Methods 3: Multilevel Statistical Modeling and Machine Learning

Week 06: *Mid-way evaluation and Machine Learning Intro*: October 8, 2024

The course plan

Week 1: *Introduction*

Instructor sessions: *Setting up* R *and* Python *and recollection of the general linear model*

Week 2: *Multilevel linear regression*

Instructor sessions: *Modelling subject level effects – and how do they differ from group level effects?*

Week 3: *Link functions and fitting generalised linear multilevel models*

Instructor sessions: *What to do when the response variable is not continuous?*

Week 4: *Evaluating Generalised linear mixed models*

Instructor sessions: *How do we assess how models compare to one another?*

Week 5: *Explanation and Prediction*

Instructor sessions: *Code review*

Week 6: *Mid-way evaluation and Machine Learning Intro*

Instructor sessions: *Getting Python Running*

Week 7: *Linear regression revisited (machine learning)*

Instructor sessions: *How to constrain our models to make them more predictive*

Week 8: *Logistic regression revisited (machine learning)*

Instructor sessions: *Categorizing responses based on informed guesses*

Week 9: *Dimensionality Reduction, Principled Component Analysis (PCA)*

Instructor sessions: *What to do with very rich data?*

Week 10: *Outlook, unsupervised classification and neural networks*

Instructor sessions: *Data with no labels and networks*

Week 11: *Organising and preprocessing messy data*

Instructor sessions: *Code review*

Week 12: Final evaluation and wrap-up of course

Instructor sessions: *Ask anything!*

The four classical levels of variables

- Nominal
  - examples: true/false, correct/incorrect, female/male, dog/cat, apples/pear, also called *categorical*
  - they are ***names*** of categories, but it does not make sense to order them
- Ordinal
  - examples: senior/junior, adult/child 1/2/3
  - they are also ***names*** of categories, but there is an explicit or implicit ***ordering***, i.e. one is greater than another
- Interval
  - examples: the year 1984 AD; the temperature 100 °C
  - they ~~are~~ can be ***continuous***, and there is ***ordering***, e.g. 100 °C > 90 °C. And intervals can be compared, e.g. the interval from 80 °C to 100 °C is as long as the one from 40 °C to 60 °C
  - crucially, there is no real 0; the year before 1 AD is not characterised by absence of time; and 0 °C is not characterised by the absence of temperature This means that we cannot say that, say, 40 °C is twice as high a temperature as 20 °C
- Ratio
  - examples: the temperature 273 K, the reaction time of a subject
  - they ~~are~~ can be ***continuous***, there is ***ordering*** and there is a ***real 0***.
  - Thus we can say that 200 K is twice the temperature of 100 K, as 0 K *is* the absence of temperature; and we can say that subject 2, 400 ms, is twice as fast as subject 1, 200 ms, because 0 ms *is* the time when the event happened

SLIDE UPDATED 2024_10_03

Overview – pooling

- Complete pooling
  - Ignores the categorical predictor, e.g. *Subject*, *altogether*
  - *lm(Reaction ~ Days)*
- No pooling
  - Overfits the categorical predictor, e.g. *Subject*, i.e. overstates the variation among *Subjects*
  - *lm(Reaction ~ Days * Subject – 1); models slopes and intercepts for each subject*
  - *lm(Reaction ~ Days + Subject – 1); models intercepts for each subject*
- Partial pooling
  - A compromise between the two extremes above. If a group, e.g. Subject, has few observations (high variance), it will be shrunk towards the overall mean. If Subject has many observations (low variance), it will be shrunk less towards the overall mean
  - *lmer(Reaction ~ Days + (Days | Subject) # models slopes and intercepts for each subject*
  - *lmer(Reaction ~ Days + (1 | Subject) # models intercepts for each subject*

SLIDE UPDATED 2024_10_03

Recap?

- Bias can be added in ways that improve prediction
  - it does this by removing collinearity
  - thereby making the model stable
  - and more generalisable

Learning goals and outline

Mid-way evaluation and Machine Learning Intro

1) Learning some early *classification* methods
   - Perceptron and ADAline
   - Classification depends on having a quantiser function
2) Learning how linear *regression* (with biasing penalties) can be constructed and cross-validated

Mid-way evaluation

~10 min

1) Write something you liked about the course so far

2) Write something you did not like about the course so far

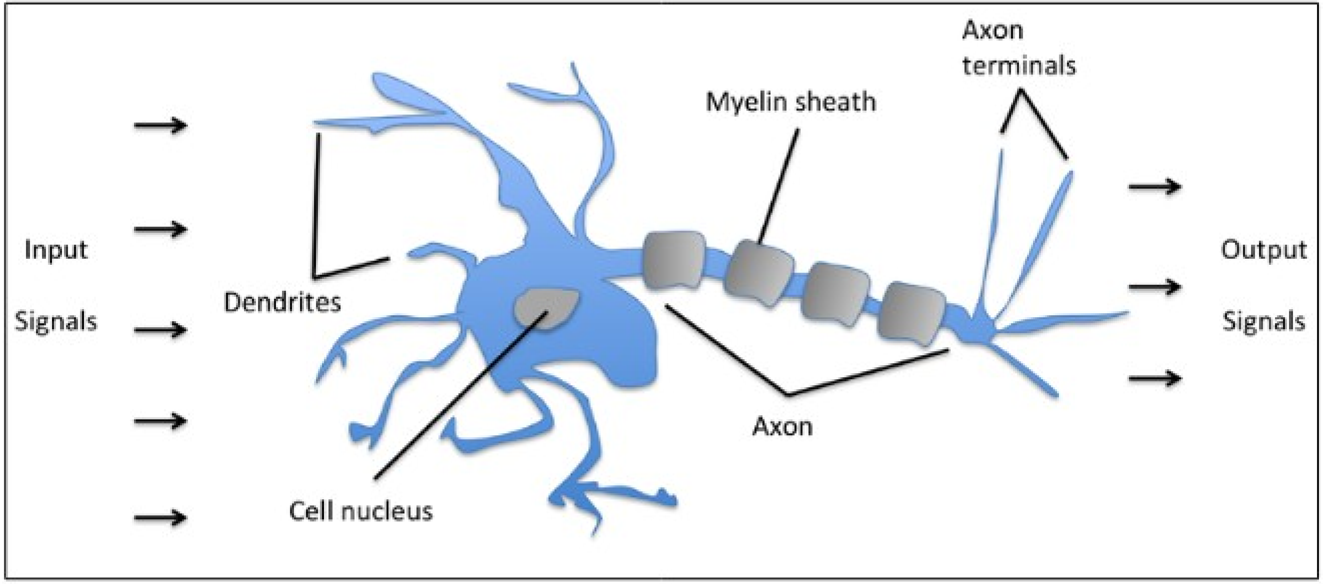
3) What would you change?

I’ll summarise the feedback on the three points, and what we’ll change for next time

The Perceptron

Raschka S (2015) Python Machine Learning. Packt Publishing Ltd

Black box idea



$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

$$z = w_1 x_1 + \ldots + w_m x_m$$

Question: what do *x*, *w* and *z* correspond to in the above picture of the *Perceptron*?

(p. 18: Raschka, 2015)

Prediction/classification rule

$$\phi(z) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$$

Perceptron fires

Perceptron doesn't fire

θ is a pre-specified threshold

(p. 18: Raschka, 2015)

Prediction/classification rule

$$w_0 = -\theta$$
$$x_0 = 1$$

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

$$z = -\theta + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

$$\phi(z) = \begin{cases} 1 \;\; if \; z \geq 0 \\ -1 \;\; otherwise \end{cases}$$

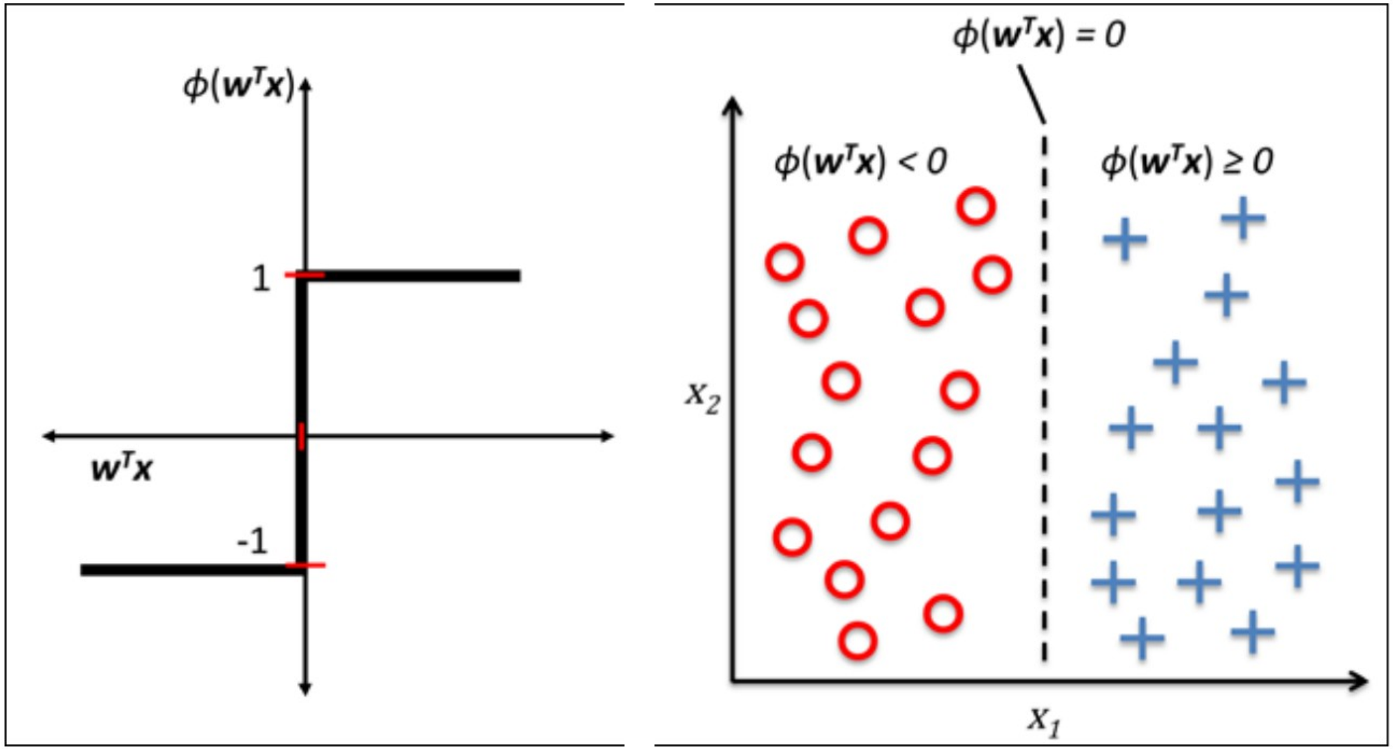
Perceptron fires

Perceptron doesn't fire

(p. 18: Raschka, 2015)

Perceptron classification

We want to find $\boldsymbol{w}^T\boldsymbol{x}$ that achieves this separation



(p. 21: Raschka, 2015)

In *Python* (in 2021)

```python
class Perceptron(object):
    """ Perceptron classifier

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.

    """
```

Special definition that indicates what the object (*Perceptron*) can be initialised with

```
def __init__(self, eta=0.01, n_iter=10):
    self.eta = eta
    self.n_iter = n_iter
```

```
ppn = Perceptron(eta=0.1, n_iter=10)
```

Specifying methods of *Perceptron*

1. Initialize the weights to 0 or small random numbers.
2. For each training sample $\boldsymbol{x}^{(i)}$ perform the following steps:
    1. Compute the output value $\hat{y}$.
    2. Update the weights.

```
def fit(self, X, y):
    """ Fit training data.

    Parameters
    ----------
    X : {array-like}, shape = [n_samples, n_features]
        Traing vectors, where n_samples
        is the number of samples and
        n_features is the number of features.
    y : array-like, shape = [n_samples]
        Target values.

    Returns
    -------
    self : object

    """
    self.w_ = np.zeros(1 + X.shape[1])
    self.errors_ = []

    for _ in range(self.n_iter):
        errors = 0
        for xi, target in zip(X, y):
            update = self.eta * (target - self.predict(xi))
            self.w_[1:] += update * xi
            self.w_[0] += update
            errors += int(update != 0.0)
        self.errors_.append(errors)
    return self
```

Compute the output value $\hat{y}$

```python
def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) + self.w_[0]

def predict(self, X):
    """Return class label after unit step"""
    return np.where(self.net_input(X) >= 0.0, 1, -1)
```

$$\phi(z) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$$

When we are right

real label

$$\Delta w_j = \eta\left(\boxed{-1} - \boxed{-1}\right)x_j^{(i)} = 0$$

predicted label

```
update = self.eta * (target - self.predict(xi))
self.w_[1:] += update * xi
self.w_[0] += update
```

real label

$$\Delta w_j = \eta\left(\boxed{1} - \boxed{1}\right)x_j^{(i)} = 0$$

predicted label

Δw_j: change in weight

η: learning rate

When we are wrong

real label

$$\Delta w_j = \eta(\boxed{1} - \boxed{-1})x_j^{(i)} = \eta(2)x_j^{(i)}$$

predicted label

real label

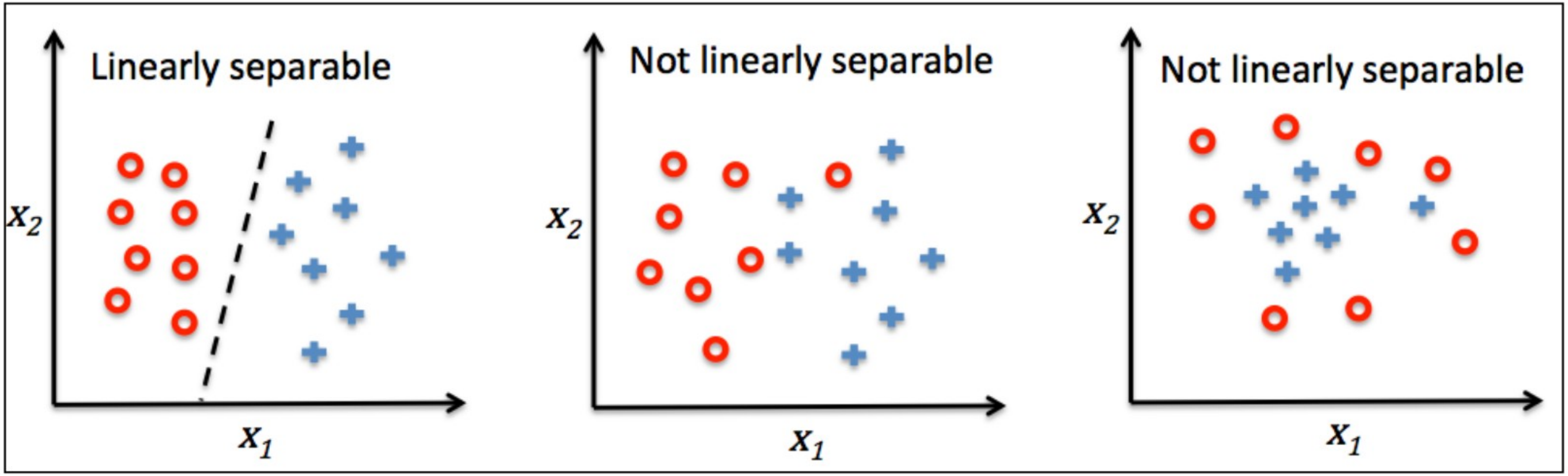
$$\Delta w_j = \eta(\boxed{-1} - \boxed{1})x_j^{(i)} = \eta(-2)x_j^{(i)}$$

predicted label

Δw_j: change in weight

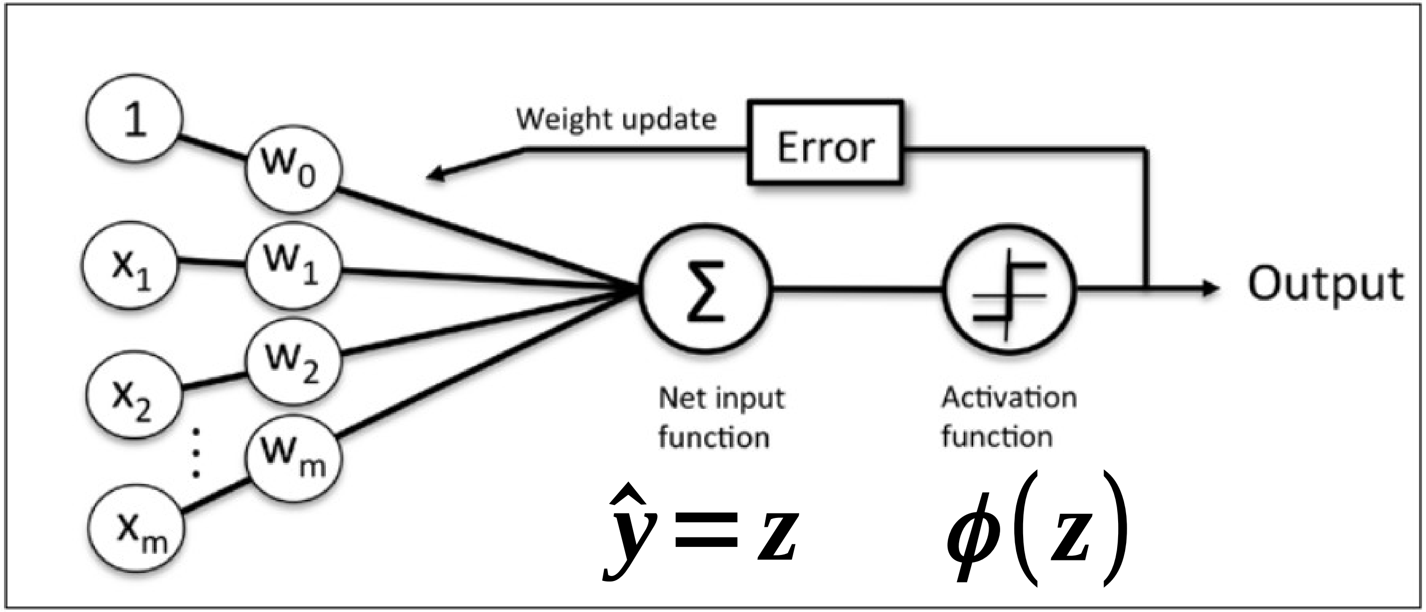
η: learning rate

Convergence only possible when linearly separable



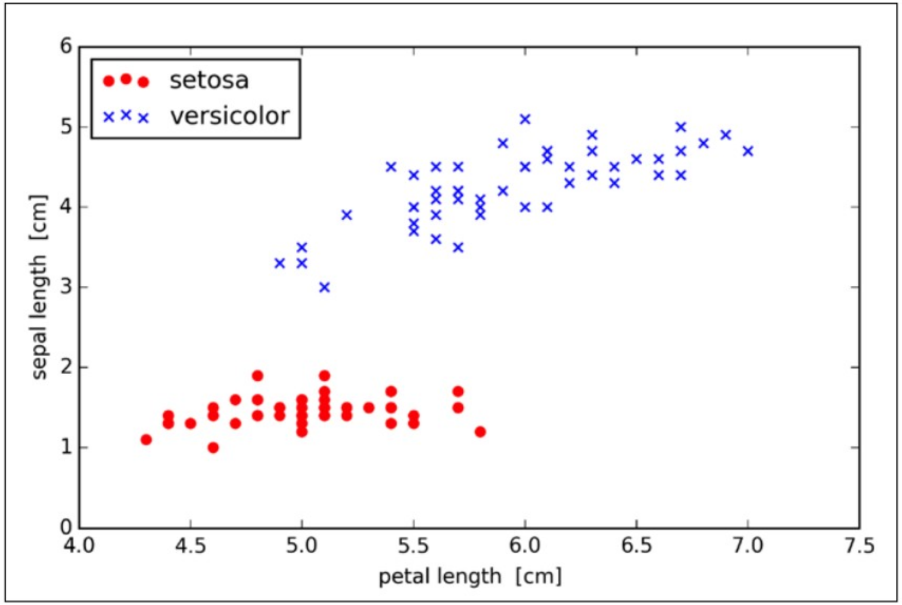
(p. 23: Raschka, 2015)

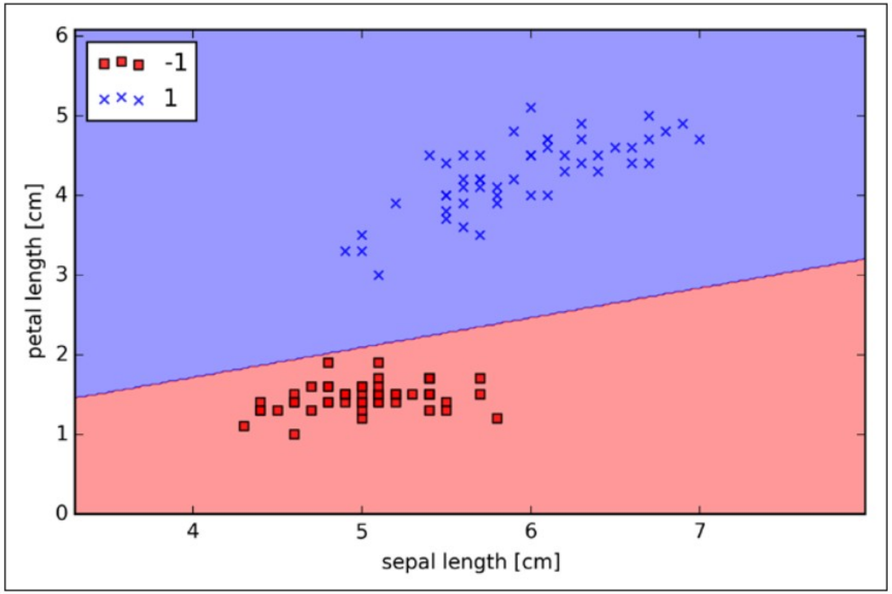
Perceptron: Graphical summary



(p. 24: Raschka, 2015)

An example






```
In [153]: ppn.w_
Out[153]: array([-0.04 , -0.068,  0.182])
```

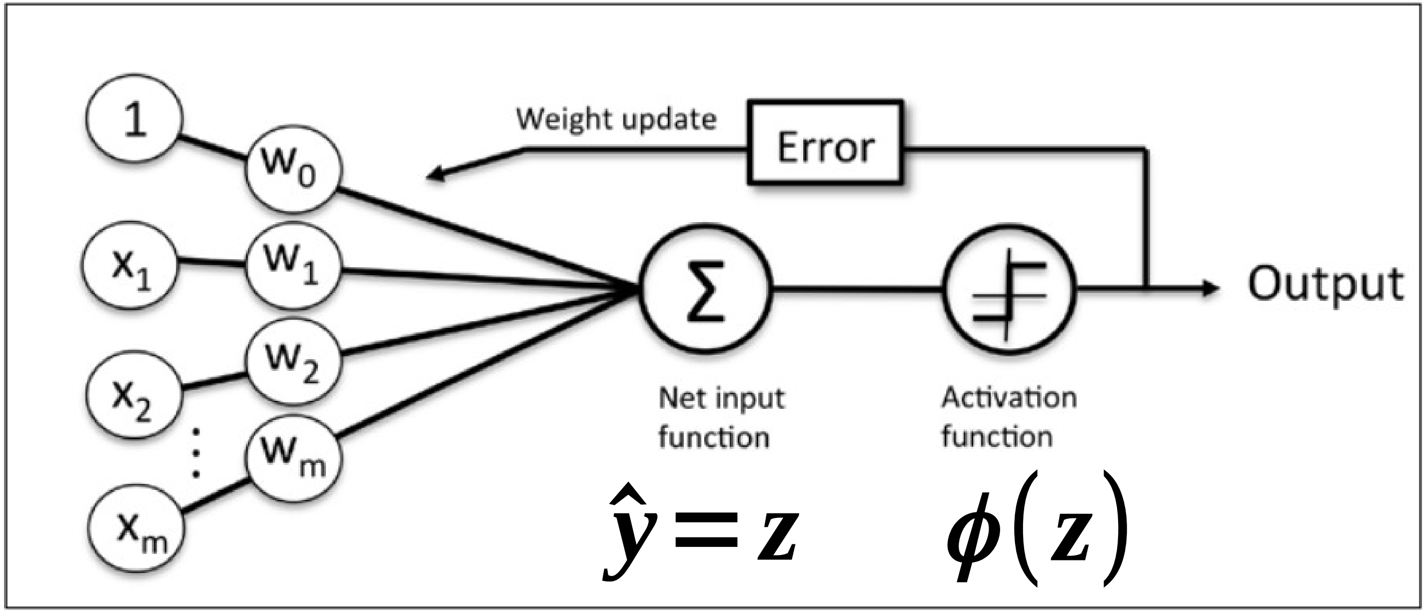
```
In [155]: X_subset[0, :]
Out[155]: array([5.1, 1.4])
```

$$\hat{y}_1 = -0.04 - 0.068 \cdot 5.1 + 0.182 \cdot 1.4 = -0.132$$

```
In [156]: ppn.net_input(X_subset[0, :])
Out[156]: -0.13199999999999953
```

(p. 29 & p. 32: Raschka, 2015)

Perceptron: Graphical summary

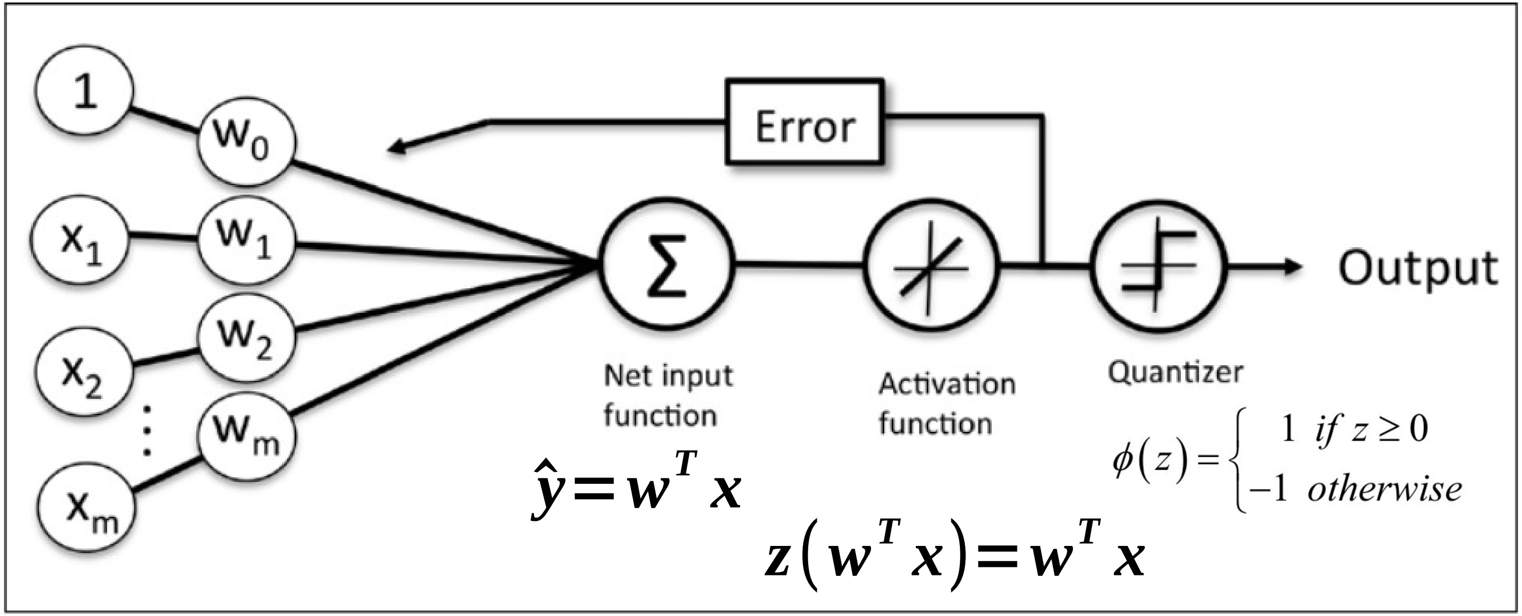


(p. 24: Raschka, 2015)

ADALINE

Raschka S (2015) Python Machine Learning. Packt Publishing Ltd

ADAptive LInear NEuron (ADALINE)



$$\hat{\boldsymbol{y}} = \boldsymbol{w}^T \boldsymbol{x}$$

$$\boldsymbol{z}(\boldsymbol{w}^T \boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$

$$\boldsymbol{w}^T \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \ldots + w_{m-1} x_{m-1} + w_m x_m$$

 (p. 33: Raschka, 2015)

ADALINE Gradient descent

```python
def __init__(self, eta=0.01, n_iter=50):
    self.eta = eta
    self.n_iter = n_iter
```

```python
class AdalineGD(object):
    """ ADAptive LInear NEuron classifier

    Parameters
    ------------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----------
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every epoch.

    """
```

Methods

$$\hat{y} = w^T x$$

$$z(w^T x) = w^T x$$

$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

```python
def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) + self.w_[0]

def activation(self, X):
    """Computer linear activation"""
    return self.net_input(X)

def predict(self, X):
    """Return class label after unit step"""
    return np.where(self.activation(X) >= 0.0, 1, -1)
```

The *fit* method

eta (η): learning rate (a constant)

output: $X\hat{\beta}=\hat{y}$

errors: $y-\hat{y}$

X.T.dot(errors): $X^T\cdot(y-\hat{y})$

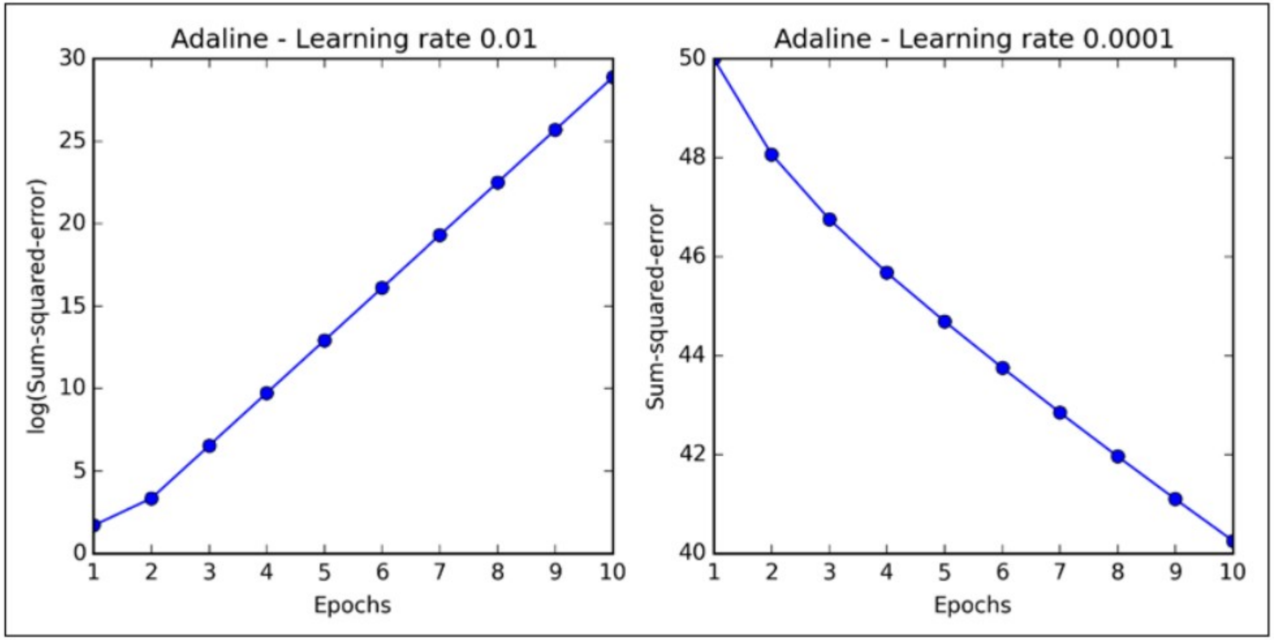
$$X^T\cdot(y-\hat{y})=\Delta w_1+\Delta w_2+\ldots+\Delta w_{m-1}+\Delta w_m$$

cost function: $\left(\sum(y-\hat{y})^2\right)/2$

```python
def fit(self, X, y):
    """ Fit training data.

    Parameters
    ----------
    X : {array-like}, shape = [n_samples, n_features]
        Traing vectors, where n_samples
        is the number of samples and
        n_features is the number of features.
    y : array-like, shape = [n_samples]
        Target values.

    Returns
    -------
    self : object

    """
    self.w_ = np.zeros(1 + X.shape[1])
    self.cost_ = []

    for i in range(self.n_iter):
        output = self.net_input(X)
        errors = (y - output)
        self.w_[1:] += self.eta * X.T.dot(errors)
        self.w_[0] += self.eta * errors.sum()
        cost = (errors**2).sum() / 2.0
        self.cost_.append(cost)
    return self
```

Learning rate

Overshooting the global minimum

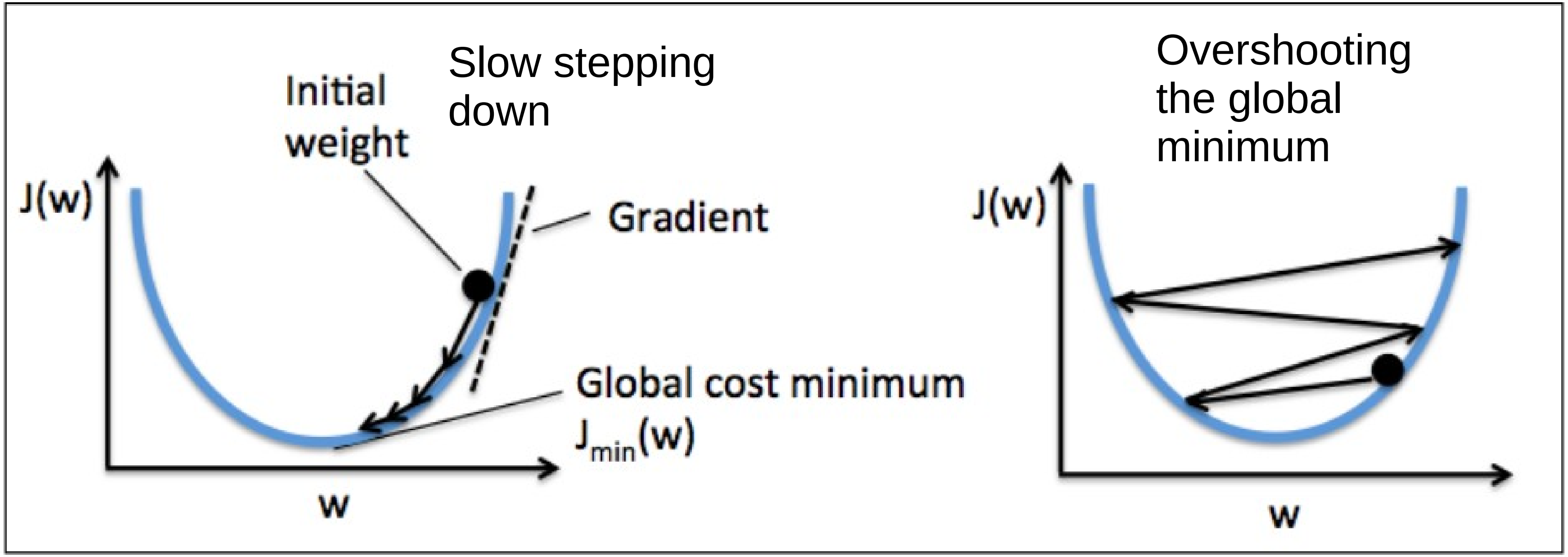


Slow stepping down

Always check whether it converges

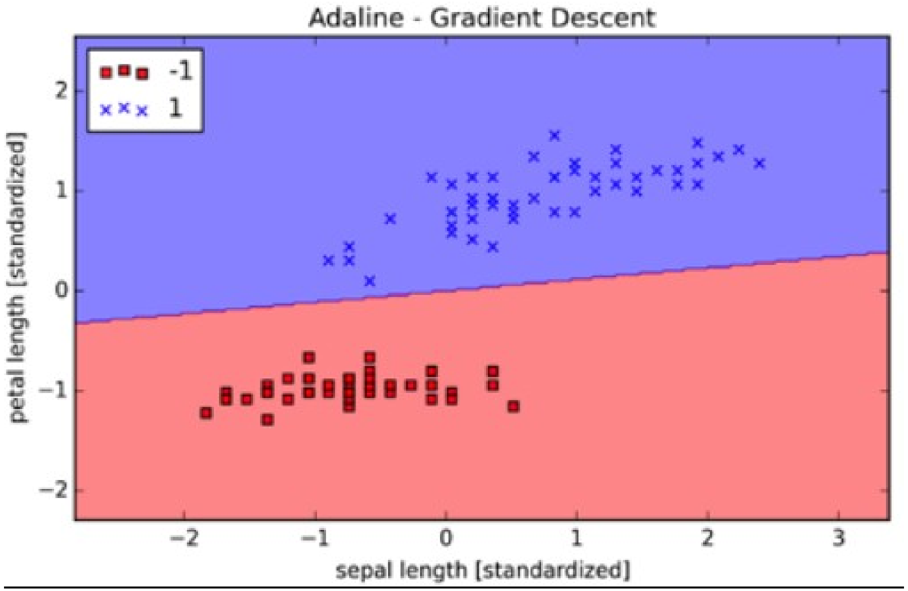
(p. 40: Raschka, 2015)

Gradient descent



$$J(w) = (\sum (y - \hat{y})^2)/2$$

(p. 40: Raschka, 2015)



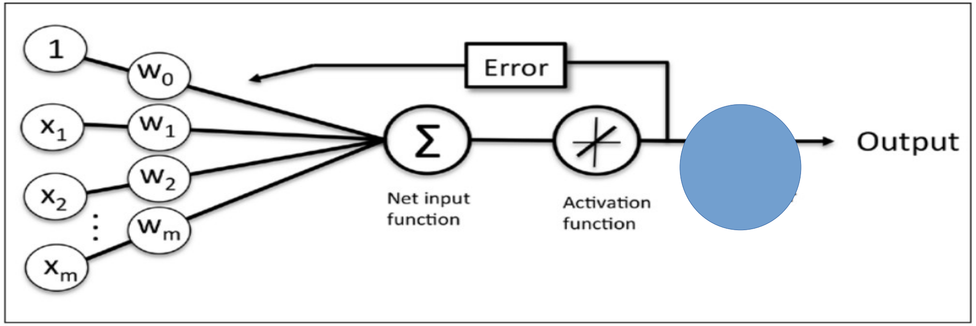
(p. 42: Raschka, 2015)

```
In [163]: ada.w_
Out[163]: array([-0.40808285, -0.33924452,  0.79202224])
```

```
In [155]: X_subset[0, :]
Out[155]: array([5.1, 1.4])
```

$$\hat{y}_1 \approx -0.408 - 0.339 \cdot 5.1 + 0.792 \cdot 1.4 = -1.03$$

```
In [164]: ada.net_input(X_subset[0, :])
Out[164]: -1.029398778794785
```



How does this relate to linear regression?

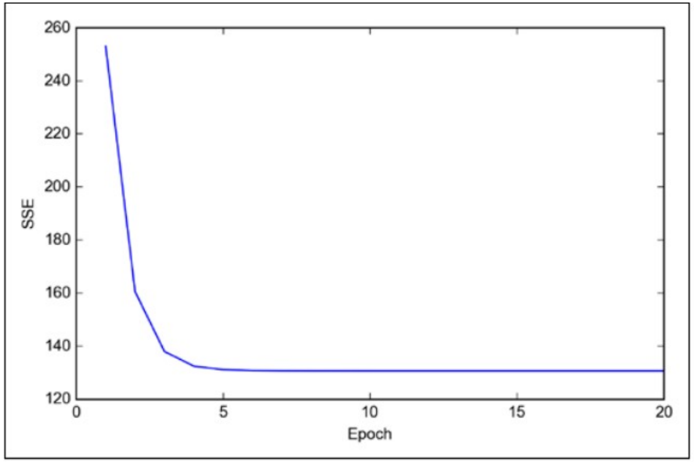
From ADALINE

$$\boldsymbol{w}^T \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \ldots + w_{m-1} x_{m-1} + w_m x_m$$

Very similar to ADALINE
and when converged will be virtually identical to the ordinary least squares solution

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Convergence




```
class LinearRegressionGD(object):

    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return self.net_input(X)
```

ADALINE

```python
def fit(self, X, y):
    """ Fit training data.

    Parameters
    ----------
    X : {array-like}, shape = [n_samples, n_features]
        Traing vectors, where n_samples
        is the number of samples and
        n_features is the number of features.
    y : array-like, shape = [n_samples]
        Target values.

    Returns
    -------
    self : object

    """
    self.w_ = np.zeros(1 + X.shape[1])
    self.cost_ = []

    for i in range(self.n_iter):
        output = self.net_input(X)
        errors = (y - output)
        self.w_[1:] += self.eta * X.T.dot(errors)
        self.w_[0] += self.eta * errors.sum()
        cost = (errors**2).sum() / 2.0
        self.cost_.append(cost)
    return self
```

```python
def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) + self.w_[0]

def activation(self, X):
    """Computer linear activation"""
    return self.net_input(X)

def predict(self, X):
    """Return class label after unit step"""
    return np.where(self.activation(X) >= 0.0, 1, -1)
```

```python
class LinearRegressionGD(object):

    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.n_iter):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):
        return self.net_input(X)
```

LINEAR REGRESSION

Olivier Grisel
@ogrisel

In scikit-learn 1.0, we decided to deprecate the sklearn.datasets.load_boston function because the design of this dataset casually assumes that people prefer to buy housing in racially segregated neighborhoods.

(p. 229: Raschka, 2015)

The features of the 506 samples may be summarized as shown in the excerpt of the dataset description:

- **CRIM**: This is the per capita crime rate by town
- **ZN**: This is the proportion of residential land zoned for lots larger than 25,000 sq.ft.
- **INDUS**: This is the proportion of non-retail business acres per town
- **CHAS**: This is the Charles River dummy variable (this is equal to 1 if tract bounds river; 0 otherwise)
- **NOX**: This is the nitric oxides concentration (parts per 10 million)
- **RM**: This is the average number of rooms per dwelling
- **AGE**: This is the proportion of owner-occupied units built prior to 1940
- **DIS**: This is the weighted distances to five Boston employment centers
- **RAD**: This is the index of accessibility to radial highways
- **TAX**: This is the full-value property-tax rate per $10,000
- **PTRATIO**: This is the pupil-teacher ratio by town
- **B**: This is calculated as *1000(Bk - 0.63)^2*, where Bk is the proportion of people of African American descent by town
- **LSTAT**: This is the percentage lower status of the population
- **MEDV**: This is the median value of owner-occupied homes in $1000s

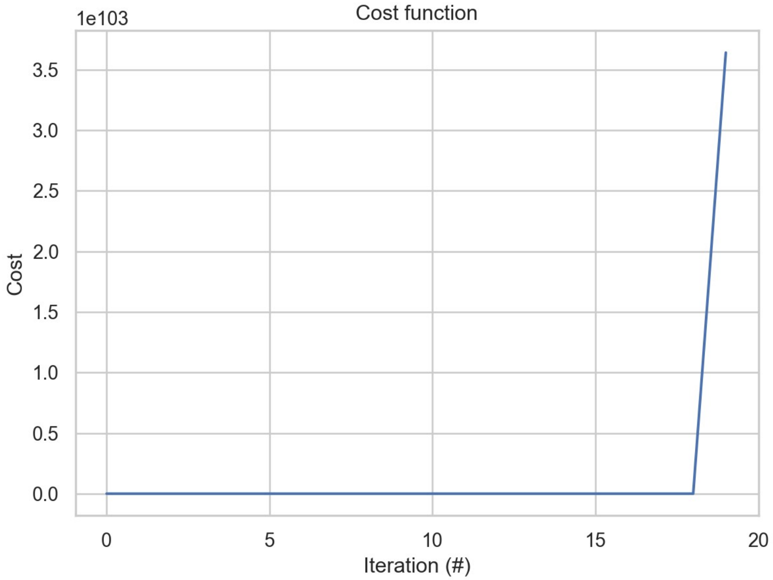
California Housing Dataset

X

MedInc: median income in block group
HouseAge: median house age in block group
AveRooms: average number of rooms per household
AveBedrms: average number of bedrooms per household
Population: block group population
AveOccup: average number of household members
Latitude: block group latitude
Longitude: block group longitude

y

MedHouseVal: The median house value for California districts, expressed in hundreds of thousands of dollars ($100,000)




```
LR = LinearRegressionGD()
LR.fit(X[:, 0:1], y) ## just fitting on Median Income
```

$$J(w) = \left(\sum (y - \hat{y})^2\right)/2$$

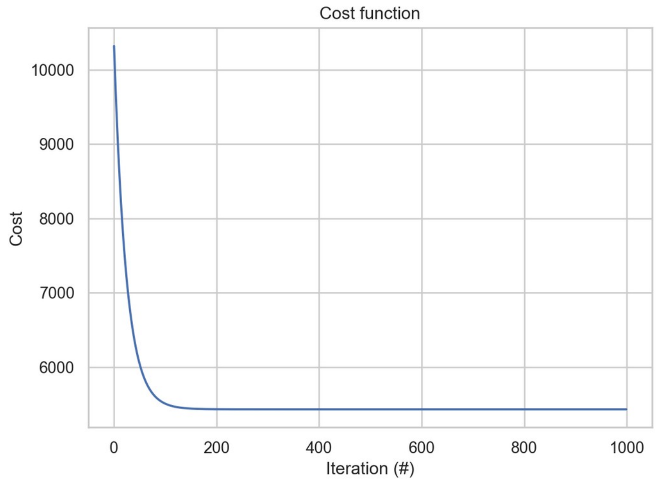
```
print(LR.w_)
```

```
## [-1.10873027e+51 -5.27207344e+51]
```

!

Check for convergence!

```python
X, y = fetch_california_housing(return_X_y=True)
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
sc_y = StandardScaler()
X_std = sc_x.fit_transform(X)
y_std = np.squeeze(sc_y.fit_transform(y.reshape(-1, 1)))

LR = LinearRegressionGD(eta=1e-6, n_iter=1000)
LR.fit(X_std[:, 0:1], y_std) ## just fitting on Median Income
```




```python
class LinearRegressionGD(object):

    def __init__(self, eta=0.001, n_iter=20):
        self.eta = eta
        self.n_iter = n_iter
```

- Standardisation

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

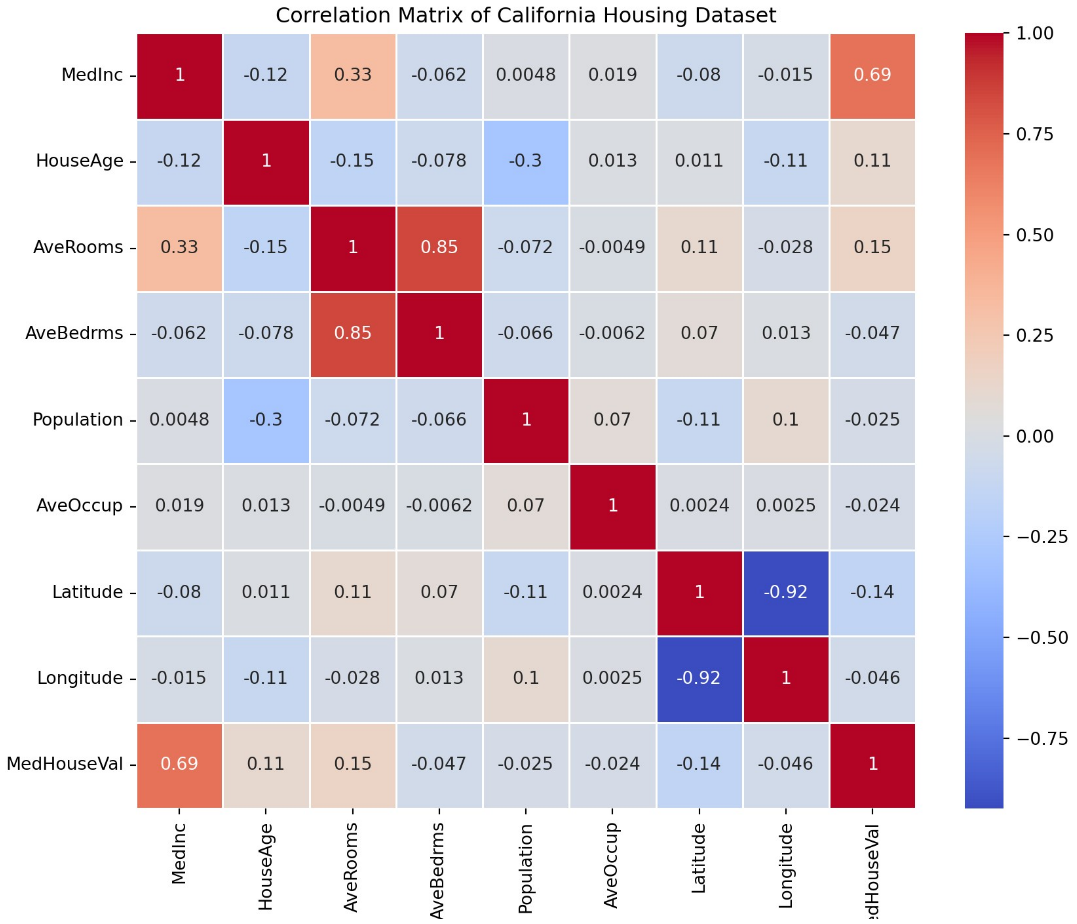
μ_x: sample mean for the feature: x

σ_x: sample standard deviation for the feature: x

Brings data onto a normal distribution with with μ=0 and σ=1

Multiple linear regression

$$\widehat{MedHouseVal} = \hat{w}_0 x_0 + \hat{w}_1 MedInc + \hat{w}_2 HouseAge$$
$$+ \hat{w}_3 AveRooms + \hat{w}_4 AveBedrms$$
$$+ \hat{w}_5 Population + \hat{w}_6 AveOccup + \hat{w}_7 Latitude + \hat{w}_8 Longitude + \epsilon$$


Correlation Matrix of California Housing Dataset
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
MedHouseVal
1 -0.12 0.33 -0.062 0.0048 0.019 -0.08 -0.015 0.69
-0.12 1 -0.15 -0.078 -0.3 0.013 0.011 -0.11 0.11
0.33 -0.15 1 0.85 -0.072 -0.0049 0.11 -0.028 0.15
-0.062 -0.078 0.85 1 -0.066 -0.0062 0.07 0.013 -0.047
0.0048 -0.3 -0.072 -0.066 1 0.07 -0.11 0.1 -0.025
0.019 0.013 -0.0049 -0.0062 0.07 1 0.0024 0.0025 -0.024
-0.08 0.011 0.11 0.07 -0.11 0.0024 1 -0.92 -0.14
-0.015 -0.11 -0.028 0.013 0.1 0.0025 -0.92 1 -0.046
0.69 0.11 0.15 -0.047 -0.025 -0.024 -0.14 -0.046 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

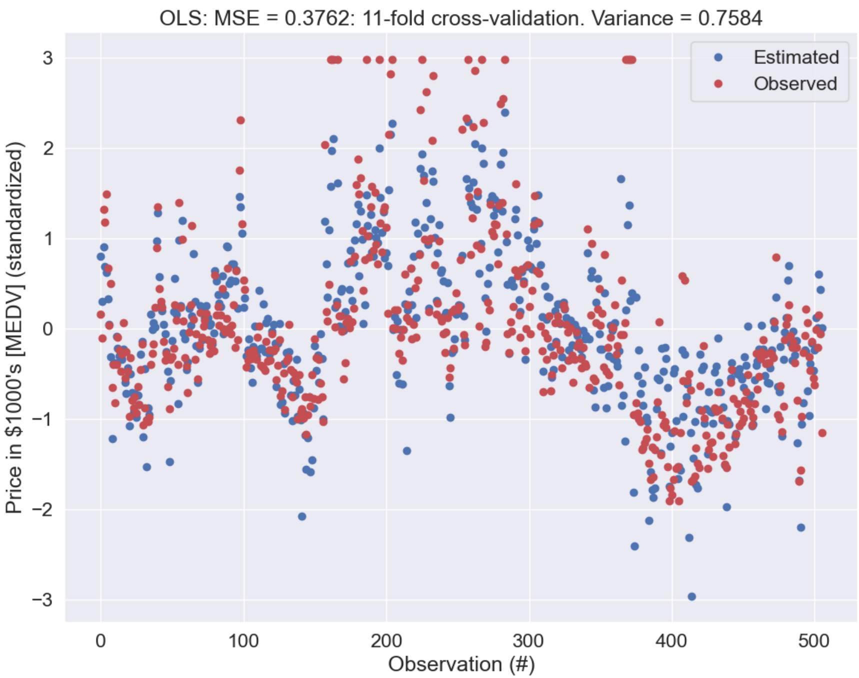
Because of the collinearity, we know we are prone to overfitting, so we do **out-of-sample** prediction instead of validating our model with traditional measures like R^2 and maximum likelihood

How to choose the **out-of-sample** dataset?

Cross-validation

(p. 176: Raschka, 2015)

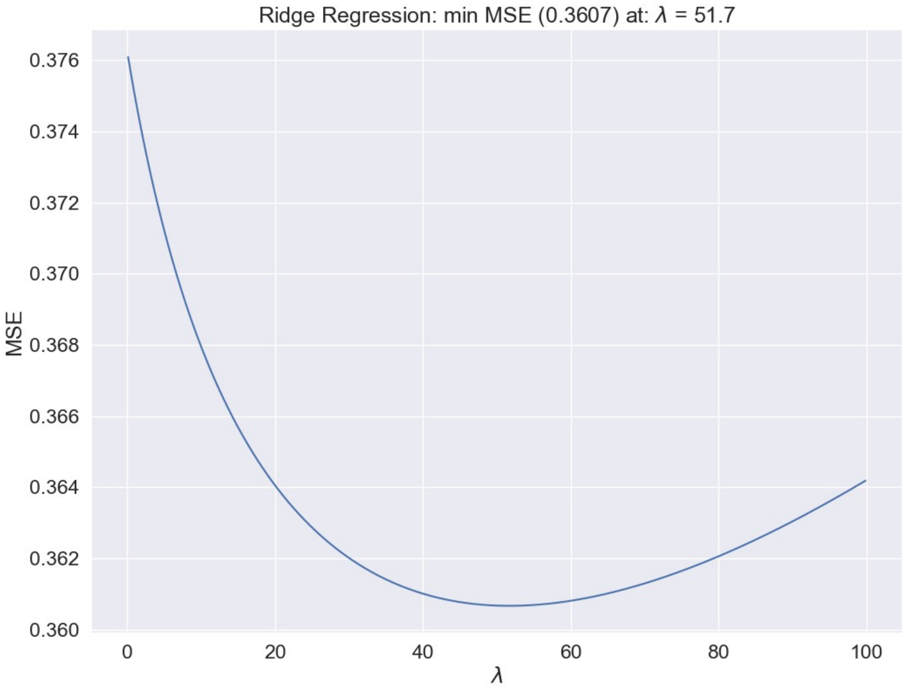
```
OLS = LinearRegression()
OLS.fit(X_std, y_std)

MSE = np.mean(cross_validate(OLS, X_std, y_std, k=11))
```

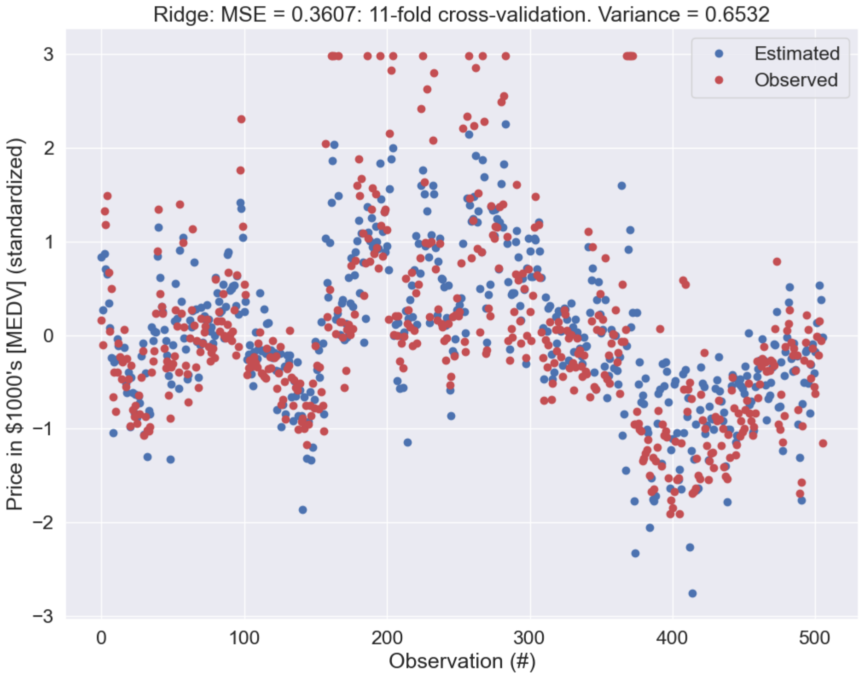


We can impose penalties

(but not on the intercept)

$$J(w)_{Ridge} = \sum^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \| w \|_2^2$$

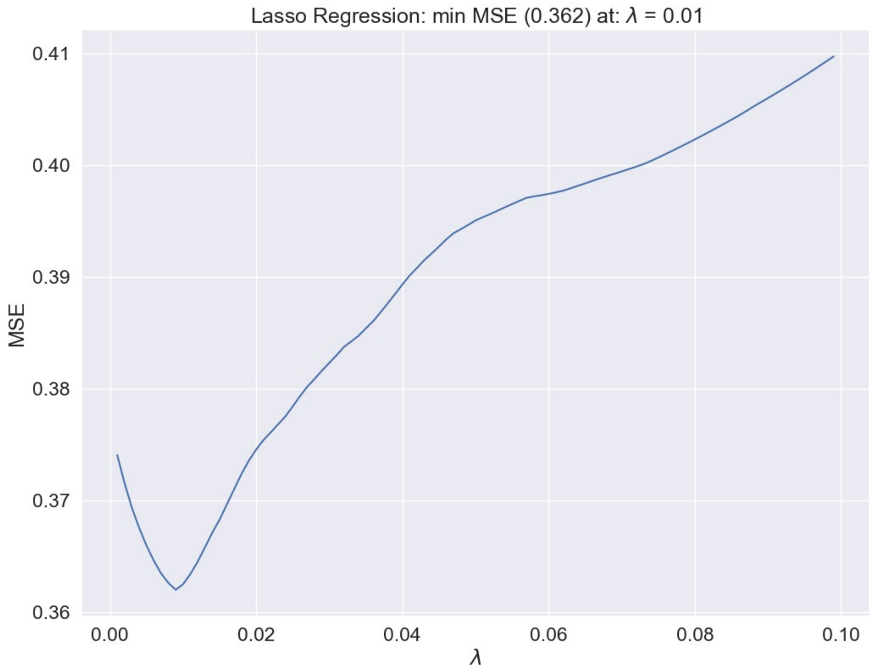


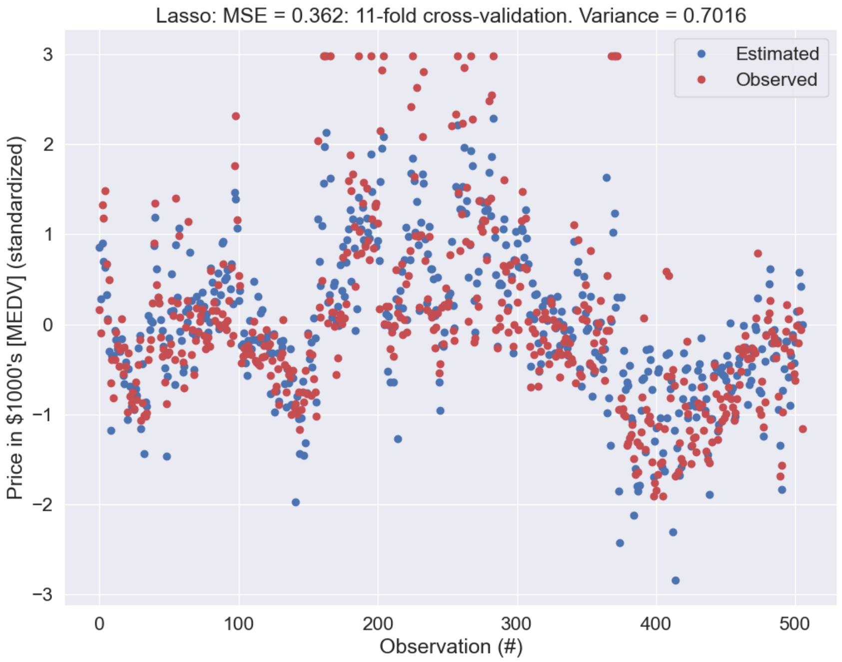


We can impose penalties

(but not on the intercept)

$$J(w)_{LASSO} = \sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \| w \|_1$$





Coefficients are shrunk

```
In [131]: OLS.coef_
Out[131]:
array([-0.09874812,  0.12473758,  0.02386168,  0.06945318, -0.22231612,
        0.2911837 ,  0.0325356 , -0.32907266,  0.34212986, -0.28361575,
       -0.21482076,  0.09763631, -0.43131412])

In [132]: RR.coef_
Out[132]:
array([-0.07536542,  0.08180161, -0.03182673,  0.07855868, -0.13185121,
        0.31082552,  0.00548938, -0.23491485,  0.11242408, -0.0959682 ,
       -0.18436614,  0.09154233, -0.36579723])

In [133]: lasso.coef_
Out[133]:
array([-0.07348948,  0.09107936, -0.        ,  0.07003181, -0.17110318,
        0.30950805,  0.        , -0.29010247,  0.16456993, -0.1319311 ,
       -0.19668957,  0.09063304, -0.41664587])
```

Summary

- We can build simple classification tools using scikit-learn
  - These can give us decision boundaries
  - That we can apply to new data (we haven’t done that yet)
- We need to define cost functions
  - These can be conceptually separated from prediction functions (remember ADALINE)
- We can use linear regression to do continuous predictions

The course plan

Week 1: *Introduction*

Instructor sessions: *Setting up* R *and* Python *and recollection of the general linear model*

Week 2: *Multilevel linear regression*

Instructor sessions: *Modelling subject level effects – and how do they differ from group level effects?*

Week 3: *Link functions and fitting generalised linear multilevel models*

Instructor sessions: *What to do when the response variable is not continuous?*

Week 4: *Evaluating Generalised linear mixed models*

Instructor sessions: *How do we assess how models compare to one another?*

Week 5: *Explanation and Prediction*

Instructor sessions: *Code review*

Week 6: *Mid-way evaluation and Machine Learning Intro*

Instructor sessions: *Getting Python Running*

Week 7: *Linear regression revisited (machine learning)*

Instructor sessions: *How to constrain our models to make them more predictive*

Week 8: *Logistic regression revisited (machine learning)*

Instructor sessions: *Categorizing responses based on informed guesses*

Week 9: *Dimensionality Reduction, Principled Component Analysis (PCA)*

Instructor sessions: *What to do with very rich data?*

Week 10: *Outlook, unsupervised classification and neural networks*

Instructor sessions: *Data with no labels and networks*

Week 11: *Organising and preprocessing messy data*

Instructor sessions: *Code review*

Week 12: Final evaluation and wrap-up of course

Instructor sessions: *Ask anything!*

Learning goals and outline

Mid-way evaluation and Machine Learning Intro

1) Learning some early *classification* methods
   - Perceptron and ADAline
   - Classification depends on having a quantiser function
2) Learning how linear *regression* (with biasing penalties) can be constructed and cross-validated

Next time

- We will do logistic regression and linear regression together
  - but we will skip an assignment and put in a code review instead
    - P7
      - ~~Assignment 3: How to constrain our models to make them more predictive~~
      - **Assignment 4: Using logistic regression to classify subjective experience from brain data**
    - P8
      - code review
- I’ll update the syllabus accordingly

Reading questions

- Chapter 10
  - What is lasso and ridge regression?
  - What is standardisation?
- Chapter 3
  - What is the cost function of logistic regression?
  - What is overfitting and underfitting?
  - What is a support vector?