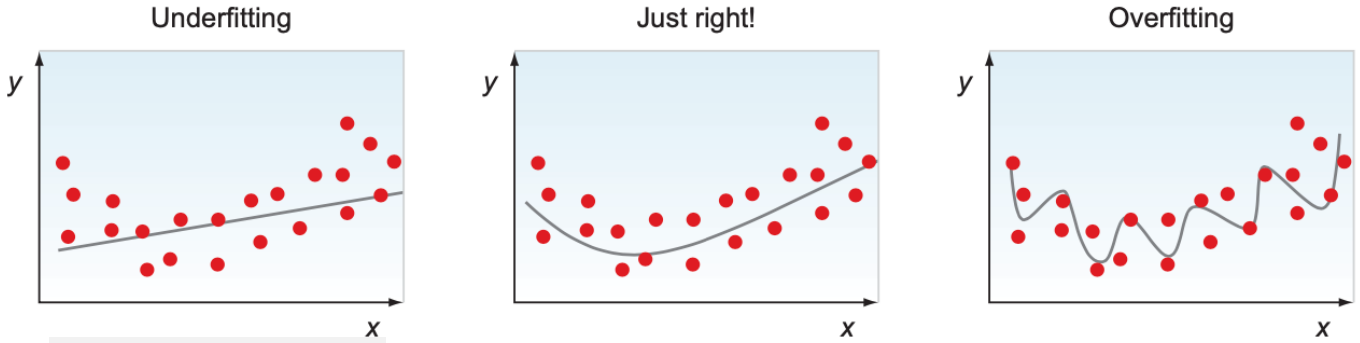
로지스틱 회귀

Multiple Linear Regression

로지스틱 회귀 실험

- **로지스틱 회귀 하이퍼파라미터 튜닝에 따른 성능 확인하기**
  - 규제화(Regularization) 값 변경에 따른 성능 실험
    - 규제화 사용의 목적 : 과대적합(overfitting) 을 막기 위함



과소적합 : 편향이 큰 경우

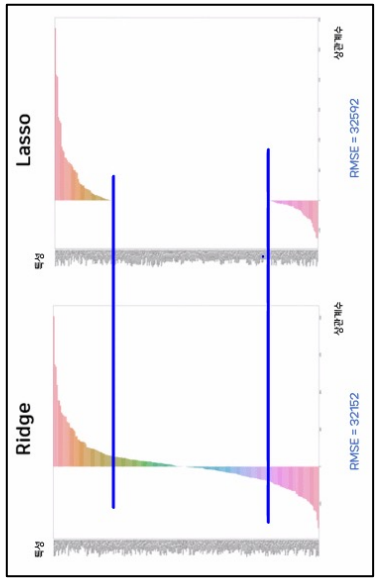
과대적합 : 분산이 큰 경우

로지스틱 회귀 실험

- **로지스틱 회귀 하이퍼파라미터 튜닝에 따른 성능 확인하기**
  - 규제항(Penalty term)을 포함한 손실함수
    - 규제항을 추가하여 모델 훈련 과정에서 가중치(W)를 줄이는 역할
    - 종류: L1, L2, L1+L2 (elastic net)

  - L1 규제 (Lasso)
    - 비용함수에 L1 norm을 추가하여 최적화
    - 일부 가중치를 완전히 0으로 축소하고 제거하는 특성이 있음
    - 특성 선택(feature selection)의 역할을 함
    - 특성의 수가 많은 데이터셋에 유용

  - L2 규제 (Ridge)
    - 비용함수에 L2 norm을 추가하여 최적화
    - 가중치를 0에 가깝게 만들지만 0으로 만들지는 않음
    - 모든 특성을 포함하고 가중치를 조정
    - 특성 간의 상관관계가 높은 경우에 유용



로지스틱 회귀 실험

- **로지스틱 회귀 하이퍼파라미터 튜닝에 따른 성능 확인하기**
  - 규제항(Penalty term)을 포함한 손실함수
    - 규제항을 추가하여 모델 훈련 과정에서 가중치(W)를 줄이는 역할

$$J(w) = \sum_{i=1}^{n} [-y^{(i)} \log\left(\emptyset\left(z^{(i)}\right)\right) - (1 - y^{(i)}) \log\left(1 - \emptyset\left(z^{(i)}\right)\right)] + \frac{\lambda}{2} ||w||^2$$

Logistic Regression Loss term

L2 Penalty term

로지스틱 회귀 실험

- ## 로지스틱 회귀 함수

class `sklearn.linear_model.`**`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

Parameters:

penalty : *{'l1', 'l2', 'elasticnet', None}, default='l2'*

Specify the norm of the penalty:

- `None`: no penalty is added;
- `'l2'`: add a L2 penalty term and it is the default choice;
- `'l1'`: add a L1 penalty term;
- `'elasticnet'`: both L1 and L2 penalty terms are added.

Warning: Some penalties may not work with some solvers. See the parameter `solver` below, to know the compatibility between the penalty and solver.

로지스틱 회귀 실험

- ## 로지스틱 회귀 함수

class `sklearn.linear_model.`**`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

solver : *{'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}, default='lbfgs'*

Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

- For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones;
- For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss;
- 'liblinear' is limited to one-versus-rest schemes.
- 'newton-cholesky' is a good choice for `n_samples` >> `n_features`, especially with one-hot encoded categorical features with rare categories. Note that it is limited to binary classification and the one-versus-rest reduction for multiclass classification. Be aware that the memory usage of this solver has a quadratic dependency on `n_features` because it explicitly computes the Hessian matrix.

Warning: The choice of the algorithm depends on the penalty chosen. Supported penalties by solver:

- 'lbfgs' - ['l2', None]
- 'liblinear' - ['l1', 'l2']
- 'newton-cg' - ['l2', None]
- 'newton-cholesky' - ['l2', None]
- 'sag' - ['l2', None]
- 'saga' - ['elasticnet', 'l1', 'l2', None]

로지스틱 회귀 실험

- **로지스틱 회귀 함수**
  - 비용함수 최적화를 위한 알고리즘 (solver)
    - 최적화알고리즘 : newton-cg, lbfgs, liblinear, sag, saga
      - 뉴튼 랩스(newton-Raphson) 계열 : newton-cg, lbfgs
      - 경사하강법 계열 : liblinear, sag, saga

  - 최적화 알고리즘별 설명
    - Newton-cg (Newton-Raphson) : 2차 도함수 계산이 필요하여 느림 (l2,none)
    - Lbfgs(Limited-memory Broyden-Fletcher-Goldfarb-Shnno) : 뉴튼랩슨을 약간 변형한 방법으로 2차 도함수를 근사 계산하여 속도 개선 (l2, none)
    - Liblinear : 경사하강법과 비슷하나 한 번에 여러개의 파라미터를 업데이트 하는 경사하강법과는 달리 한 번에 하나의 파라미터만 업데이트 함 (L1, L2)
    - Sag(Stochastic Average Gradient descent) : 경사하강법과 유사하나, 이전 업데이트 단계에서의 경사값을 현재 업데이트에 사용한다는 점에서 차이가 있고, 이런 특성으로 학습 속도가 더 빠름 (l2, none)
    - Saga : l1, l2, elasticnet, none 모두 지원

로지스틱 회귀 실습(1)

- 실습 코드: 링크
- 데이터셋: Iris 데이터
- 학습/시험 데이터: x_train/x_test
- 학습/시험 데이터 라벨: y_train/y_test ➔ (0,1,2)

- 데이터 로더, 범주형 데이터 변환, 데이터 분할
  - KNN 실습과 동일

- 로지스틱 회귀 API
  - LogisticRegression 클래스 호출 C=1/λ 로 규제화의 정도를 조절
    - C값이 클수록 규제화의 강도가 줄어듬

$$J(w) = \sum_{i=1}^{n}[-y^{(i)} \log\left(\emptyset\left(z^{(i)}\right)\right) - (1 - y^{(i)})\log\left(1 - \emptyset\left(z^{(i)}\right)\right)] + \frac{\lambda}{2}||w||^2$$

```
[5]  1 # Logistic regression
     2 from sklearn.linear_model import LogisticRegression
     3 Logit = LogisticRegression(C=200, random_state=11)  # C = 1/λ. 디폴트: L2, Auto.
```

로지스틱 회귀 실험

- 로지스틱 회귀 평가
  - 학습데이터 정밀도(accuracy), 시험데이터 정밀도, 정확도 행렬(confusion matrix)

```
[6]  1 # Accuracy score
     2 from sklearn.metrics import accuracy_score
     3 print(accuracy_score(y_train,y_train_pred))
     4 print(accuracy_score(y_test,y_test_pred))
     5
```

```
0.9809523809523809
1.0
```

```
1 # Confusion matrix
2 from sklearn.metrics import confusion_matrix
3 print(confusion_matrix(y_test, y_test_pred))  # Confusion matrix
```

```
[[15  0  0]
 [ 0 15  0]
 [ 0  0 15]]
```

로지스틱 회귀 실험

- 실습코드: 링크
- 데이터셋: 와인 데이터
- 학습/시험 데이터: x_train/x_test (13개의 특성변수)
- 학습/시험 데이터 라벨: y_train/y_test ➔ (1,2,3) 로 이미 범주형으로 저정되어 있음

- 다양한 규제강도에 따른 실험
  - L_1, L_2 규제화
  - $C=1/\lambda$ 로 규제화

- 다양한 규제강도에 따른 초정계수 실험
  - (관찰1) 규제강도가 클수록 추정된 계수들의 절대값이 작아짐
  - (관찰2) L1규제화의 경우 규제강도가 클수록 계수에 0이 많아짐. 즉, 계수에 대응하는 특성변수를 제거하는 역할을 담당함

로지스틱 회귀 실험

- 데이터 로더

```
[2]  1 # 데이터 불러오기. y값은 이미 범주형으로 되어있음.
     2 dat_wine=pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/'
     3                      'wine/wine.data',header=None)
     4 dat_wine.head()
     5 dat_wine.columns = ['class label', 'alchohol', 'malic acid', 'ash',
     6                     'alcalinity of ash', 'magnesium', 'total phenols',
     7                     'flavanoids', 'nonflavanoid phenols',
     8                     'proanthocyanins', 'color intensity', 'hue',
     9                     'OD208', 'proline']  # Column names
    10 print('class label:', np.unique(dat_wine['class label']))  # Class 출력
    11 dat_wine.head()
```

```
class label: [1 2 3]
```

- 학습/테스트 데이터 나누기

```
 1 # 전체 data를 training set과 test set으로 split
 2 from sklearn.model_selection import train_test_split
 3 X, y = dat_wine.iloc[:,1:].values, dat_wine.iloc[:,0].values
 4 X_train, X_test, y_train,y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

로지스틱 회귀 실험

- L1, L2 규제화에 따른 실험

```python
# Logistic Regression with L2 or L1 Regularization
from sklearn.linear_model import LogisticRegression
lr2_10 = LogisticRegression(penalty='l2', C=10.0, solver='saga')  # L2 with C(=1/λ)=10
lr2_1 = LogisticRegression(penalty='l2', C=1.0, solver='saga')    # L2 with C(=1/λ)=1
lr2_0_1 = LogisticRegression(penalty='l2', C=0.1, solver='saga')  # L2 with C(=1/λ)=0.1

lr1_10 = LogisticRegression(penalty='l1', C=10.0, solver='saga')  # L1 with C(=1/λ)=10
lr1_1 = LogisticRegression(penalty='l1', C=1.0, solver='saga')    # L1 with C(=1/λ)=1
lr1_0_1 = LogisticRegression(penalty='l1', C=0.1, solver='saga')  # L1 with C(=1/λ)=0.1
```

- L1, L2 규제화에 따른 정밀도

```python
# 규제화 방법(L2 or L1)과 규제강도(λ)를 바꿔가며 accuracy score 계산
lr2_10.fit(X_train_std, y_train)
print('Training accuracy with L2 and λ=0.1:', lr2_10.score(X_train_std, y_train))
print('Test accuracy with L2 and λ=0.1:', lr2_10.score(X_test_std, y_test))
```

```
Training accuracy with L2 and λ=0.1: 1.0
Test accuracy with L2 and λ=0.1: 0.9814814814814815
```

로지스틱 회귀 실험

- 규제화 강도에 따른 특성 변수 제거 관련

```
# L2 규제의 규제강도(C=1/λ)를 바꿔가며 계수 추정치 관찰
print(lr2_10.intercept_)
print(lr2_1.intercept_)
print(lr2_0_1.intercept_)

print(lr2_10.coef_)
print(lr2_1.coef_)
print(lr2_0_1.coef_)
```

```
[ 0.32296362  0.60033615 -0.92329977]
[ 0.28173282  0.6024029  -0.88413572]
[ 0.0683881   0.45688086 -0.52526895]
[[ 1.25664178  0.15899529  0.39926374 -1.53634982  0.08473759  0.37407767
   0.83876311 -0.28751864  0.09268499  0.12775152  0.0722625   0.97188895
   1.39116593]
 [-1.5372395  -0.43915291 -1.23984712  1.21218732 -0.32703465 -0.51670074
   0.85895303  0.40866438  0.39442514 -1.36535608  1.14060554  0.02219384
  -1.75612335]
 [ 0.28059772  0.28015763  0.84058338  0.32416249  0.24229707  0.14262307
  -1.69771614 -0.12114574 -0.48711013  1.23760457 -1.21286804 -0.99408279
   0.36495742]]
[[ 0.75495729  0.06165881  0.2336697  -0.8925231   0.02649841  0.29464787
   0.56064124 -0.2071806   0.13401678  0.12719203  0.10165733  0.61737663
   0.90976587]
 [-0.98657135 -0.32327905 -0.65176965  0.66792906 -0.22933211 -0.20753188
   0.43824097  0.19874428  0.24373934 -0.78043161  0.63697798  0.08558965
  -1.03461549]
 [ 0.23161406  0.26162025  0.41809995  0.22459404  0.2028337  -0.08711598
  -0.99888221  0.00843632 -0.37775612  0.65323959 -0.73863531 -0.70296628
   0.12484963]]
[[ 0.41030119 -0.03148562  0.13676704 -0.41134759  0.05383263  0.22360478
   0.31670971 -0.15968597  0.11370031  0.07036472  0.1110665   0.30981116
   0.51691919]
 [-0.5426495  -0.20155646 -0.25667025  0.28071006 -0.14835806 -0.0406011
   0.12453008  0.0829087   0.10087435 -0.44571802  0.27319166  0.09645505
  -0.51870207]
 [ 0.13234831  0.23304208  0.1199032   0.13063753  0.09452543 -0.18300368
  -0.44123979  0.07677727 -0.21457466  0.3753533  -0.38425816 -0.40626621
   0.00178288]]
```

```
# L1 규제의 규제강도(C=1/λ)를 바꿔가며 계수 추정치 관찰
print(lr1_10.intercept_)
print(lr1_1.intercept_)
print(lr1_0_1.intercept_)

print(lr1_10.coef_)
print(lr1_1.coef_)
print(lr1_0_1.coef_)
```

```
[ 0.39334184  0.5958625  -0.98920434]
[ 0.28462985  0.54413385 -0.8287637 ]
[ 0.05085504  0.30693808 -0.35779311]
[[ 1.19997383e+00  0.00000000e+00  1.54580407e-01 -1.57710818e+00
   0.00000000e+00  2.84110640e-01  8.25653391e-01  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  8.79401443e-01
   1.62309433e+00]
 [-1.78064452e+00 -4.56713816e-01 -1.47383462e+00  1.05802427e+00
  -2.92893382e-01 -4.06457188e-01  7.01037258e-01  4.65557053e-01
   2.08045179e-01 -1.56217122e+00  1.10248967e+00  0.00000000e+00
  -1.93798201e+00]
 [ 5.39392997e-02  1.17354860e-01  7.93799029e-01  0.00000000e+00
   1.77579266e-01  0.00000000e+00 -2.03734189e+00 -6.11569248e-06
  -3.41178760e-01  1.12646922e+00 -1.21297461e+00 -9.54584609e-01
   0.00000000e+00]]
[[ 0.0313239   0.          0.         -1.17721748  0.          0.
   0.02148411  0.          0.          0.          0.          0.62105
   0.97646011]
 [-1.58450657 -0.1446383  -0.77389205  0.04388529 -0.0731721   0.
   0.          0.12895574  0.         -0.98296559  0.23622794  0.
  -1.21587376]
 [ 0.          0.          0.          0.          0.          0.
  -2.08083573  0.         -0.04410569  0.27113817 -0.80443491 -0.65929035
   0.        ]]
[[ 0.          0.          0.         -0.04184623  0.          0.
   0.23299536  0.          0.          0.          0.          0.
   0.84042285]
 [-0.8348383   0.          0.          0.          0.          0.
   0.          0.          0.         -0.4233421   0.          0.
  -0.20651752]
 [ 0.          0.          0.          0.          0.          0.
  -0.60150089  0.          0.          0.10503944 -0.3520872  -0.521087
   0.        ]]
```