Selecting features for model performance

DIMENSIONALITY REDUCTION IN PYTHON

Jeroen Boeye
Machine Learning Engineer, Faktion

datacamp

Ansur dataset sample

Gender	chestdepth	handlength	neckcircumference	shoulderlength	earlength
Female	243	176	326	136	62
Female	219	177	325	135	58
Male	259	193	400	145	71
Male	253	195	380	141	62

Pre-processing the data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

평균이 0이고 분산이 1이되도록 training feature dataset X_train을 표준화 함

```
X_train_std = scaler.fit_transform(X_train)
```

Creating a logistic regression model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

lr = LogisticRegression()
lr.fit(X_train_std, y_train)

X_test_std = scaler.transform(X_test)


y_pred = lr.predict(X_test_std)
print(accuracy_score(y_test, y_pred))
```

trasform()은 fit()으로 설정된 정보를 이용하여 데이터 변환을 하라는 명령

표준화의 경우에는 평균을 빼고 표준편차로 나누고, 정규화의 경우에는 최솟값을 빼고 최댓값과 최솟값의 차이로 나누어 줌. (fit_transform() : train data에서만 사용할 수 있음. test data에서는 훈련 데이터에서 설정된 정보를 가지고 transform()을 해야함

```
0.99
```

Inspecting the feature coefficients

```
print(lr.coef_)
```

logistic regression 모델이 결정 함수에 사용하는 feature 계수를 보면 일부 값이 0에 가까움
model이 예측할 때 이러한 계수는 feature 값과 곱하기 때문에 계수가 0에 가까운 feature은 최종 결과에 거의 기여하지 않음

```
array([[-3.   ,  0.14,  7.46,  1.22,  0.87]])
```

```
print(dict(zip(X.columns, abs(lr.coef_[0]))))
```

zip()을 사용하여 feature간의 계수를 dictionary로 만듬

```
{'chestdepth': 3.0,
 'handlength': 0.14,
 'neckcircumference': 7.46,
 'shoulderlength': 1.22,
 'earlength': 0.87}
```

예측에 가장 적은 영향을 미치는 feature를 제거하려면 계수가 제일 낮은handlength를 제거해야함

Features that contribute little to a model

```
X.drop('handlength', axis=1, inplace=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

lr.fit(scaler.fit_transform(X_train), y_train)

print(accuracy_score(y_test, lr.predict(scaler.transform(X_test))))
```

```
0.99
```

handlength를 제거하면 model accuracy가 99%로 변경되지않지만 dataset의 복잡성은 감소함.

Recursive Feature Elimination

```
from sklearn.feature_selection import RFE

rfe = RFE(estimator=LogisticRegression(), n_features_to_select=2, verbose=1)
rfe.fit(X_train_std, y_train)
```

"제귀 feature 제거"를 위한 RFE는 feature 계수 또는 feature importance값을 생성하는 모든 model에 감길 수 있는 feature selection 알고리즘

사용하고자 하는 모델과 선택하고자 하는 feature 수를 전달

data를 fit하는 동안 내부 모델을 먼저 fit한 다음 가장 약한 계수로 feature을 삭제하는 process를 반복함

```
Fitting estimator with 5 features.
Fitting estimator with 4 features.
Fitting estimator with 3 features.
```

RFE의 상세 파라미터를 0보다 높게 설정하면 fitting하는 동안 feature가 하나씩 drop 되는 것을 볼 수 있음
model을 한 번 적합시킨 후 계수가 가장 높은 2개의 feature만 유지하기로 결정할 수 있지만,
한 feature을 떨어뜨리면 다른 계수가 변경되기 때문에 재귀방법이 더 안전함

Dropping a feature will affect other feature's coefficients

Inspecting the RFE results

```
X.columns[rfe.support_]
```

True/False 값을 포함하는 support_ 속성을 확인하여 데이터 세트에 어떤 기능이 유지되었는지 확인

```
Index(['chestdepth', 'neckcircumference'], dtype='object')
```

```
print(dict(zip(X.columns, rfe.ranking_)))
```

zip 함수를 한 번 더 사용하면 RFE의 ranking_ 속성을 확인하여 어떤 반복에서 기능이 삭제되었는지 확인할 수 있음

```
{'chestdepth': 1,
 'handlength': 4,
 'neckcircumference': 1,
 'shoulderlength': 2,
 'earlength': 3}
```

값이 1이면 형상이 끝까지 데이터 세트에 유지되었음을 의미하며, 값이 높으면 형상이 초기에 삭제되었음을 의미

```
print(accuracy_score(y_test, rfe.predict(X_test_std)))
```

```
0.99
```

chestdepth와 neckcircumference 두가지 feature 만으로 model의 정확도를 test할 수 있는데, 여전히 정확도는 99%
(손상되지 않았고, 다른 3개의 feature이 model에 전혀 영향을 미치지 않았음을 의미

Let's practice!

DIMENSIONALITY REDUCTION IN PYTHON

some models will perform feature selection by design to avoid overfitting

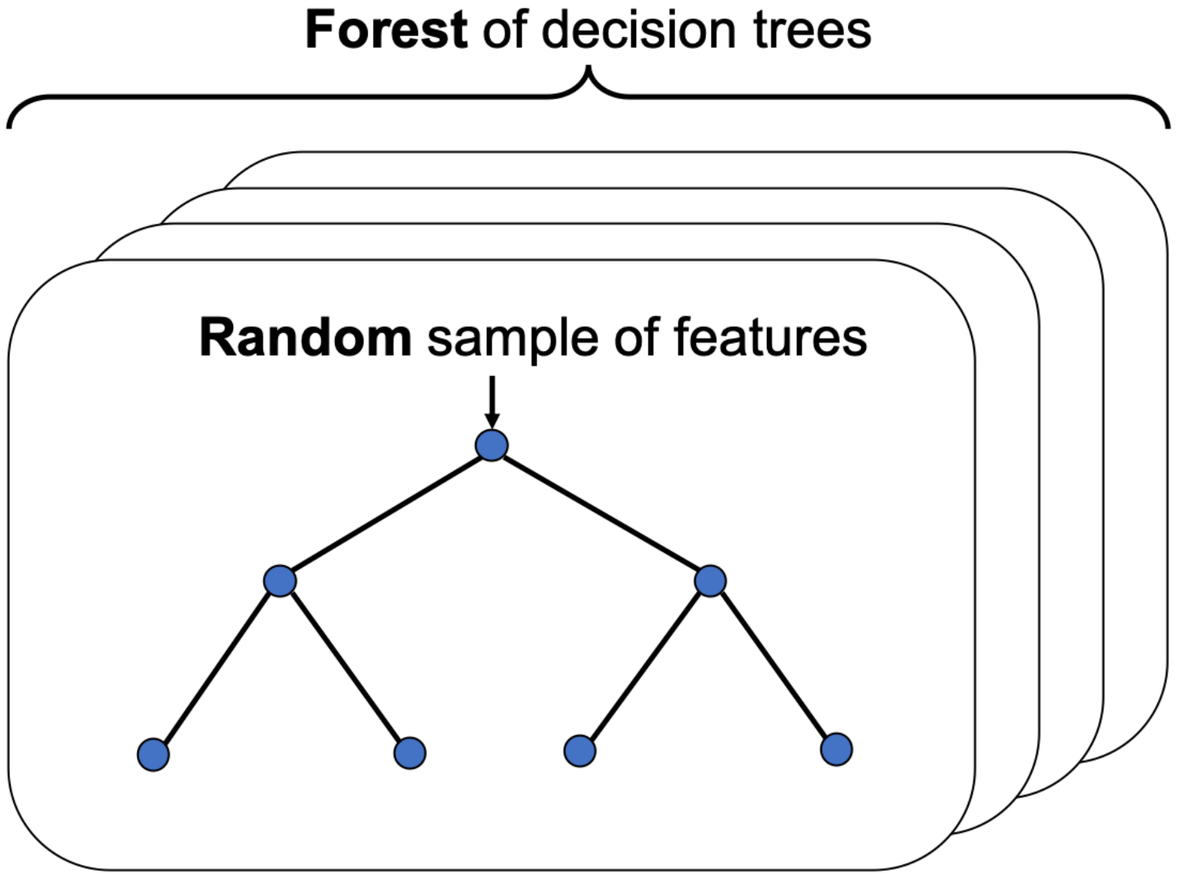
Tree-based feature selection

DIMENSIONALITY REDUCTION IN PYTHON

Jeroen Boeye
Machine Learning Engineer, Faktion

datacamp

Random forest classifier



다양한 random feature의 하위 집합을 다수의 의사 결정 트리에 전달하는 앙상블 모델
예측을 하기 위해 개별 tree의 예측에 대해 집계
여기서 RF 예에는 4개의 의사 결정 트리가 있음
RF 는 종종 매우 정확하고 기본 Scikit-learn 설정을 사용하더라도 과적합을 방지함

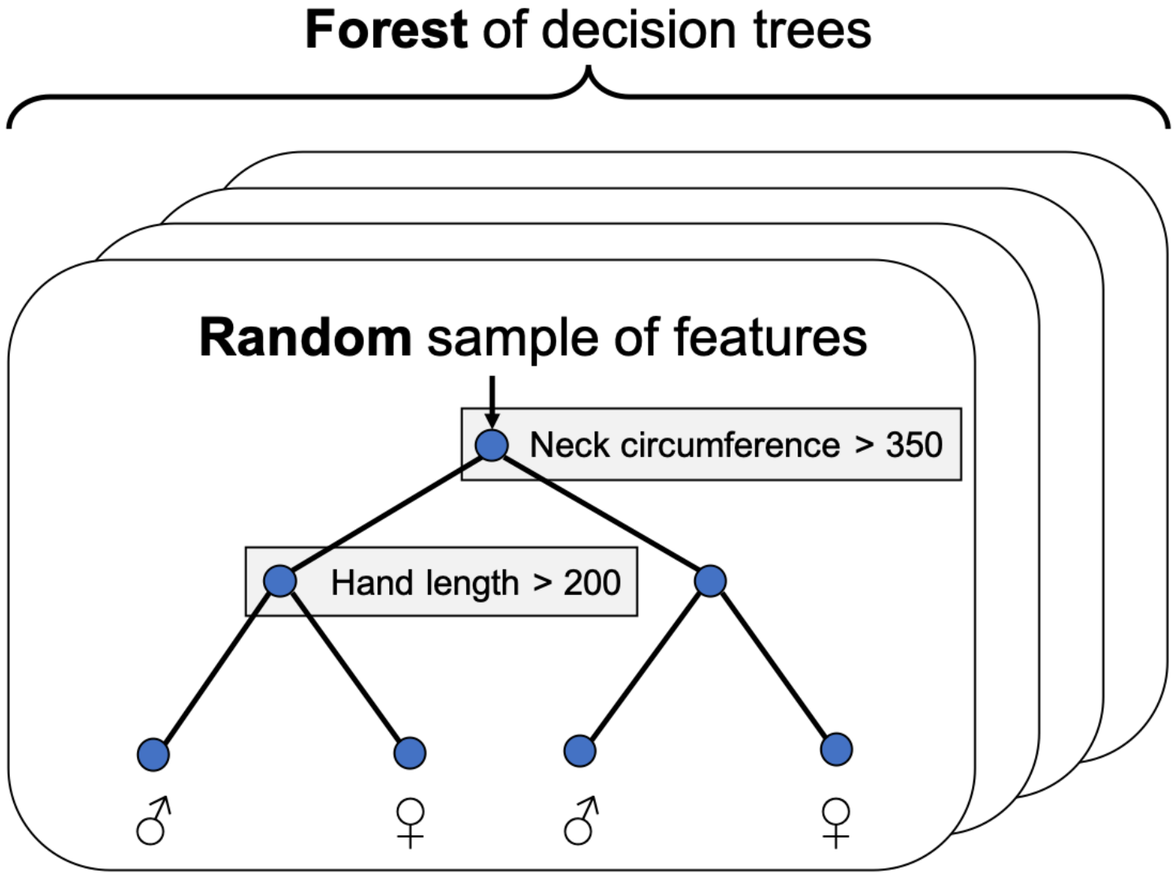
Random forest classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

rf = RandomForestClassifier()

rf.fit(X_train, y_train)

print(accuracy_score(y_test, rf.predict(X_test)))
```

```
0.99
```

차원의 저주를 벗어남

Random forest classifier



첫 번째 결정 node의 목 둘레 feature 와 나중에 사람이 남자인지 여자인지를 결정하기 위해 손 길이를 사용함
다양한 의사 결정 트리 내에서 결정을 내리기 위해 feature가 얼마나 자주 사용되는지 평균화하고,
이것이 tree의 root 근처의 중요한 결정인지 아니면 tree의 작은 branch에서 덜 중요한 결정인지를 고려하여 RF 알고리즘은 feature의 중요도 값을 계산함

Feature importance values

```
rf = RandomForestClassifier()


rf.fit(X_train, y_train)


print(rf.feature_importances_)
```

lr와 마찬가지로 중요하지 않는 feature의 경우 0에 가까우므로
이러한 feature의 중요도 값을 사용하여 feature selection을 수행할 수 있음

```
array([0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.04, 0.  , 0.01, 0.01,
       0.  , 0.  , 0.  , 0.  , 0.01, 0.01, 0.  , 0.  , 0.  , 0.  , 0.05,
       ...
       0.  , 0.14, 0.  , 0.  , 0.  , 0.06, 0.  , 0.  , 0.  , 0.  , 0.  ,
       0.  , 0.07, 0.  , 0.  , 0.01, 0.  ])
```

```
print(sum(rf.feature_importances_))
```

feature importances 값의 장점은 항상 최대 1까지 합해지므로 기본적으로 feature 간에 비교 가능

```
1.0
```

Feature importance as a feature selector

```
mask = rf.feature_importances_ > 0.1

print(mask)
```

```
array([False, False, ..., True, False])
```

임계값 충족하는지

```
X_reduced = X.loc[:, mask]

print(X_reduced.columns)
```

```
Index(['chestheight', 'neckcircumference', 'neckcircumferencebase',
       'shouldercircumference'], dtype='object')
```

RFE with random forests

```
from sklearn.feature_selection import RFE

rfe = RFE(estimator=RandomForestClassifier(),
          n_features_to_select=6, verbose=1)

rfe.fit(X_train,y_train)
```

약한 feature을 삭제하면 다른 feature가 상대적으로 더 중요하거나
덜 중요해질 수 있기 때문에 잘못된 feature을 삭제할 위험을 최소화하려면
가장 중요하지 않은 feature을 한번에 모두 삭제하지 마고 하나씩 삭제해야함.
이를 위해 model은 Recursive Feature Eliminator 또는 RFE()로 감쌀 수 있음.
여기서는 test set 정확도에 영향을 주지 않으면서 feature 수를 6개로 줄임
삭제하려는 각 feature에 대해 model을 한번에 train하면 너무 많은 계산 overhead가
발생할 수 있음

```
Fitting estimator with 94 features.
Fitting estimator with 93 features
...
Fitting estimator with 8 features.
Fitting estimator with 7 features.
```

```
print(accuracy_score(y_test, rfe.predict(X_test))
```

```
0.99
```

RFE with random forests

```
from sklearn.feature_selection import RFE

rfe = RFE(estimator=RandomForestClassifier(), 프로세스 속도를 높이기 위해 “step” 매개변수를 사용
          n_features_to_select=6, step=10, verbose=1)
                                  각 반복에서 가장 중요하지 않은 10가지 feature을 삭제
rfe.fit(X_train,y_train)
```

```
Fitting estimator with 94 features.
Fitting estimator with 84 features.
...
Fitting estimator with 24 features.
Fitting estimator with 14 features.
```

```
print(X.columns[rfe.support_])
```

```
Index(['biacromialbreadth', 'handbreadth', 'handcircumference',
       'neckcircumference', 'neckcircumferencebase', 'shouldercircumference'], dtype='object')
```

Let's practice!

DIMENSIONALITY REDUCTION IN PYTHON

Regularized linear regression

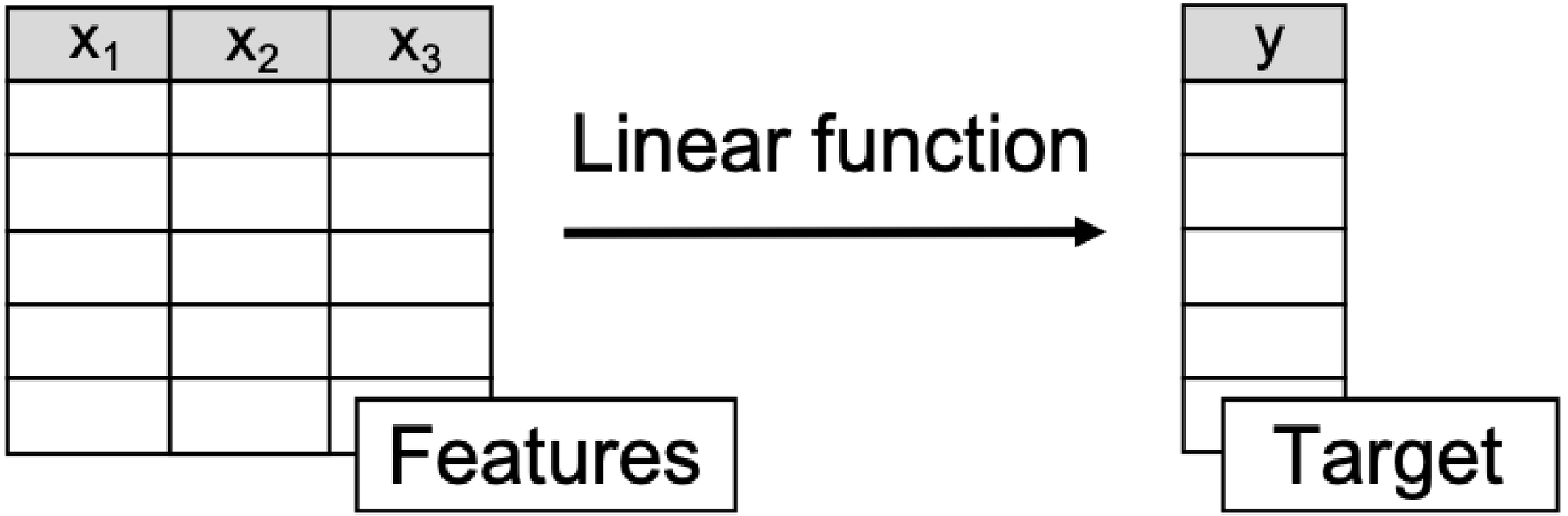
DIMENSIONALITY REDUCTION IN PYTHON

분류 알고리즘을 사용하여 차원을 줄이는 방법에 초점을 맞춤
지금부터는 **regression**에 초점을 맞춤

Jeroen Boeye
Machine Learning Engineer, Faktion

datacamp

Linear model concept



선형 회귀 분석의 작동 방식을 새로 고치기 위해, 우리는 세 개의 입력 값과 목표값 사이의 선형 함수를 도출하는 모델을 만들 것이다.
그러나 우리는 직접 형상 데이터 세트와 선형 함수를 생성하여 모델이 도출하려고 하는 실측을 제어할 수 있을 것이다.

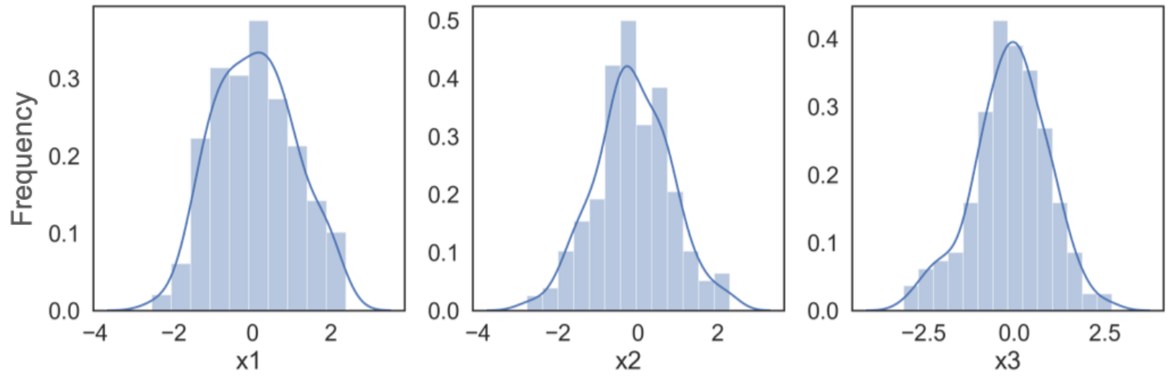
Creating our own dataset

x1	x2	x3
1.76	-0.37	-0.60
0.40	-0.24	-1.12
0.98	1.10	0.77
...	...	...

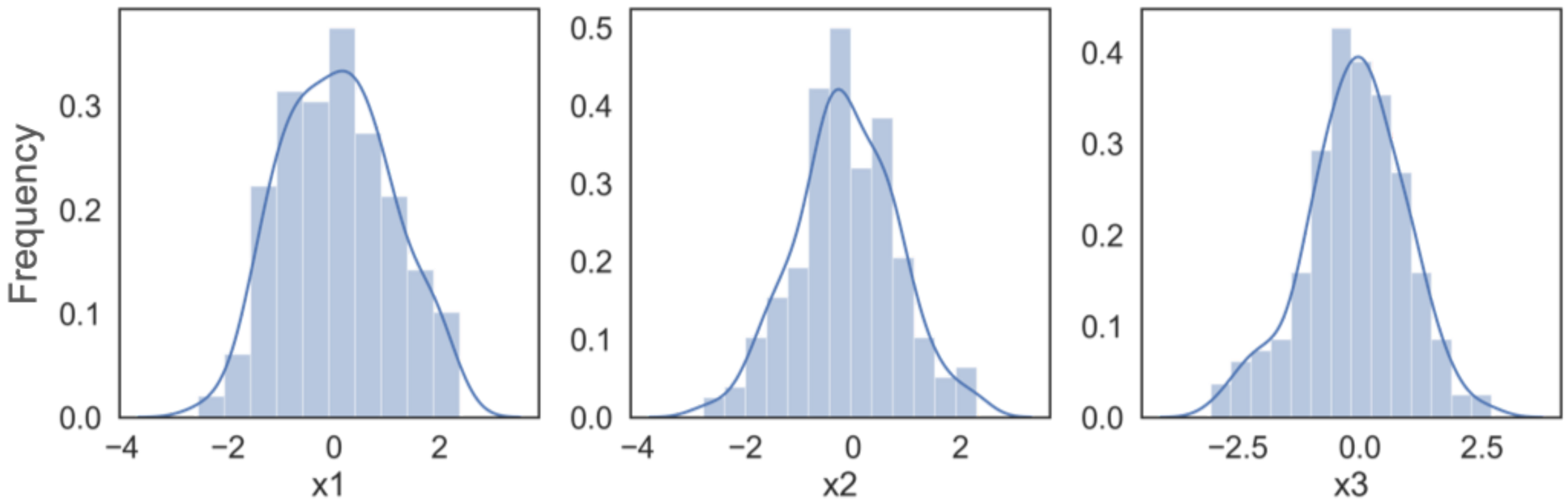
Creating our own dataset

x1	x2	x3
1.76	-0.37	-0.60
0.40	-0.24	-1.12
0.98	1.10	0.77
...	...	...

모두 단순한 정규 분포를 따름



Creating our own dataset



Creating our own target feature:

$$y = 20 + 5x_1 + 2x_2 + 0x_3 + error$$

시작점의 20을 절편이라고 함.
5, 2, 0은 feature의 계수이며, 각각이 대상에 미치는 영향을 결정함.
세 번째 feature은 계수가 0이므로 대상에 전혀 영향을 미치지 않음.
model을 혼동하여 overfit할 수 있으므로 dataset에서 제거하는 것이 가장 좋음.

Linear regression in Python

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train)

# Actual coefficients = [5 2 0]
print(lr.coef_)
```

```
[ 4.95  1.83 -0.05]
```

앞에서 5, 2, 0으로 설정한 세 가지 값이고 model이 꽤 정확하게 추정함.

```
# Actual intercept = 20
print(lr.intercept_)
```

```
19.8
```

Linear regression in Python

```
# Calculates R-squared
print(lr.score(X_test, y_test))
```

```
0.976
```

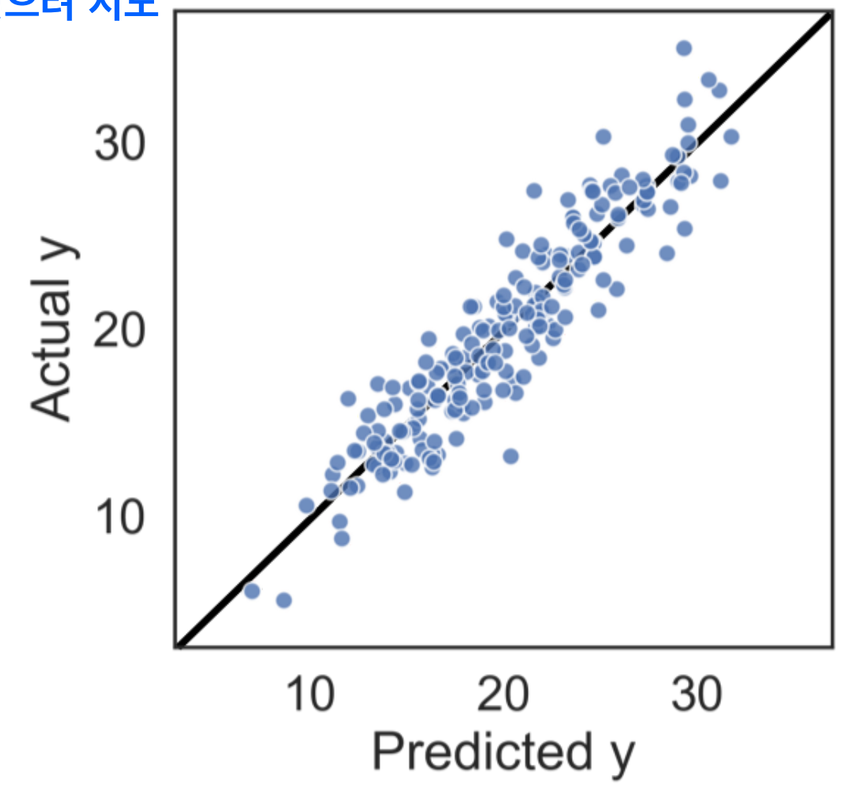
Linear regression in Python

```
from sklearn.linear_model import LinearRegression


lr = LinearRegression()
lr.fit(X_train, y_train)

# Actual coefficients = [5 2 0]
print(lr.coef_)
```

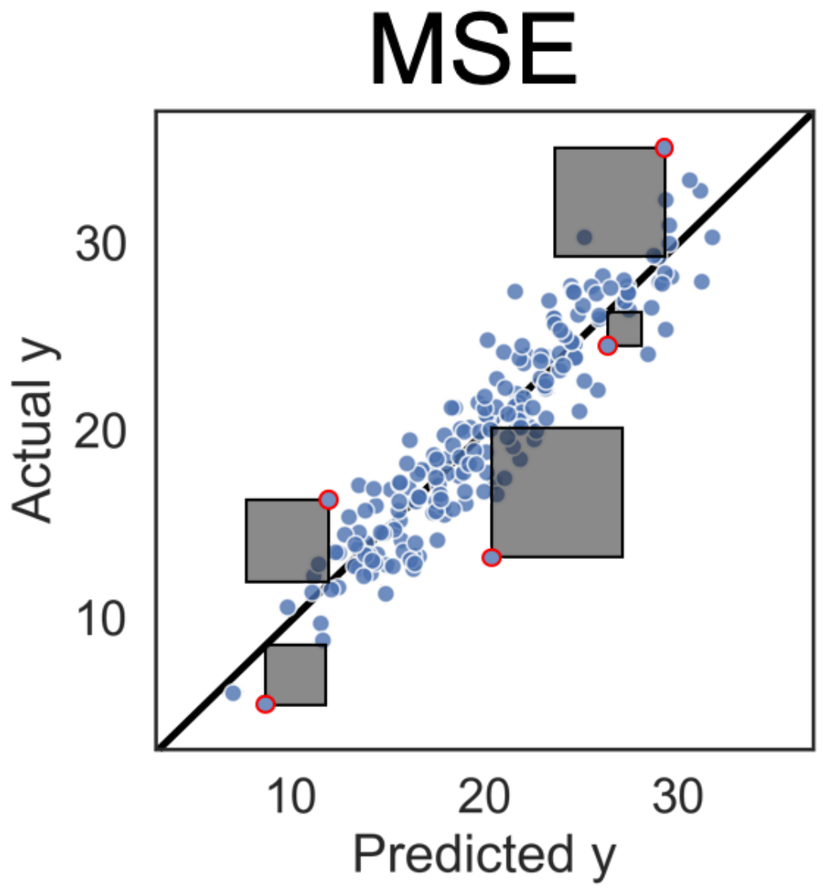
```
[ 4.95  1.83 -0.05]
```

Loss function: Mean Squared Error

model은 loss function을 최소화하여
intercept 와 coefficients에 대한 최적값을 찾으려 시도



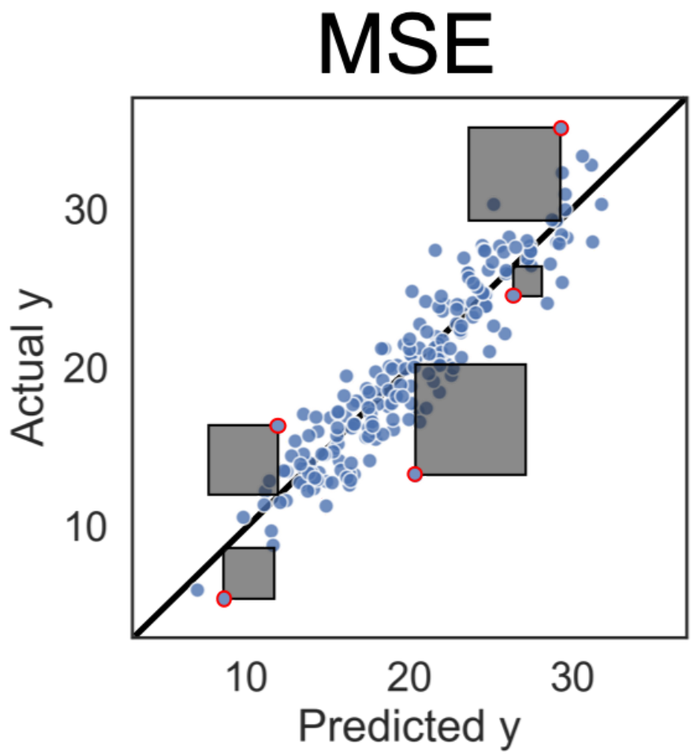
Loss function: Mean Squared Error



MSE는 그래프의 회색 네모인 실제값과 예측값 간의 차이 제곱의 평균 합을 포함함.
그러나 더 이상 새로운 데이터로 일반화되지 않는 경우 모델이 train set에서 매우 정확하지 않기를 바람

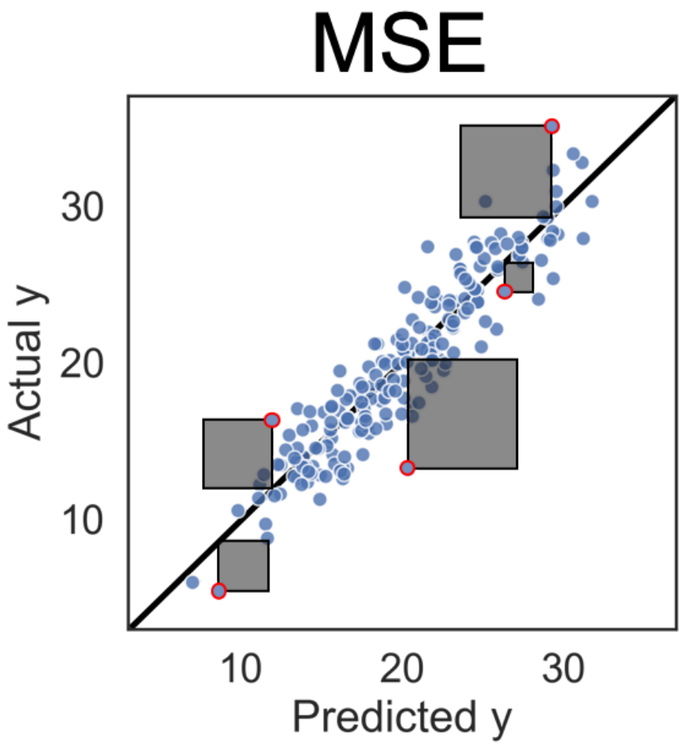
Adding regularization

이를 방지하기 위해 정규화를 도입하면 모델은 MSE를 최소화함으로써 가능한 정확하게 하려고 할 뿐만 아니라 계수를 낮게 유지하여 모형을 단순하게 유지하려고 함



$$\text{Regularization term} \quad + \quad \alpha(|\beta_1|+|\beta_2|+|\beta_3|)$$

Adding regularization



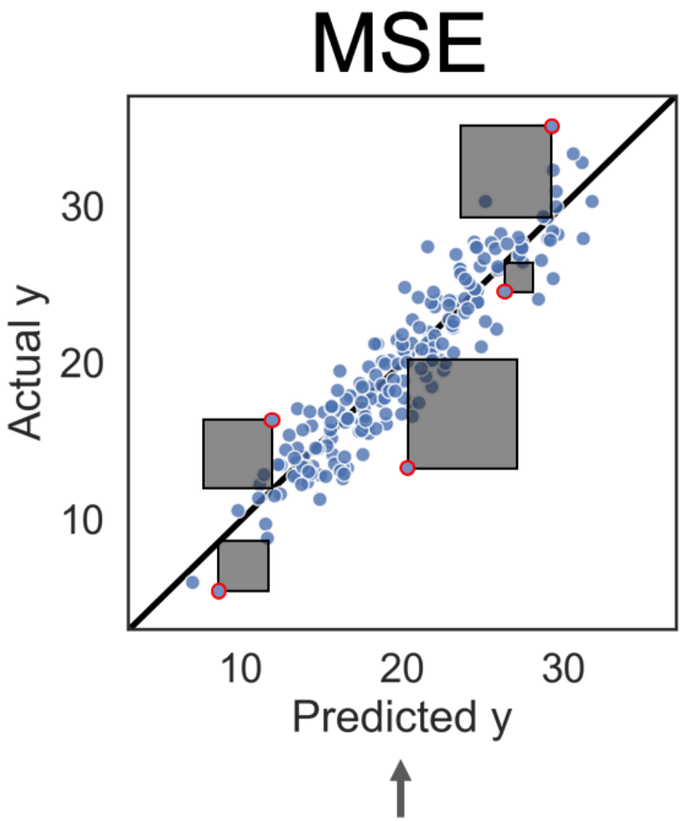
Tries to make model accurate

+ $\overbrace{\alpha(|\beta_1|+|\beta_2|+|\beta_3|)}$

Regularization term

Tries to make model simple

Adding regularization



Tries to make model accurate

$$+ \quad \overbrace{\alpha(|\beta_1|+|\beta_2|+|\beta_3|)}^{\text{Regularization term}}$$

Tries to make model simple

[1] alpha, when it's too low the model might overfit, when it's too high the model might become too simple and inaccurate. One linear model that includes this type of regularization is called Lasso, for least absolute shrinkage

Lasso regressor

```
from sklearn.linear_model import Lasso

la = Lasso()
la.fit(X_train, y_train)

# Actual coefficients = [5 2 0]
print(la.coef_)
```

dataset에 fit하면서 세번째 feature의 계수가 0으로 감소했고,
다른 계수도 감소하여 R 제곱이 더 낮아짐.

```
[4.07 0.59 0.  ]
```

```
print(la.score(X_test, y_test))
```

```
0.861
```

Lasso regressor

```
from sklearn.linear_model import Lasso

la = Lasso(alpha=0.05)
la.fit(X_train, y_train)

# Actual coefficients = [5 2 0]
print(la.coef_)
```

이를 방지하기 위해 알파 파라미터를 변경할 수 있음.
0.05로 설정하면 세 번째 feature는 여전히 무시되지만 다른 계수는
더 적게 감소하고 R 제곱은 다시 올라감

```
[ 4.91  1.76 0.  ]
```

```
print(la.score(X_test, y_test))
```

```
0.974
```

Let's practice!

DIMENSIONALITY REDUCTION IN PYTHON

Combining feature selectors

DIMENSIONALITY REDUCTION IN PYTHON

Jeroen Boeye
Machine Learning Engineer, Faktion

datacamp

Lasso regressor

```
from sklearn.linear_model import Lasso

la = Lasso(alpha=0.05)
la.fit(X_train, y_train)

# Actual coefficients = [5 2 0]
print(la.coef_)
```

```
[ 4.91  1.76 0. ]
```

```
print(la.score(X_test, y_test))
```

```
0.974
```

LassoCV regressor

```
from sklearn.linear_model import LassoCV

lcv = LassoCV()

lcv.fit(X_train, y_train)

print(lcv.alpha_)
```

최적의 값을 갖는 alpha_ 속성을 얻을 수 있음

```
0.09
```

LassoCV regressor

```
mask = lcv.coef_ != 0

print(mask)
```

Lasso regression 분석기가 0 계수를 할당한 feature을 실제로 제거하기 위해,
0이 아닌 모든 계수에 대해 True값을 가지는 mask를 생성

```
[ True True False ]
```

```
reduced_X = X.loc[:, mask]
```

Taking a step back

- Random forest is combination of decision trees.
- We can use combination of models for feature selection too.

Feature selection with LassoCV

```
from sklearn.linear_model import LassoCV

lcv = LassoCV()
lcv.fit(X_train, y_train)

lcv.score(X_test, y_test)
```

```
0.99
```

(R 제곱)

```
lcv_mask = lcv.coef_ != 0
sum(lcv_mask)
```

feature의 계수가 0과 다른지 여부를 알려주는 마스크를 만들면 91개 feature의 66개 feature의 경우임을 알 수 있음.

```
66
```

Feature selection with random forest

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor

rfe_rf = RFE(estimator=RandomForestRegressor(),
             n_features_to_select=66, step=5, verbose=1)
```

model 주위에 Recursive Feature Selector 또는 RFE를 감아 LassoCV() 회귀기와 동일한 수의 피쳐를 선택하도록 했음

```
rfe_rf.fit(X_train, y_train)
rf_mask = rfe_rf.support_
```

Feature selection with gradient boosting

```
from sklearn.feature_selection import RFE
from sklearn.ensemble import GradientBoostingRegressor

rfe_gb = RFE(estimator=GradientBoostingRegressor(),
             n_features_to_select=66, step=5, verbose=1)

rfe_gb.fit(X_train, y_train)
gb_mask = rfe_gb.support_
```

RF처럼 GB는 feature importance 값을 계사하는 앙상블 방법

Combining the feature selectors

```
import numpy as np

votes = np.sum([lcv_mask, rf_mask, gb_mask], axis=0)

print(votes)
```

```
array([3, 2, 2, ..., 3, 0, 1])
```

```
mask = votes >= 2
reduced_X = X.loc[:, mask]
```

정보가 손실되지 않도록하려면 최소한 한번의 투표로 모든 feature을 선택할 수 있는데,
여기서는 featured을 유지하기 위해 적어도 두 개의 model이 feature에 투표하도록 선택함.

Let's practice!

DIMENSIONALITY REDUCTION IN PYTHON