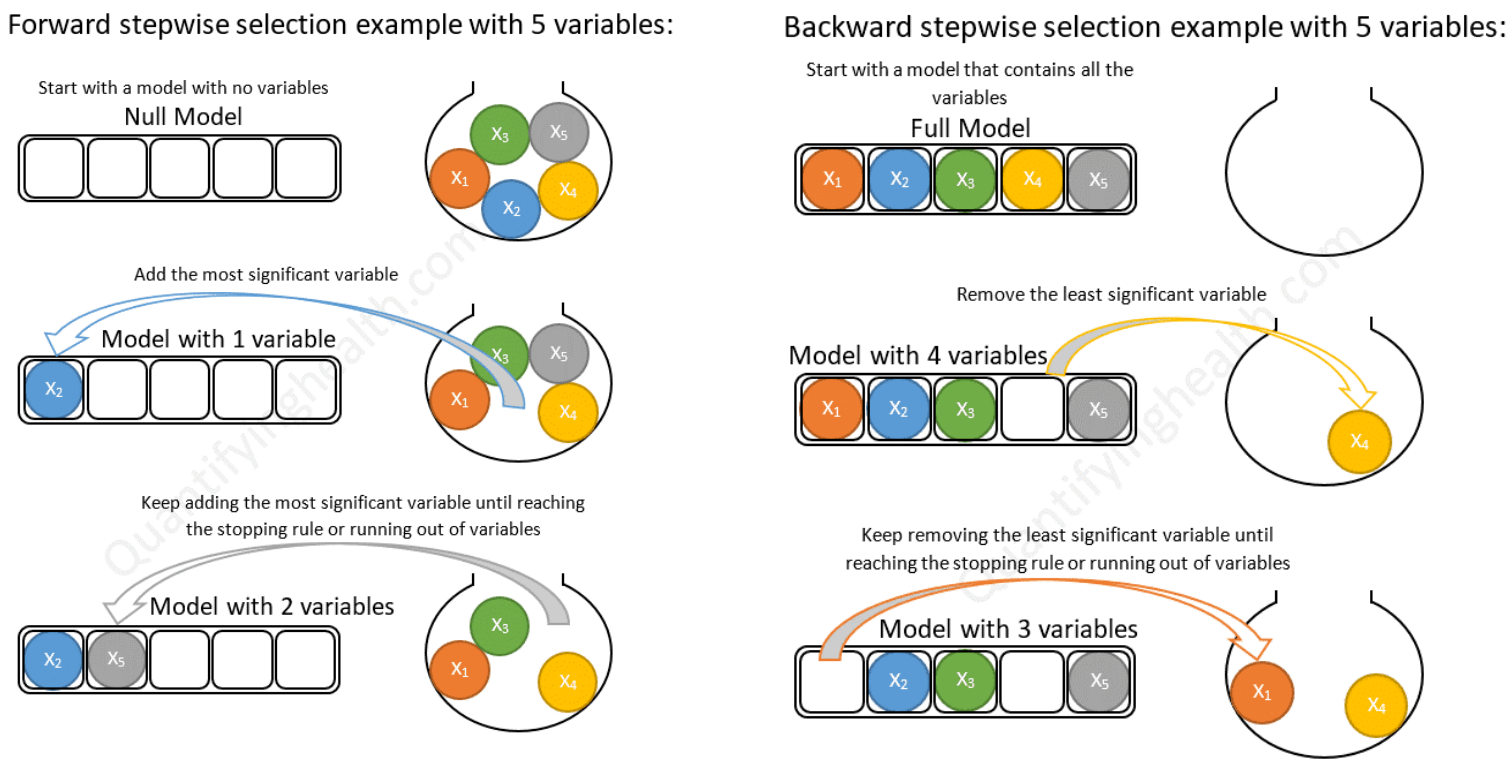
- Which is a good feature ?
  - Good features has clear class boundary



- Filter method
  - Evaluate each feature and select best *n* features
  - Easy and fast
  - Problem : does not consider feature interaction
    - Best{ 1,2,5 } can be better than Best{1,2,3}

1. Feature selection

- Feature subset selection
  - Forward selection
  - Backward elimination



https://quantifyinghealth.com/stepwise-selection/

1. Feature selection

- Scikit-learn
  - The classes in the sklearn.feature_selection module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.
  - https://scikit-learn.org/stable/modules/feature_selection.html

1.13. Feature selection

1.13.1. Removing features with low variance

1.13.2. Univariate feature selection

1.13.3. Recursive feature elimination

1.13.4. Feature selection using SelectFromModel

1.13.5. Feature selection as part of a pipeline

1. Feature selection

- Evaluation functions for filter method

See also:

`f_classif`
ANOVA F-value between label/feature for classification tasks.

`mutual_info_classif`
Mutual information for a discrete target.

`chi2`
Chi-squared stats of non-negative features for classification tasks.

`f_regression`
F-value between label/feature for regression tasks.

`mutual_info_regression`
Mutual information for a continuous target.

`SelectPercentile`
Select features based on percentile of the highest scores.

`SelectFpr`
Select features based on a false positive rate test.

`SelectFdr`
Select features based on an estimated false discovery rate.

`SelectFwe`
Select features based on family-wise error rate.

`GenericUnivariateSelect`
Univariate feature selector with configurable mode.

1. Feature selection

07.feature_selection.py

```
# Feature selection Example

import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

# prepare the dataset
df = pd.read_csv('D:/data/PimaIndiansDiabetes.csv.csv')
print(df.head())
print(df.columns)   # column names

df_X = df.loc[:, df.columns != 'diabetes']
df_y = df['diabetes']

# whole features
model = LogisticRegression(solver='lbfgs', max_iter=500)
scores = cross_val_score(model, df_X, df_y, cv=5)
print("Acc: "+str(scores.mean()))
```

Solver 는 모델의 최적화 방법을 지정하는 매개변수로 default 가 'lbfgs' 임.

```
In [314]: print("Acc: "+str(scores.mean()))
Acc: 0.7721925133689839
```

1. Feature selection

```
#######################################################################
# feature selection by filter method
#######################################################################
# feature evaluation method : chi-square
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

test = SelectKBest(score_func=chi2, k=df_X.shape[1])
fit = test.fit(df_X, df_y)
# summarize evaluation scores
print(np.round(fit.scores_, 3))

f_order = np.argsort(-fit.scores_)  # sort index by decreasing order
sorted_columns = df.columns[f_order]
```

```
In [281]: print(np.round(fit.scores_, 3))
[ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393  181.304]
```

```
In [289]: print(sorted_columns.tolist())
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps', 'pressure', 'pedigree']
```

1. Feature selection

```
# test classification accuracy by selected features (RF)
model = LogisticRegression(solver='lbfgs', max_iter=500)
for i in range(1, df_X.shape[1]+1):
    fs = sorted_columns[0:i]
    df_X_selected = df_X[fs]
    scores = cross_val_score(model, df_X_selected, df_y, cv=5)
    print(fs.tolist())
    print(np.round(scores.mean(), 4))
```

```
['insulin']
0.6563
['insulin', 'glucose']
0.7475
['insulin', 'glucose', 'age']
0.7371
['insulin', 'glucose', 'age', 'mass']
0.7683
['insulin', 'glucose', 'age', 'mass', 'pregnant']
0.7709
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps']
0.7696
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps', 'pressure']
0.7748
['insulin', 'glucose', 'age', 'mass', 'pregnant', 'triceps', 'pressure', 'pedigree']
0.7722
```

1. Feature selection

```
#######################################################################
# Backward elimination (Recursive Feature Elimination)
#######################################################################
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs', max_iter=500)
rfe = RFE(model, n_features_to_select=4)
fit = rfe.fit(df_X, df_y)
print("Num Features: %d" % fit.n_features_)
fs = df_X.columns[fit.support_].tolist()   # selected features
print("Selected Features: %s" % fs)

scores = cross_val_score(model, df_X[fs], df_y, cv=5)
print("Acc: "+str(scores.mean()))
```

```
In [299]: print("Num Features: %d" % fit.n_features_)
Num Features: 4
```

```
In [301]: print("Selected Features: %s" % fs)
Selected Features: ['pregnant', 'glucose', 'mass', 'pedigree']

In [304]: print("Acc: "+str(scores.mean()))
Acc: 0.7695526695526695
```

5. Feature selection

- RFE

Parameters:

estimator : *object*

A supervised learning estimator with a `fit` method that provides information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

n_features_to_select : *int or None (default=None)*

The number of features to select. If `None`, half of the features are selected.

step : *int or float, optional (default=1)*

If greater than or equal to 1, then `step` corresponds to the (integer) number of features to remove at each iteration. If within (0.0, 1.0), then `step` corresponds to the percentage (rounded down) of features to remove at each iteration.

verbose : *int, (default=0)*

Controls verbosity of output.

```
rfe = RFE(model, n_features_to_select=4)
```

1. Feature selection

```
#######################################################################
# Forward selection
#######################################################################
# please install 'mlxtend' moudle

from mlxtend.feature_selection import SequentialFeatureSelector as SFS

model = LogisticRegression(solver='lbfgs', max_iter=500)
sfs1 = SFS(model,
           k_features=5,          # number of features
           verbose=2,
           scoring='accuracy',
           cv=5)

sfs1 = sfs1.fit(df_X, df_y, custom_feature_names=df_X.columns)
sfs1.subsets_              # selection process
sfs1.k_feature_idx_        # selected feature index
sfs1.k_feature_names_      # selected feature name
```

1. Feature selection

```
In [352]: sfs1.subsets_              # selection process
Out[352]:
{1: {'feature_idx': (1,),
  'cv_scores': array([0.708, 0.708, 0.766, 0.771, 0.784]),
  'avg_score': 0.7474747474747474,
  'feature_names': ('glucose',)},
 2: {'feature_idx': (1, 5),
  'cv_scores': array([0.773, 0.734, 0.766, 0.784, 0.739]),
  'avg_score': 0.7591206179441474,
  'feature_names': ('glucose', 'mass')},
 3: {'feature_idx': (1, 5, 7),
  'cv_scores': array([0.773, 0.734, 0.74 , 0.804, 0.791]),
  'avg_score': 0.7683048977166624,
  'feature_names': ('glucose', 'mass', 'age')},
 4: {'feature_idx': (1, 4, 5, 7),
  'cv_scores': array([0.766, 0.734, 0.753, 0.804, 0.784]),
  'avg_score': 0.7682964094728801,
  'feature_names': ('glucose', 'insulin', 'mass', 'age')},
 5: {'feature_idx': (0, 1, 4, 5, 7),
  'cv_scores': array([0.753, 0.74 , 0.786, 0.791, 0.784]),
  'avg_score': 0.7708768355827178,
  'feature_names': ('pregnant', 'glucose', 'insulin', 'mass', 'age')}}

In [353]: sfs1.k_feature_idx_        # selected feature index
Out[353]: (0, 1, 4, 5, 7)

In [354]: sfs1.k_feature_names_      # selected feature name
Out[354]: ('pregnant', 'glucose', 'insulin', 'mass', 'age')
```

1. Feature selection

- SequentialFeatureSelector (SFS)
  - http://rasbt.github.io/mlxtend/api_subpackages/mlxtend.feature_selection/#sequentialfeatureselector

Parameters

- `estimator` : scikit-learn classifier or regressor
- `k_features` : int or tuple or str (default: 1)
- `forward` : bool (default: True)

  Forward selection if True, backward selection otherwise
- `floating` : bool (default: False)

  Adds a conditional exclusion/inclusion if True.
- `verbose` : int (default: 0), level of verbosity to use in logging.

  If 0, no output, if 1 number of features in current set, if 2 detailed logging i ncluding timestamp and cv scores at step.
- `scoring` : str, callable, or None (default: None)

```
sfs1 = SFS(model,
           k_features=5,
           verbose=2,
           scoring='accuracy',
           cv=5)
```

1. Feature selection

```
scores = cross_val_score(model, df_X[list(sfs1.k_feature_names_)], df_y, cv=5)
print("Acc: "+str(scores.mean()))
```

```
In [356]: print("Acc: "+str(scores.mean()))
Acc: 0.7708768355827178
```

- Note. mlxtend 설치

```
선택 Anaconda Prompt (Miniconda3)

(base) C:₩Users₩mango>pip install mlxtend
```

1. Feature selection

- Summary

	# of feature	accuracy
Whole features	8	0.772
Filter method	7	0.775
Forward selection	5	0.771
Backward elimination	4	0.770

- Scikit-learn RFE use 'importance of each feature'
  - Applied models should have coef_ attribute or feature_importances_
  - KNN cannot use RFE

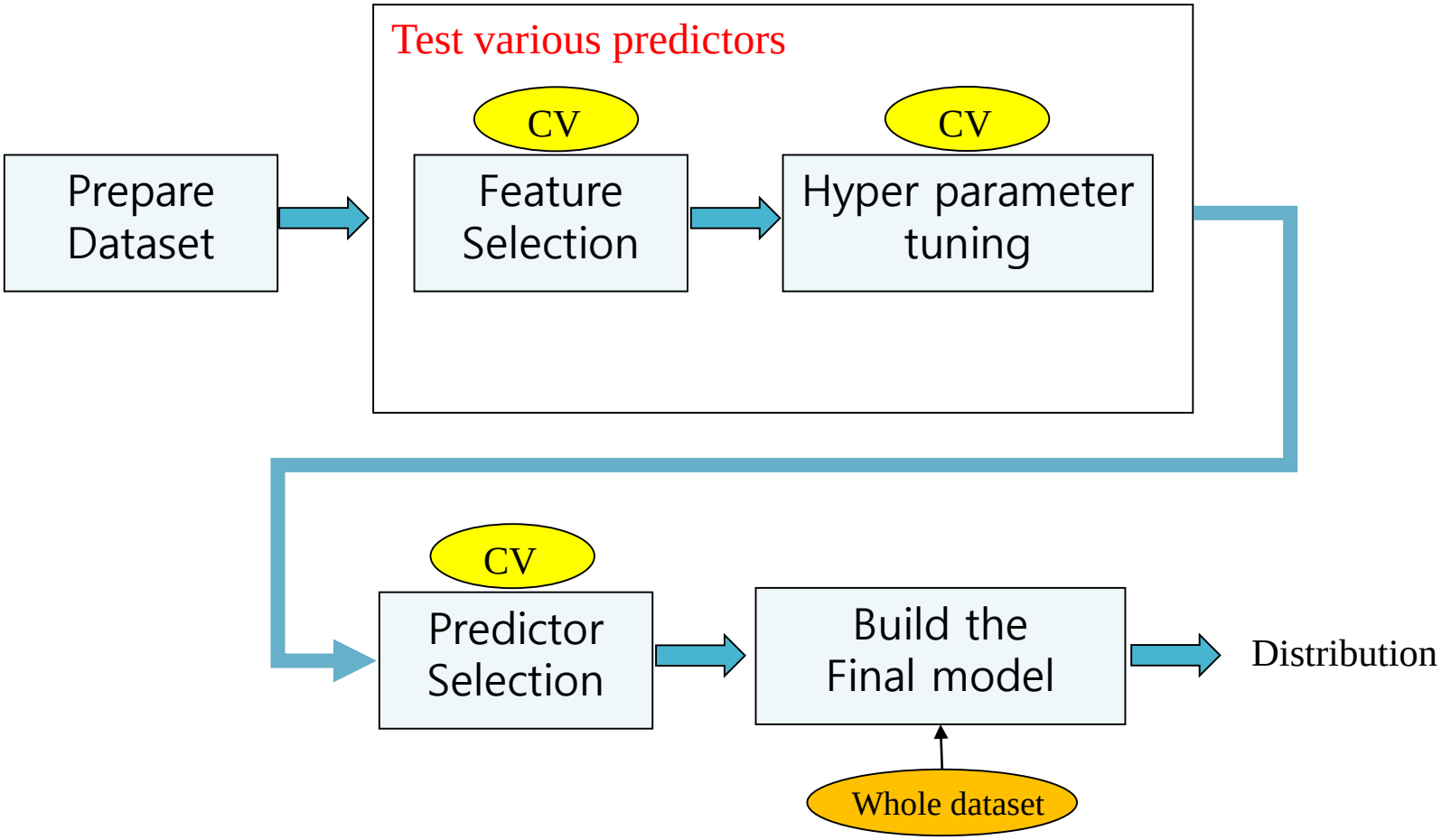
1. Feature selection

- 기타 feature selection 방법
  - MRMR
    - https://pypi.org/project/pymrmr/

  - BORUTA
    - https://towardsdatascience.com/feature-selection-with-boruta-in-python-676e3877e596
    -

Conclusion

- Practical model development process



2. Voting Classifier

scikit-learn 0.24.2
Other versions

Please **cite us** if you use the software.

https://scikit-learn.org/stable/modules/ensemble.html#bagging-meta-estimator

2. Voting Classifier

- Simple voting
  - 여러 종류의 예측 알고리즘을 이용하여 모델을 만든 뒤 각 모델의 예측 결과로 투표 (classification problem) 또는 평균 (regression problem) 을 구함

- 실습을 위한 문제 정의
  - 유방암 데이터셋 (breast cancer) 에 대해 LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier 3개의 모델을 만들고 예측결과를 취합하여 (soft voting) 최종 진단 결과를 내린다.

Classes	2
Samples per class	212(M),357(B)
Samples total	569
Dimensionality	30
Features	real, positive

2. Voting Classifier

```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
import numpy as np

# load dataset
df_X, df_y  = load_breast_cancer(return_X_y=True)

# scaling data
sc_data = StandardScaler().fit(df_X)
df_X = sc_data.transform(df_X)
```

2. Voting Classifier

```
# Define single models
clf_lr = LogisticRegression()
clf_knn = KNeighborsClassifier(n_neighbors=1)
clf_dt = DecisionTreeClassifier(random_state=1)

# Define voting classifer
clf_voting = VotingClassifier(
              estimators=[('LR', clf_lr),
                          ('KNN', clf_knn),
                          ('DT', clf_dt)],
              voting='soft')
```

[Voting 방법]
'hard' : 하나의 classifier 가 한표씩 행사
'soft' : classifier의 지지 확률을 합산 (권장)

2. Voting Classifier

```
## Test Single classifiers ##########################
models = [clf_lr, clf_knn, clf_dt]
for model in models:
    scores = cross_val_score(model, df_X, df_y, cv=5)
    model_name = model.__class__.__name__
    print(f"{model_name} \t : {np.mean(scores)}")
```

```
LogisticRegression    : 0.9806862288464524
KNeighborsClassifier      : 0.9507685142058687
DecisionTreeClassifier    : 0.9173886042539978
```

2. Voting Classifier

```
## Test Voting classifier ############################
voting_scores = cross_val_score(clf_voting, df_X, df_y, cv=5)
print('Voting accuracy', np.mean(voting_scores))
```

```
In [93]: voting_scores = cross_val_score(clf_voting, df_X, df_y, cv=5)
    ...: print('Voting accuracy', np.mean(voting_scores))
Voting accuracy 0.9718832479428661
```

- Note
  - Voting이 항상 좋은 결과를 보여주지는 않는다.
  - Simple voting은 각 모델의 성능을 최대한 끌어올린 뒤 시도한다.

3. Bagging meta-estimator

scikit-learn 0.24.2
Other versions

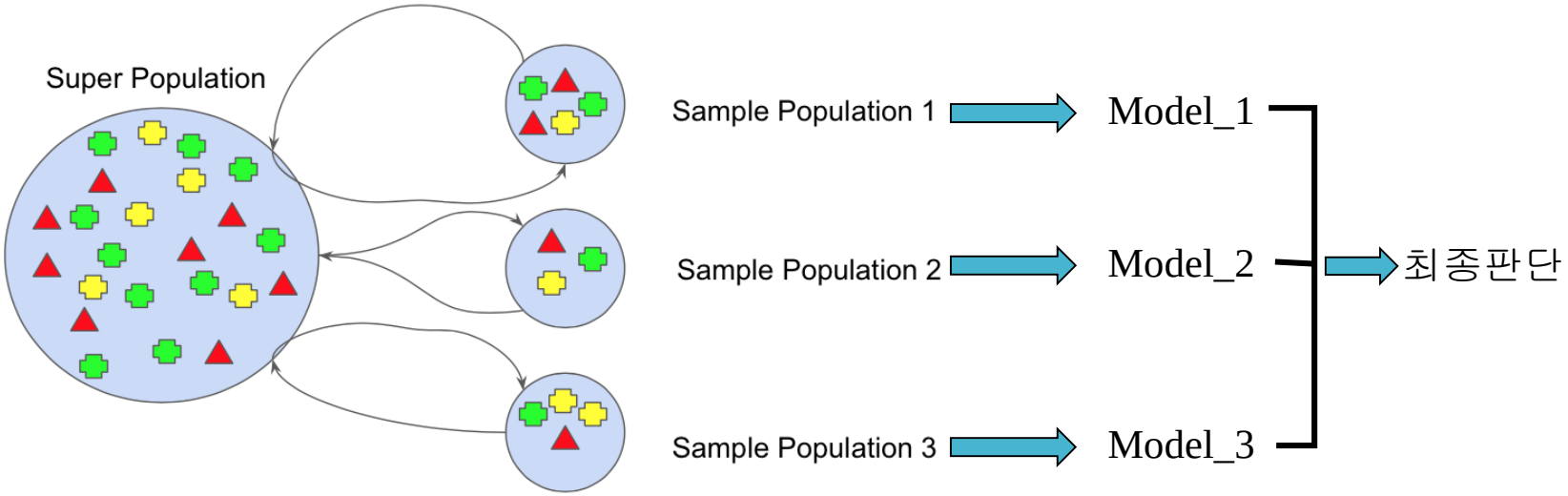
Please **cite us** if you use the software.

1.11. Ensemble methods

https://scikit-learn.org/stable/modules/ensemble.html#bagging-meta-estimator

3. Bagging meta-estimator

- Idea
  - 동일한 데이터셋에 대해 sampling을 하여 다수의 데이터셋을 구성한 뒤 단일 알고리즘으로 모델 생성
  - 각 모델의 결과를 종합하여 최종 의사결정 (예측)



이미지출처: https://jjeongil.tistory.com/909

3. Bagging meta-estimator

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
import numpy as np

# load dataset
df_X, df_y  = load_breast_cancer(return_X_y=True)

# scaling data
sc_data = StandardScaler().fit(df_X)
df_X = sc_data.transform(df_X)

# evaluate base model
model_base = KNeighborsClassifier()
scores = cross_val_score(model_base, df_X, df_y, cv=5)
print(np.mean(scores))
```

```
In [95]: print(np.mean(scores))
0.9648501785437045
```

3. Bagging meta-estimator

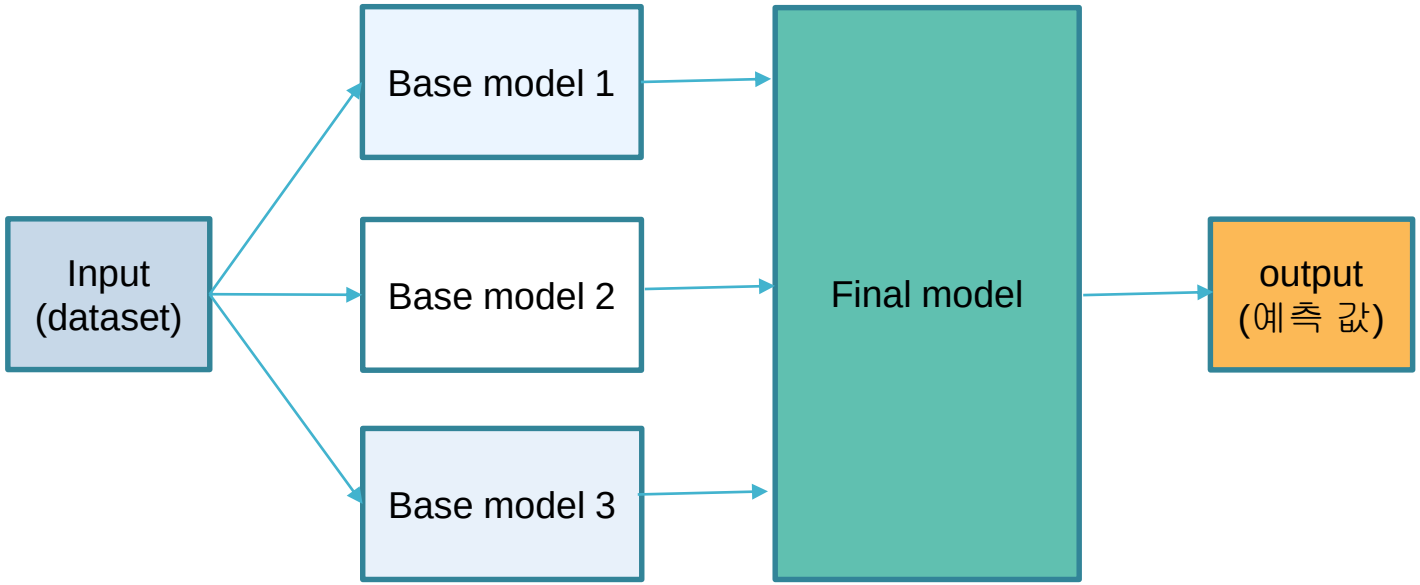
```
# bagging predictor
model_bagging = BaggingClassifier(KNeighborsClassifier(),
                                  n_estimators = 100,        # 모델 수
                                  max_samples=0.5,           # 인스턴스 선택 비율
                                  max_features=1.0,          # feature 선택 비율
                                  n_jobs=-1,
                                  random_state=123)

scores_bagging = cross_val_score(model_bagging, df_X, df_y, cv=5)
print(np.mean(scores_bagging))
```

```
In [97]: print(np.mean(scores_bagging))
0.9666356155876417
```

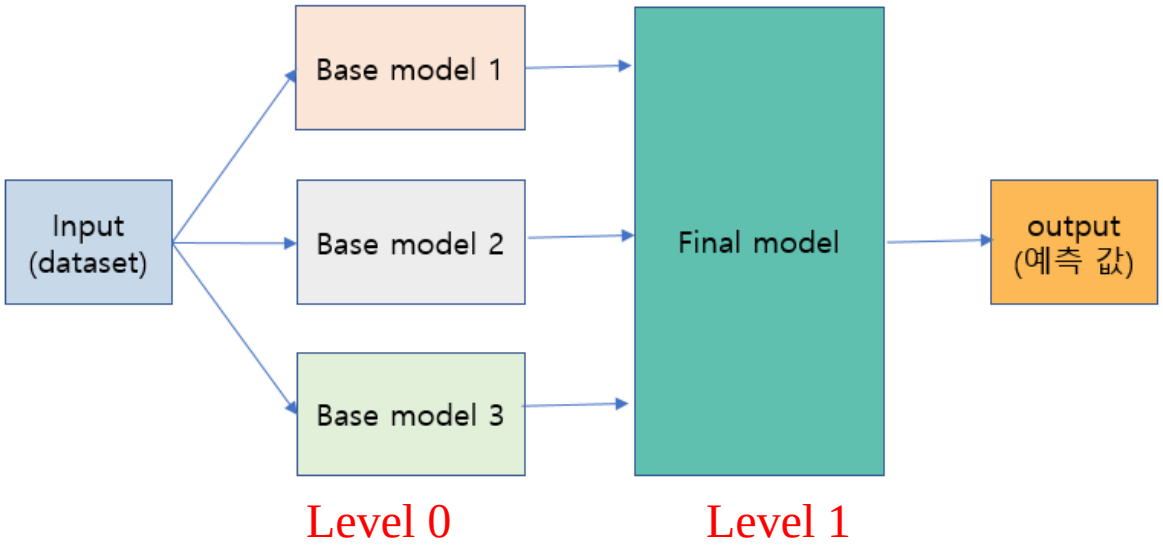
4. Model stacking

- 모델 스태킹이란
  - 앙상블 방법중의 하나
  - stacked generalization 이라고도 한다
  - 여러 모델의 예측값을 input으로 해서 새로운 모델로 학습
  - 과적합(overfitting) 방지를 위해 사용되기도 함



4. Model stacking

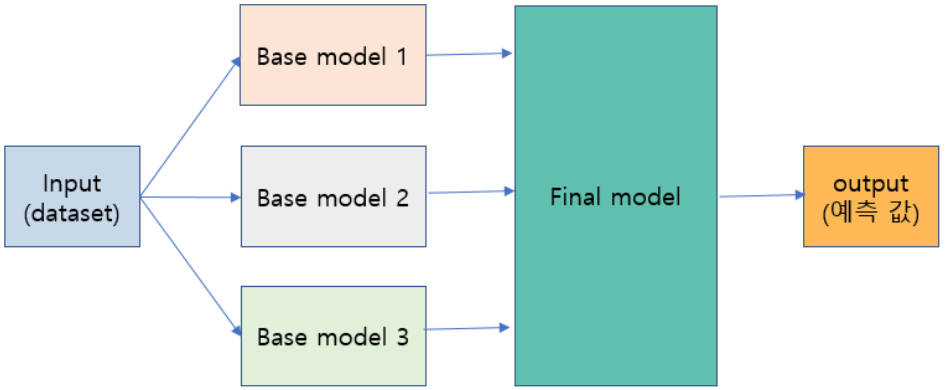
- 모델 스태킹 절차
  - Level 0 : training dataset을 이용하여 sub-model 예측 결과를 생성
  - Level 1 : level 0의 output 결과가 level 1의 input 값이 된다. level 0의 output을 training data로 사용하여 meta learner 모델을 생성



4. Model stacking

- 모델 스태킹 절차
  - level 0 모델들은 되도록 다양한 예측 결과를 meta learner에서 input 값으로 활용할 수 있도록 <u>각기 다른 알고리즘을 사용하는 것이 좋음</u>
  - 각 sub-model의 결과를 결합하는 meta learner 모델은 simple linear model을 사용하는 것이 일반적

- Level 0 출력값 처리
  - regression의 경우 각 Level 0 출력 값이 그대로 Level 1의 input 값으로 사용
  - Classification의 경우 Level 1 출력 값을 그대로 사용하는 것 보다는 label probabilities 시용하는 것이 더 효과적



Level 0 Level 1

4. Model stacking

- Scikit-learn 은 모델 스태킹을 편리하게 할 수 있는 방법 제공
  - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html

sklearn.ensemble.StackingClassifier ¶

class sklearn.ensemble.StackingClassifier(*estimators, final_estimator=None, *, cv=None, stack_method='auto', n_jobs=None, passthrough=False, verbose=0*) [source]

Stack of estimators with a final classifier.

Stacked generalization consists in stacking the output of individual estimator and use a classifier to compute the final prediction. Stacking allows to use the strength of each individual estimator by using their output as input of a final estimator.

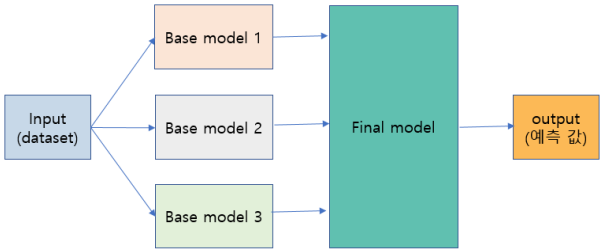
Note that `estimators_` are fitted on the full `X` while `final_estimator_` is trained using cross-validated predictions of the base estimators using `cross_val_predict`.

Read more in the User Guide.

4. Model stacking

- StackingClassifier() 주요 매개변수

매개변수	설명
estimators	Level 0에 사용할 predictor들
final_estimator	Level 1에 사용할 predictor (default: LogisticRegression)
cv	cross validation 값 (미지정시 5-fold cross validation 적용)
stack_method	‘auto’: 아래의 방법중 하나를 적용 ‘predict_proba’: Level 0 의 Label probability를 input으로 사용 ‘decision_function’: Level 0 의 출력을 처리하는 함수 적용 ‘predict’: Level 0 출력을 그대로 input으로 사용
n_jobs	multi-processing (-1: 모든 core사용)
passthrough	False: Level 0 의 출력만 Level 1 의 학습데이터로 사용 True: Level 0 의 출력+ original dataset 이 Level 1 의 학습데이터로 사용



4. Model stacking

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

# prepare dataset
df_X, df_y  = load_breast_cancer(return_X_y=True)
```

4. Model stacking

```
# define level 0
estimators = [
     ('rf', RandomForestClassifier(n_estimators=10,
                                   random_state=42)),
     ('svr', make_pipeline(StandardScaler(),
                           LinearSVC(random_state=42)))]

# define model
model_1 = StackingClassifier(
      estimators=estimators,
      final_estimator=LogisticRegression())

scores_1 = cross_val_score(model_1, df_X, df_y, cv=5)
print(np.mean(scores_1))
```

```
In [102]: print(np.mean(scores_1))
0.971914299021891
```

4. Model stacking

```
model_2 = StackingClassifier(
          estimators=estimators,
          final_estimator=LogisticRegression(max_iter=500),
          passthrough=True)

scores_2 = cross_val_score(model_2, df_X, df_y, cv=5)
print(np.mean(scores_2))
```

```
In [108]: print(np.mean(scores_2))
0.9630957925787922
```

- LogisticRegression 의 max_iter : 정답에 수렴할때까지 몇번이나 계산(경사하강법)을 반복할지를 결정

4. Model stacking

```
estimators = [
     ('rf', RandomForestClassifier(n_estimators=10,
                                   random_state=42)),
     ('svr', make_pipeline(StandardScaler(),
                           LinearSVC(random_state=42))),
     ('lr', LogisticRegression(max_iter=5000))]

from xgboost import XGBClassifier

model_3 = StackingClassifier(
         estimators=estimators,
         final_estimator=XGBClassifier(use_label_encoder=False,
                                       eval_metric='logloss',
                                       random_state=42),
         passthrough=True)

scores_3 = cross_val_score(model_3, df_X, df_y, cv=5)
print(np.mean(scores_3))
```

```
In [110]: print(np.mean(scores_3))
0.9806707033069397
```

정리

- Practical model development process