Computer Science

CSC380: Principles of Data Science

Linear Models

Kyoungseok Jang

Announcements

- Withdrawal deadline: Mar. 28th
- HW5 deadline: Mar. 31st

Naïve Bayes Classifier: Class prior parameters

For the class prior distribution, take categorical distribution.

$$y \sim \text{Categorical}(\pi), \qquad \pi \in \mathbb{R}^C, \pi_c \geq 0, \sum_c \pi_c = 1$$

$\Rightarrow p(y = c) = \pi_c$

$\Rightarrow$ ***C-1*** *parameters for the 'class prior distribution'*

- I will fix this notation.

- I will emphasize this on our final exam review.

Outline

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

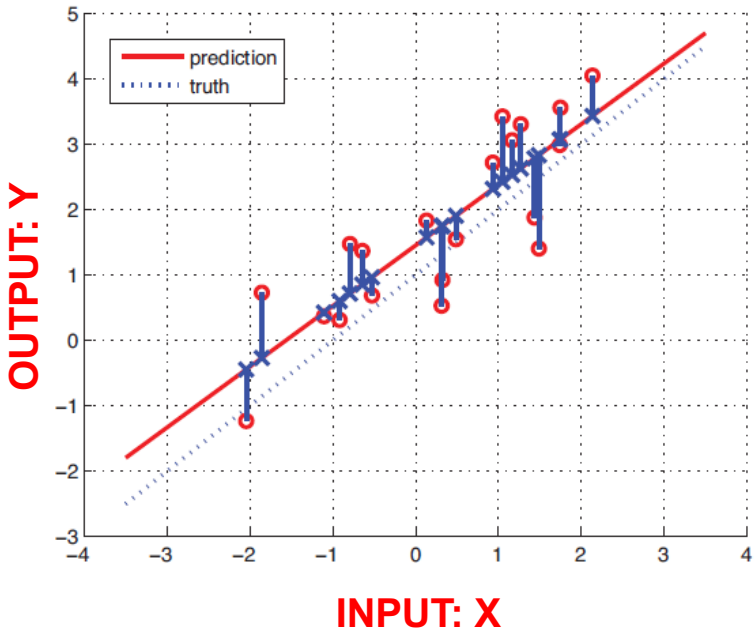
Outline

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

we will first focus on **what** is a linear function

then learn **how to train** a linear function from data

Linear Regression



Regression Learn a function that predicts outputs from inputs,

$$y = f(x)$$

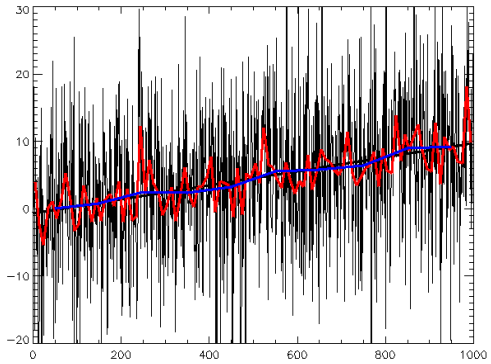
Outputs y are real-valued

Linear Regression As the name suggests, uses a *linear function*:

$$y = w^T x + b$$

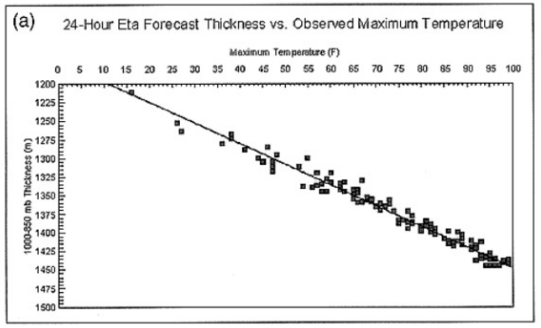
$$w^T x := \sum_{d=1}^{D} w_d x_d$$

When is linear regression useful?



Trendlines

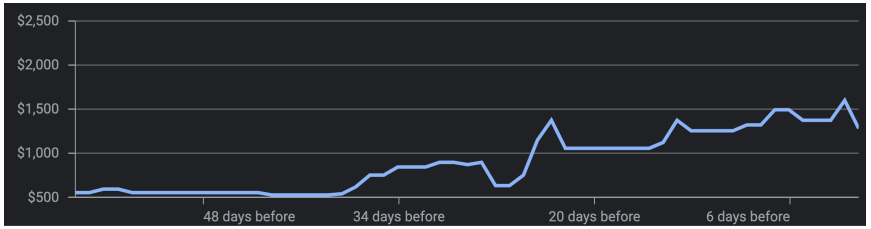
Stock Prediction



Climate Models

Massie and Rose (1997)

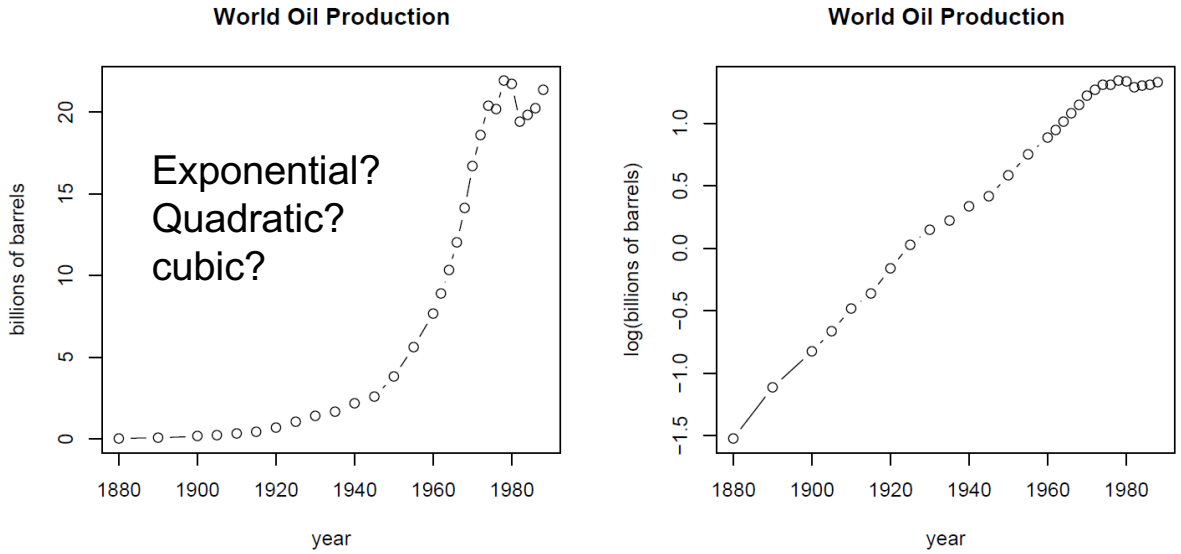
Price of an airline ticket:



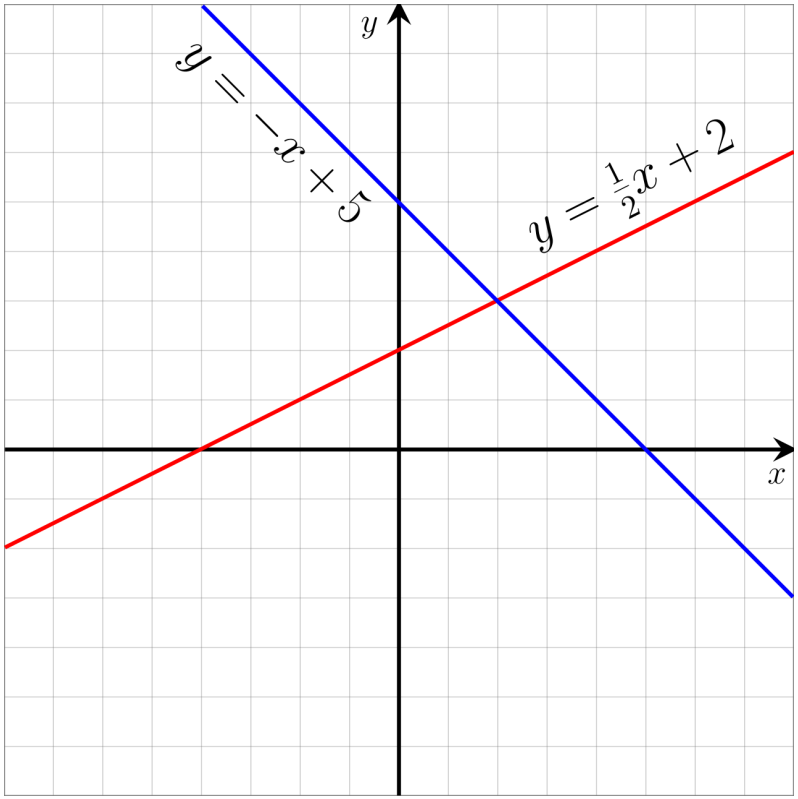
Used anywhere a linear relationship is assumed between inputs / (real-valued) outputs

Linear Regression

- Piazza question: Why should we express something by linear function?
  - Because it's easy to understand
  - Much more explicit than other nonlinear functions when visualize



Line Equation



Recall the equation for a line has a *slope* and an *intercept*,

$$y = w \cdot x + b$$

Slope **Intercept**

- Intercept (b) indicates where line crosses y-axis
- Slope controls angle of line
- Positive slope (w) → Line goes up left-to-right
- Negative slope → Line goes down left-to-right

Moving to higher dimensions…

- **1d regression**: regression with 1d input:

$$y = wx + b$$

- **D-dimensional regression**: input vector is $x \in \mathbb{R}^D$.

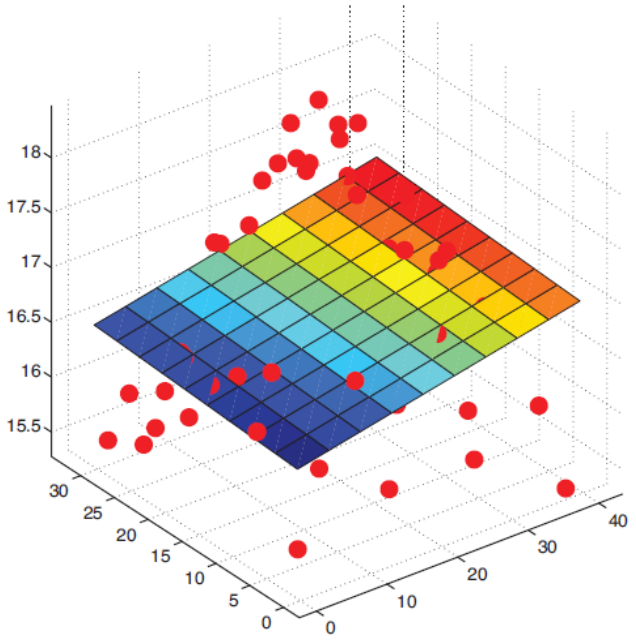
$$y = w_1 x_1 + w_2 x_2 + \cdots + w_D x_D + b$$

The definition of an *inner product*:

$$w^T x = w_1 x_1 + w_2 x_2 + \ldots + w_D x_D = \sum_{d=1}^{D} w_d x_d$$

The model is $y = w^T x + b$

$w = (w_1, \cdots, w_D), x = (x_1, \cdots, x_D)$



[Image: Murphy, K. (2012)]

Moving to higher dimensions…

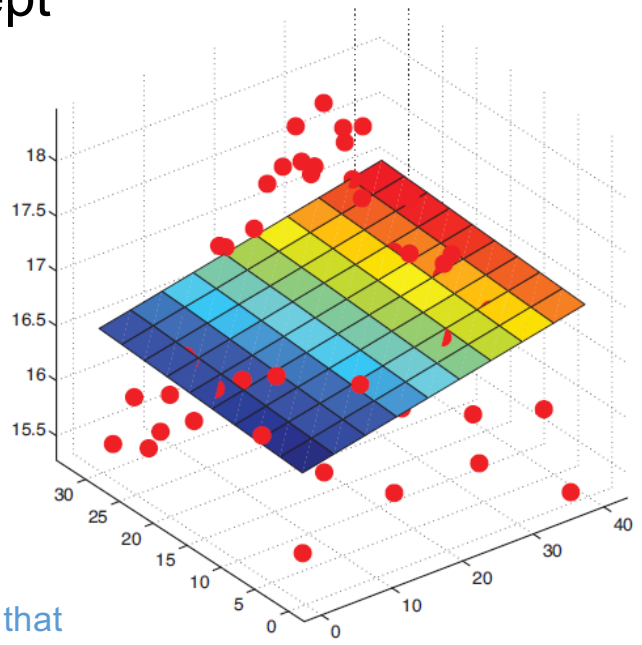
Often we simplify this by including the intercept into the weight vector,

$$\widetilde{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_D \\ b \end{pmatrix} \qquad \widetilde{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_D \\ 1 \end{pmatrix} \qquad y = \widetilde{w}^T \widetilde{x}$$

Since:

$$\widetilde{w}^T \widetilde{x} = \sum_{d=1}^{D} w_d x_d + b \cdot 1$$
$$= w^T x + b$$

from now on, we assume that $w \in \mathbb{R}^D$ and $x \in \mathbb{R}^D$ already has b and 1 in the last coordinate respectively.



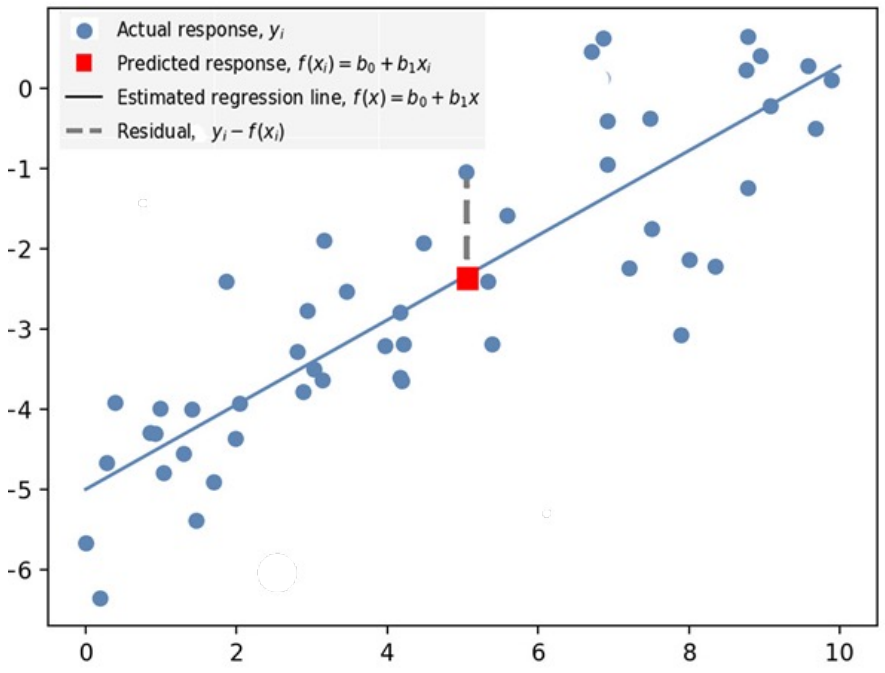
[Image: Murphy, K. (2012)]

There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

Fitting Linear Regression



Intuition Find a line that is as *close as possible* to every training data point

The distance from each point to the line is the **residual**

$$y - w^T x$$

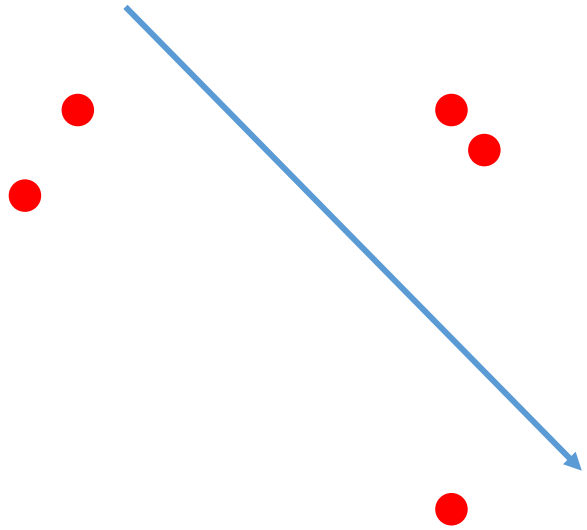
Training Output **Prediction**

Let's find w that will minimize the residual!

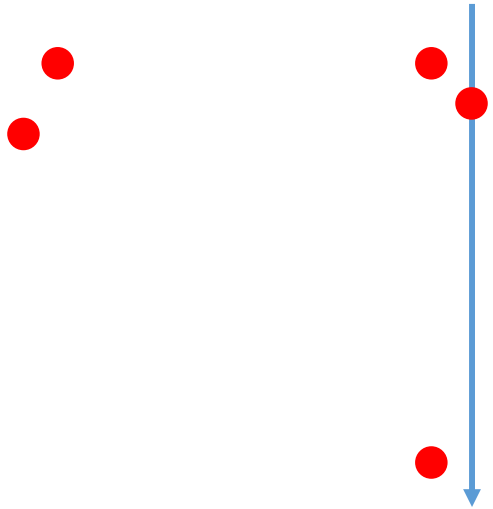
https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

Formal definition: what is the criterion?

- What is the definition of ‘closeness’ in this case?
  - There are multiple points….



Case 1

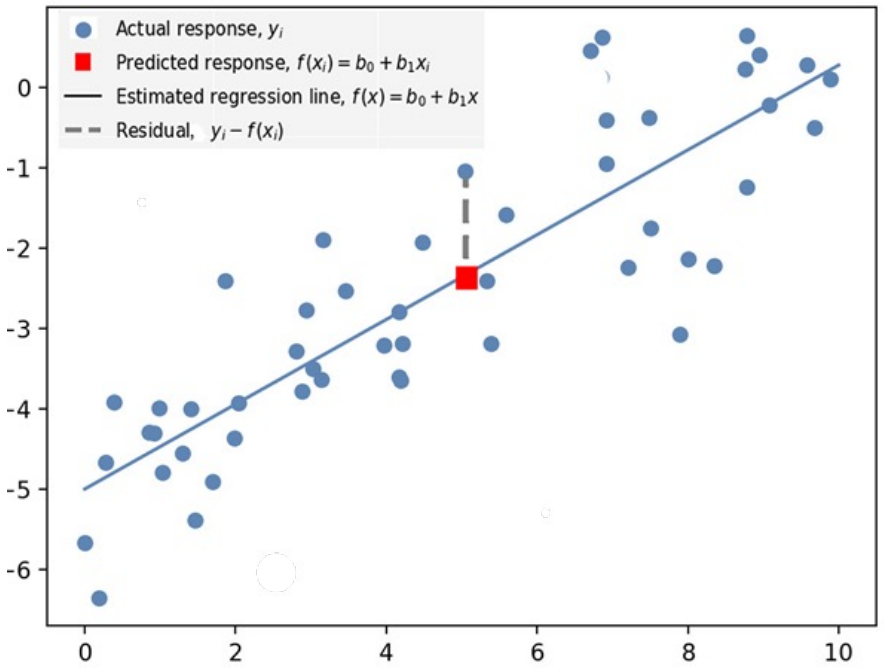


Case 2

Outline

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

Least Squares Solution



Functional Find a line that minimizes the sum of squared residuals!

Given: $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Compute:

$$w^* = \arg\min_w \sum_{i=1}^{m} \left(y^{(i)} - w^T x^{(i)}\right)^2$$

Least squares regression

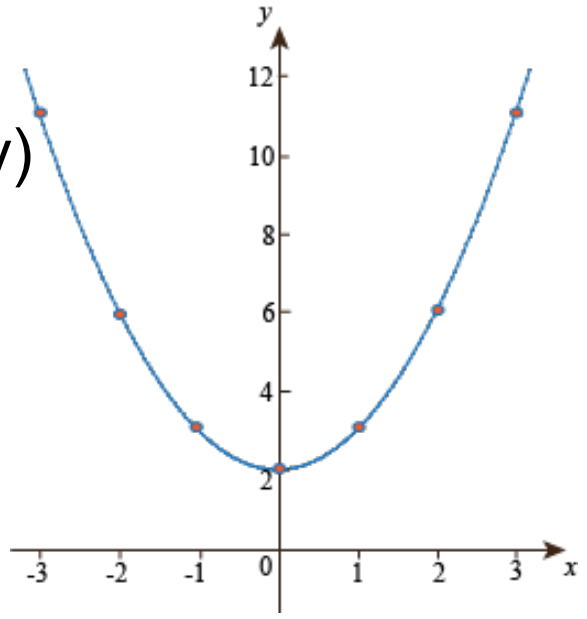
https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

Least Squares

$$\min_w \sum_{i=1}^{N} (y^{(i)} - w^T x^{(i)})^2$$

This is just a quadratic function…

- *Convex,* unique minimum (curves up only)
- Minimum given by zero-derivative
- Can find a closed-form solution



Let's see for scalar case with no bias,

$$y = wx$$

Least Squares : Simple Case

$$\frac{d}{dw}\sum_{i=1}^{N}(y^{(i)} - wx^{(i)})^2 =$$

Derivative (+ chain rule)

$$= \sum_{i=1}^{N} 2(y^{(i)} - wx^{(i)})(-x^{(i)}) = 0 \Rightarrow$$

Distributive Property (and multiply -1 both sides)

$$0 = \sum_{i=1}^{N} y^{(i)}x^{(i)} - w\sum_{j=1}^{N}(x^{(j)})^2$$

Algebra

$$w = \frac{\sum_i y^{(i)}x^{(i)}}{\sum_j (x^{(j)})^2}$$

Least Squares in Higher Dimensions

Things are a bit more complicated in higher dimensions and involve more linear algebra,

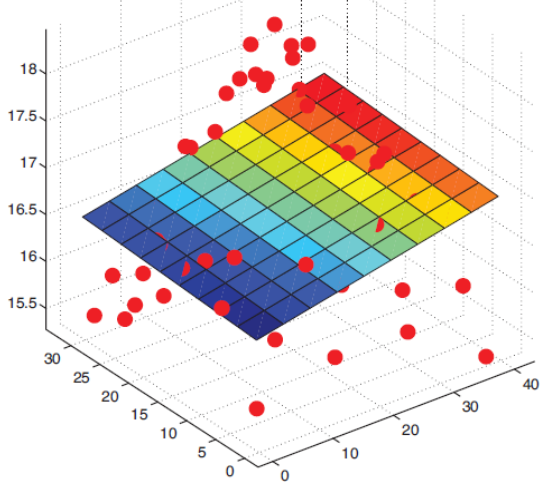
$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & \dots & x_D^{(1)} & 1 \\ x_1^{(2)} & \dots & x_D^{(2)} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(m)} & \dots & x_D^{(m)} & 1 \end{pmatrix}$$

Design Matrix
(each row is a data point)

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

Vector of labels

[Image: Murphy, K. (2012)]



Can write regression over *all training data* more compactly…

$$\mathbf{y} \approx \mathbf{X}w \quad \longleftarrow \text{mx1 Vector}$$

$$= \begin{pmatrix} (x^{(1)})^\top w \\ \dots \\ (x^{(m)})^\top w \end{pmatrix}$$

Least Squares in Higher Dimensions

Least squares can also be written more compactly,

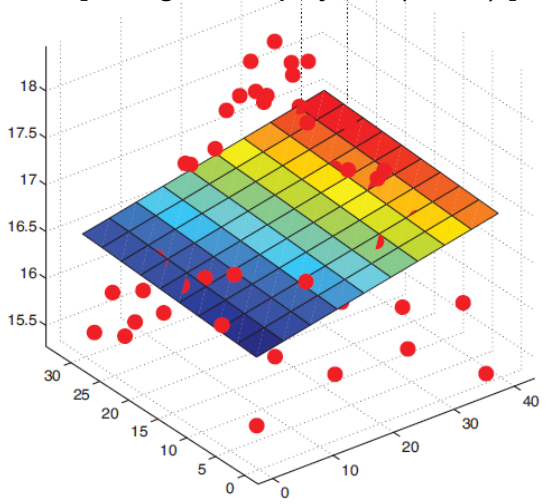
$$\min_w \sum_{i=1}^{N} (y^{(i)} - w^T x^{(i)})^2 = \|\mathbf{y} - \mathbf{X}w\|^2$$

Some slightly more advanced linear algebra gives us a solution,

$$w = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \left(\sum_j x^{(j)} \left(x^{(j)}\right)^\top\right)^{-1} \sum_i y^{(i)} x^{(i)}$$

Ordinary Least Squares *(OLS)* solution

[Image: Murphy, K. (2012)]



compare with the 1d version: $w = \frac{\sum_i y^{(i)} x^{(i)}}{\sum_j (x^{(j)})^2}$

Derivation a bit advanced for this class, but enough to know

- We know it has a closed-form and why
- We can evaluate it
- Generally know where it comes from.

Least Squares Implementation in Numpy


```
import numpy as np
import numpy.linalg as la

X_orig = np.array([[1,2], [3,4], [5,6], [7,8], [9,1]])
y = np.array([5,4,3,2,1])
m,D = X_orig.shape
X = np.hstack((X_orig, np.ones((m,1))))
```

✓ 0.2s

```
w_hat = la.inv(X.T@X) @ (X.T@y)
w_hat
```

✓ 0.2s

```
array([-0.5,  0. ,  5.5])
```

5 by 2 matrix

.shape returns the dimensions of the array

5 by 3 matrix

- .T for transpose
- la.inv() is the inversion operator
- @ operator for matrix-matrix, matrix-vector, or vector-vector product.

There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

Probabilistic Assumptions

- Assume $x \sim \mathcal{D}_X$ from some distribution. We then assume that

$$y = w^T x + \epsilon \quad \text{where } \epsilon \sim \mathcal{N}(0, \sigma^2)$$

- Equivalently,

$$p(y|x; w) = \mathcal{N}(w^T x, \sigma^2)$$

Why? Adding a constant to a Normal RV is still a Normal RV,

$$z \sim \mathcal{N}(m, P) \qquad z + c \sim \mathcal{N}(m + c, P)$$

for our case, linear regression $z \leftarrow \epsilon$ and $c \leftarrow w^T x$

MLE for Linear Regression

Given training data $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, maximize the likelihood!

$$\hat{w} = \arg\max_{w} \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; w)$$

$$= \arg\max_{w} \log \prod_{i=1}^{m} p(x^{(i)})\, p(y^{(i)}|x^{(i)}; w)$$

note $p(x^{(i)})$ does not depend on w!

$$= \arg\max_{w} \left[\sum_{i=1}^{m} \log p\left(y^{(i)}|x^{(i)}; w\right) + \sum_{i=1}^{m} \log p\left(x^{(i)}\right) \right]$$

subtracting a constant w.r.t. w does not affect the solution w!

$$= \arg\max_{w} \sum_{i=1}^{m} \log p\left(y^{(i)}|x^{(i)}; w\right)$$

note model assumption! $p(y|x; w) = \mathcal{N}(w^T x, \sigma^2)$

Univariate Gaussian (Normal) Distribution

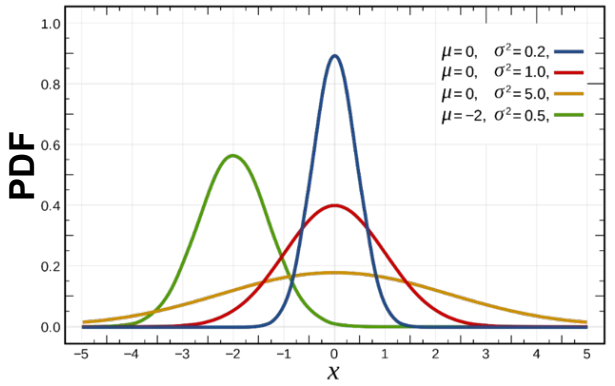
Let's focuson 1d case.
Let $\mu = w^T x$ for now.

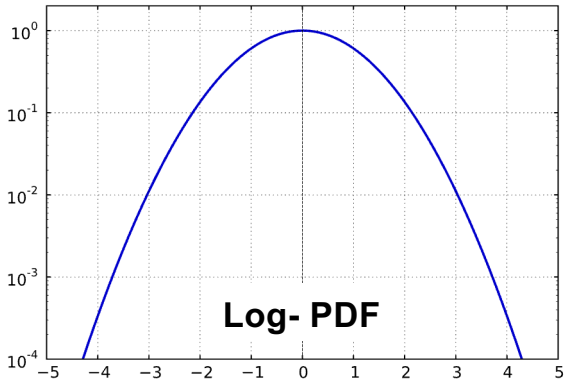
Gaussian (a.k.a. Normal) distribution with mean (location) μ and variance (squared scale) σ^2 parameters,

$$\mathcal{N}(y; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}(y-\mu)^2/\sigma^2\right)$$

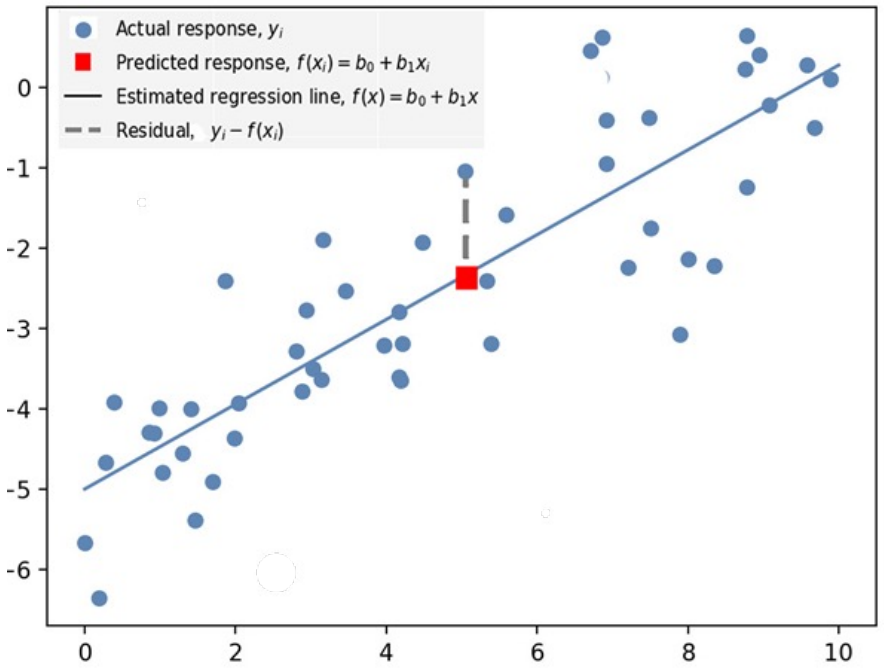
The logarithm of the PDF is just a negative quadratic,

$$\log \mathcal{N}(y; \mu, \sigma^2) = \underbrace{-\frac{1}{2}\log 2\pi - \log\sigma}_{\text{Constant w.r.t. mean}} \underbrace{- \frac{1}{2\sigma^2}(y-\mu)^2}_{\text{Quadratic Function of mean}}$$





MLE of Linear Regression



Substitute linear regression prediction into MLE solution and we have,

$$\arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2$$

So for Linear Regression,
MLE = Least Squares Estimation

https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

Linear Regression Summary

1. The linear regression model (assumption),

$$y = w^T x + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

2. For N iid training data fit using least squares,

$$w^{\mathrm{OLS}} = \arg\min_w \sum_{i=1}^{N} (y^{(i)} - w^T x^{(i)})^2$$

3. Equivalent to maximum likelihood solution

A word on matrix inverses…

$$w^{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

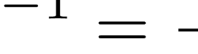
Least squares solution requires inversion of the term,

$$(\mathbf{X}^T\mathbf{X})^{-1}$$

What are some issues with this?

1. Requires $\mathcal{O}(D^3)$ time for D input features
2. May be numerically unstable (or even non-invertible)

$$(x+\epsilon)^{-1} = \frac{1}{x+\epsilon}$$



if x is small, then small numerical errors in input can lead to large errors in solution

Pseudoinverse

$$w^{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Use *Moore-Penrose pseudoinverse* (‘dagger’ notation)

$$w^{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{\dagger}\mathbf{X}^T\mathbf{y}$$

- Generalization of the standard matrix inverse for nonvertible matrices.
- Directly computable in most libraries
- In Numpy it is: `linalg.pinv`

Linear Regression in Scikit-Learn

Load your libraries,

For Evaluation

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

Load data,

```
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
```

^: same as diabetes_X[:,[2]]

Samples total	442
Dimensionality	10
Features	real, -.2 < x < .2
Targets	integer 25 - 346

Train / Test Split:

```
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

```
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
```

Linear Regression in Scikit-Learn

Train (fit) and predict,

```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
```

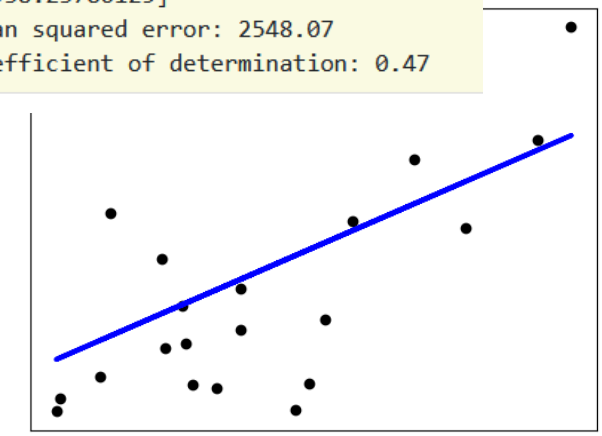
```
Coefficients:
 [938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47
```

Plot regression line with the test set,

```
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



`regr.coef_` : coefficient (array)
`regr.intercept_` : intercept (float)

Outline

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

Alternatives to Ordinary Least Squares (OLS)

Recall: OLS solution may not

$$w^{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Use *Moore-Penrose pseudoinverse* ('dagger' notation)

$$w^{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{\dagger}\mathbf{X}^T\mathbf{y}$$

Or, use L2 Regularized Least Squares (RLS)

$$w^{\mathrm{L2}} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{y}$$

For scalar, this inverse matrix is something like $\frac{1}{x^2+\lambda}$

why is this called regularized least squares?

Regularization

$$w^{\mathrm{L2}} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{y}$$

Turns out, w^{L2} is the solution of

$$w^{\mathrm{L2}} = \arg\min_w \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

recall: $\|w\| = \sqrt{\sum_{d=1}^{D} w_d^2}$

λ: Regularization Strength

$\|w\|^2$:Regularization Penalty

Prefers smaller magnitudes for w!

λ very small: almost OLS, especially when $\lambda = 0$, it is exactly OLS.

λ very large: $w \approx 0$ and high trainset error

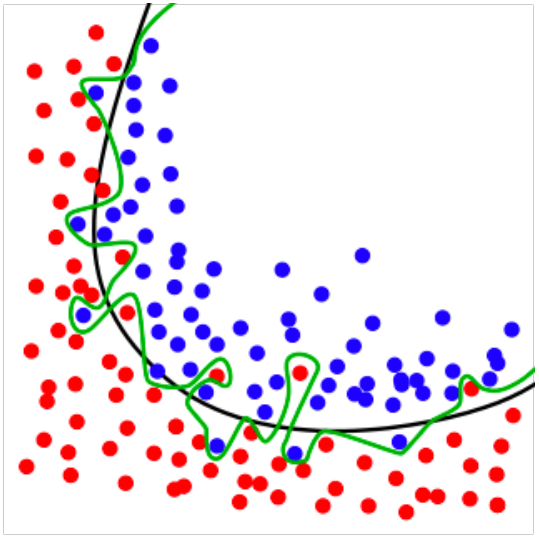
Regularization

Okay, we have a training data. Why not learn the most complex function that can work flawlessly for the training data and be done with it? (i.e., classifies every data point correctly)

Extreme example: Let's memorize the data. To predict an unseen data, just follow the label of the closest memorized data.

Doesn't generalize to unseen data – called *overfitting* the training data.

Solution: Fit the train set but don't "over-do" it. This is called **regularization**.



green: almost memorization
black: true decision boundary

- 1d case
  - Suppose that $y = wx + \epsilon$, and the true model is $w = 0$ $(y = \epsilon)$
  - However, OLS is highly probable to 'exaggerate' the effect of x to decrease train set error: (overfitting)

$$w = \frac{\sum_i y^{(i)} x^{(i)}}{\sum_j (x^{(j)})^2}$$

  - On the other hand, RLS will try to balance the train set error and the penalty caused by the large norm

$$w^{RLS} = \frac{\sum_i y^{(i)} x^{(i)}}{\sum_j (x^{(j)})^2 + \lambda}$$

$$|w^{RLS}| < |w^{OLS}|$$

Regularization

$$w^{\mathrm{RLS}} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{y}$$

Turns out, w^{RLS} is the solution of

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

recall: $\|w\| = \sqrt{\sum_{d=1}^{D} w_d^2}$

λ: Regularization Strength

$\|w\|^2$:Regularization Penalty

In short, the benefits of L2-RLS

- No need to worry about the estimator being undefined (due to matrix inversion)
- Avoid overfitting (if λ is chosen well)!

Notes on L2 Regularization

- Feature weights are "shrunk" towards zero – statisticians often call this a "shrinkage" method
- Common practice: Do **not** penalize bias (y-intercept, w_D) parameter,

$$\min_w \sum_i (y^{(i)} - w^T x^{(i)})^2 + \frac{\lambda}{2} \sum_{d=1}^{D-1} w_d^2$$

Recall: we enforced $x_D^{(i)} = 1$ so that w_D encodes the intercept

  - Penalizing intercept will make solution depend on origin for Y. i.e., add a constant c to $y^{(i)}$'s $\Longrightarrow$ the solutions changes!
- Typically, we standardize (e.g. Z-score) features before fitting model (Sklearn `StandardScaler`)
  - Why? If we multiply 0.1 to feature d, w_d needs to be much larger to compensate for it. But the regularization would prevent that!

As a general rule, if you are unsure, do standardization for every ML algorithm anyways

Scikit-Learn : L2 Regularized Regression

sklearn.linear_model.Ridge

class sklearn.linear_model.Ridge(*alpha=1.0, *, fit_intercept=True, normalize='deprecated', copy_X=True, max_iter=None, tol=0.001, solver='auto', positive=False, random_state=None*) [source]

Minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

Alpha is what we have been calling λ

alpha : *{float, ndarray of shape (n_targets,)}, default=1.0*

Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization. Alpha corresponds to `1 / (2C)` in other linear models such as `LogisticRegression` or `LinearSVC`. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.

Scikit-Learn : L2 Regularized Regression

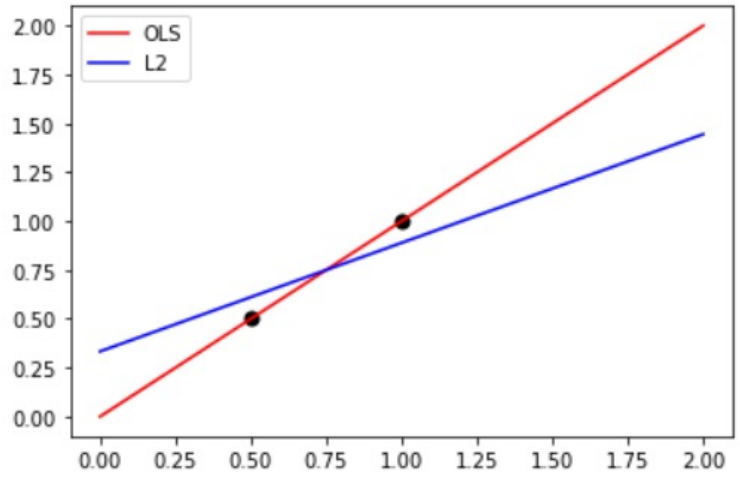
Define and fit OLS and L2 regression,

```
ols=linear_model.LinearRegression()
ols.fit(X_train, y_train)
ridge=linear_model.Ridge(alpha=0.1)
ridge.fit(X_train, y_train)
```

Plot results,

```
fig, ax = plt.subplots()
ax.scatter(X_train, y_train, s=50, c="black", marker="o")
ax.plot(X_test, ols.predict(X_test), color="red", label="OLS")
ax.plot(X_test, ridge.predict(X_test), color="blue", label="L2")

plt.legend()
plt.show()
```



quiz candidate

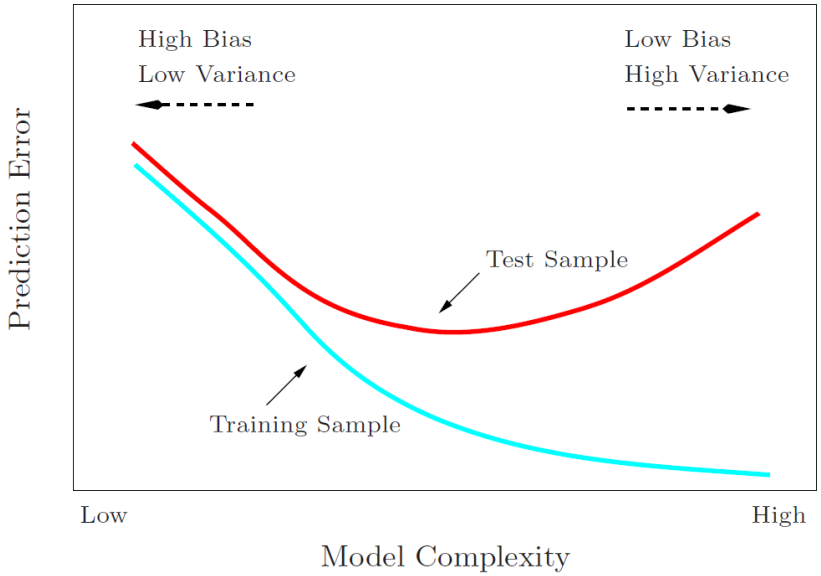
Q: why L2 has a lower slope?

L2 (Ridge) reduces impact of any single data point

Choosing Regularization Strength

We need to tune regularization strength to avoid over/under fitting…

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$



Recall bias/variance tradeoff

High regularization *reduces* model complexity: *increases* bias / *decreases* variance

Q: How should we properly tune λ ?

cross validation!

CSC380: Principles of Data Science

Linear Models 2

Kyoungseok Jang

Announcement

- HW5 minor announcement
  - In problem 1b, we said ‘Please use StandardScalar to fit and transform all of your data.’
  - It does not mean that you should use StandardScalar on your whole data frame.
  - Just apply StandardScalar on your X_train.
  - I will post it on piazza soon.

Review – important points

- Linear Regression
  - Definition of inner product, linear equation
  - We set our prediction model as $y = w^\top x$.
  - Its graph will look like a line/plane.
  - We want to find a line as close to every training data point as possible.
  - Distance: residual $y^{(i)} - w^\top x^{(i)}$.
  - How to integrate all these residuals at the same time?

Review – important points

- Least Square Estimation (LSE)
  - We want to find a line that minimize the sum of squared residuals
  - $w^* = \arg\min_w \sum_{i=1}^{m} \left(y^{(i)} - w^\top x^{(i)}\right)^2$.
  - We call it as least squares regression.
  - Don't need to remember the details now, but
    - This loss function is quadratic:
      - convex (curves up only),
      - unique minimum given by zero derivative,
      - can find the closed-form solution
    - When we assume the label model $y = x^{\top x} + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$, then this least square estimator is equivalent to MLE.

Review - important points

- Regularized Least Squares
  - LSE on our previous slide (Ordinary Least Square, OLS) has two problems
    - Non-invertibility problem (divide it by zero problem)
    - Overfitting
  - To prevent this issue, we introduced a new method called L2-RLS.
    - $w^* = \arg\min_w \sum_{i=1}^{m} \left(y^{(i)} - w^\top x^{(i)}\right)^2 + \lambda ||w||^2$
    - This regularization term tends to reduces the impact of each coordinate.
    - λ: hyperparameter. Use Cross Validation!

$||w||^2$:**Regularization Penalty**

λ: **Regularization Strength**

Outline

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
  - L2 Regularized Least Squares
  - L1 Regularized Least Squares
- Logistic Regression

Regularized Least Squares

Ordinary least-squares (OLS) estimation (no regularizer),

$$w^{\mathrm{OLS}} = \arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2$$

L2 norm: $\|w\| = \sqrt{\sum_{d=1}^{D} w_d^2}$

L1 norm: $\|w\|_1 = \sum_{d=1}^{D} |w_d|$

L2-regularized Least-Squares (Ridge)

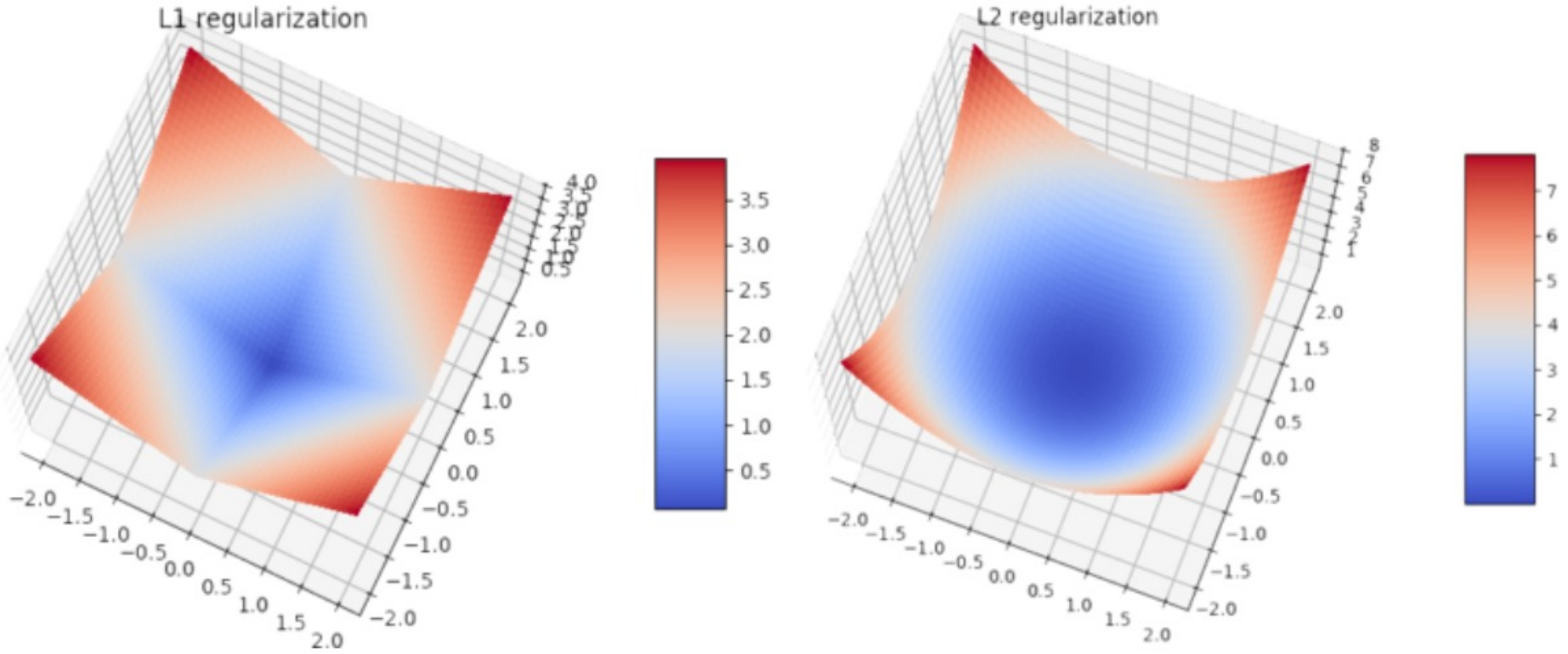
$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

Convention: Just saying 'RLS' means L2-RLS

L1-regularized Least-Squares (LASSO) LASSO: Least Absolute Shrinkage and Selection Operator

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|_1$$

L1 vs L2



Looks subtle but L2-RLS and L1-RLS are often quite different!

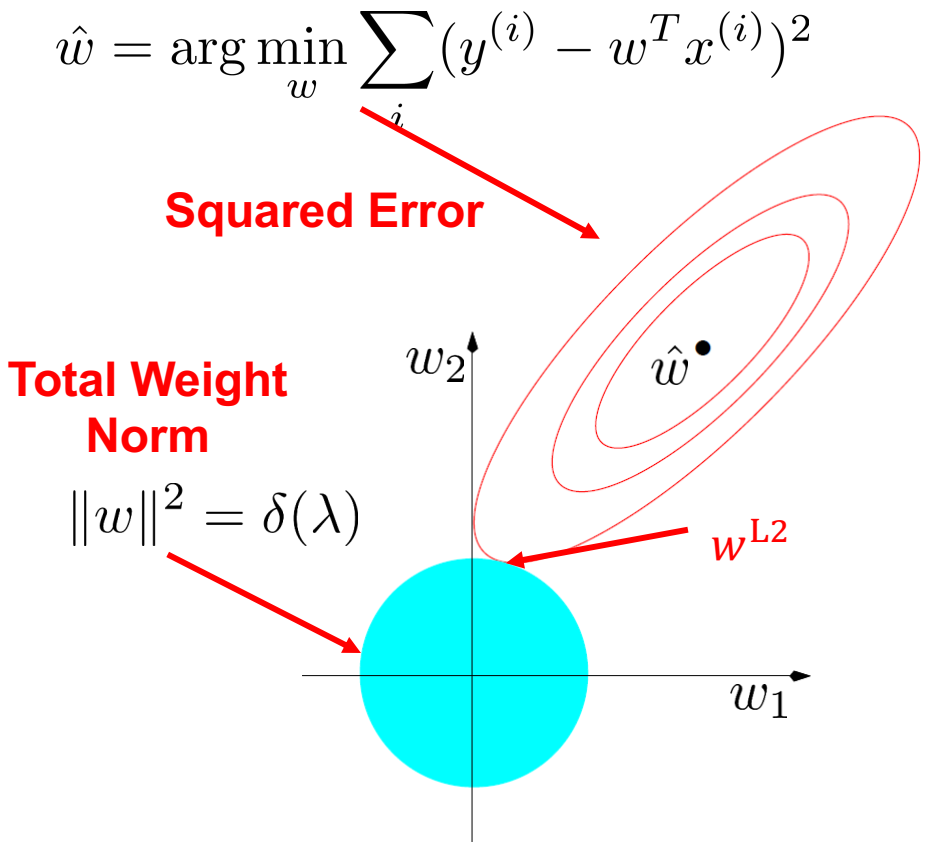
Constrained Optimization Viewpoint

(Theorem) If

$$w^{\mathrm{L2}} = \arg\min_w \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

then there exists a function $\delta(\lambda)$ s.t.

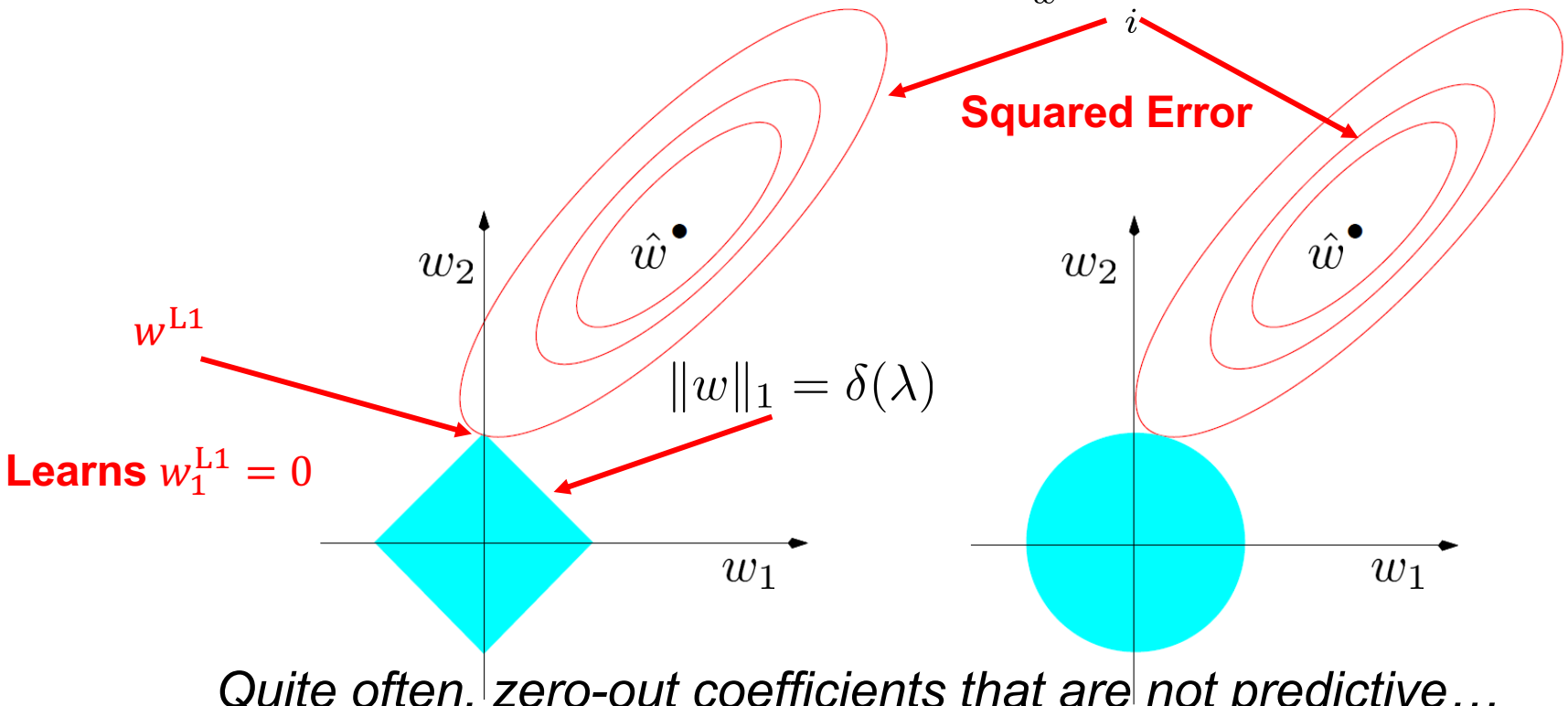
$$w^{\mathrm{L2}} = \arg\min_w \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2$$
$$\text{subject to} \quad \|w\|^2 \le \delta(\lambda)$$



[Source: Hastie et al. (2001)]

L1 Regularized Least-Squares

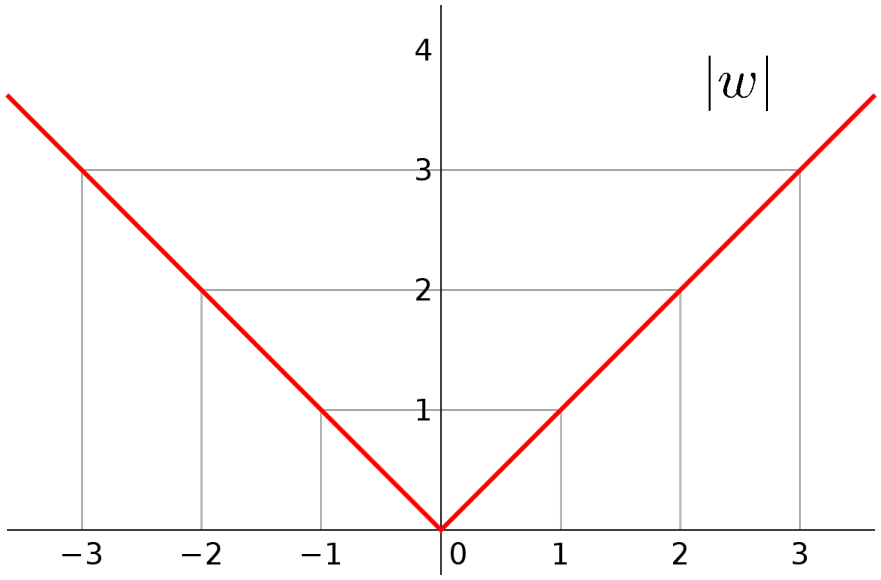
$$\hat{w} = \arg\min_{w} \sum_{i} (y^{(i)} - w^T x^{(i)})^2$$



Quite often, zero-out coefficients that are not predictive…

Learning L1 Regularized Least-Squares

$$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|_1$$



Not differentiable…

$$\frac{d}{dx}|x|$$

…doesn't exist at x=0

Can't set derivatives to zero as in the L2 case!

Learning L1 Regularized Least-Squares

- **Not differentiable**, no closed-form solution. => Need to use iterative methods

- But it is **convex**!
  - Global minimum can be found!
  - Efficient optimization algorithms exist

- *Least Angle Regression* (LAR) computes full solution path for a range of values λ

sklearn.linear_model.Lasso

class sklearn.linear_model.Lasso(*alpha=1.0, *, fit_intercept=True, normalize='deprecated', precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic'*) ¶ [source]

Parameters:

alpha : *float, default=1.0* ← this is λ

Constant that multiplies the L1 term. Defaults to 1.0. `alpha = 0` is equivalent to an ordinary least square, solved by the `LinearRegression` object. For numerical reasons, using `alpha = 0` with the `Lasso` object is not advised. Given this, you should use the `LinearRegression` object.

fit_intercept : *bool, default=True*

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

precompute : *'auto', bool or array-like of shape (n_features, n_features), precompute*

Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument. For sparse input this option is always `False` to preserve sparsity.

copy_X : *bool, default=True*

If `True`, X will be copied; else, it may be overwritten.

Specialized methods for cross-validation…

sklearn.linear_model.LassoCV

class sklearn.linear_model.LassoCV(*, *eps=0.001*, *n_alphas=100*, *alphas=None*, *fit_intercept=True*, *normalize='deprecated'*, *precompute='auto'*, *max_iter=1000*, *tol=0.0001*, *copy_X=True*, *cv=None*, *verbose=False*, *n_jobs=None*, *positive=False*, *random_state=None*, *selection='cyclic'*) [source]

Tries out a range of α values and reports the best, but maintains other values of α as well.

L1 Regression Cross-Validation

Perform L1 Least Squares (LASSO) 20-fold cross-validation,

```
model = LassoCV(cv=20).fit(X, y)
```

or

Plot the error for range of alphas,

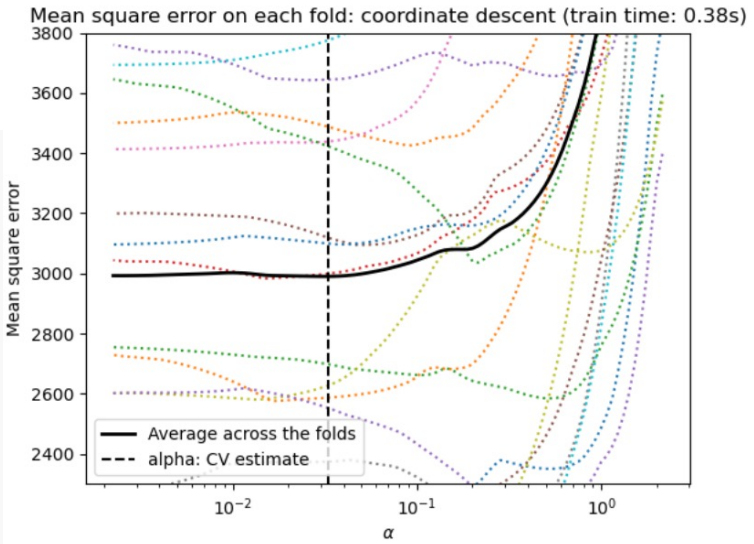
```
plt.figure()
ymin, ymax = 2300, 3800
plt.semilogx(model.alphas_ + EPSILON, model.mse_path_, ":")
plt.plot(
    model.alphas_ + EPSILON,
    model.mse_path_.mean(axis=-1),
    "k",
    label="Average across the folds",
    linewidth=2,
)
plt.axvline(
    model.alpha_ + EPSILON, linestyle="--", color="k", label="alpha: CV estimate"
)
```

all these colored dotted lines for each test fold

all alphas_

adds vertical line

the best alpha



Example: Prostate Cancer Dataset

Term	LS	Ridge	Lasso
Intercept	2.465	2.452	2.468
lcavol	0.680	0.420	0.533
lweight	0.263	0.238	0.169
age	−0.141	−0.046	
lbph	0.210	0.162	0.002
svi	0.305	0.227	0.094
lcp	−0.288	0.000	
gleason	−0.021	0.040	
pgg45	0.267	0.133	

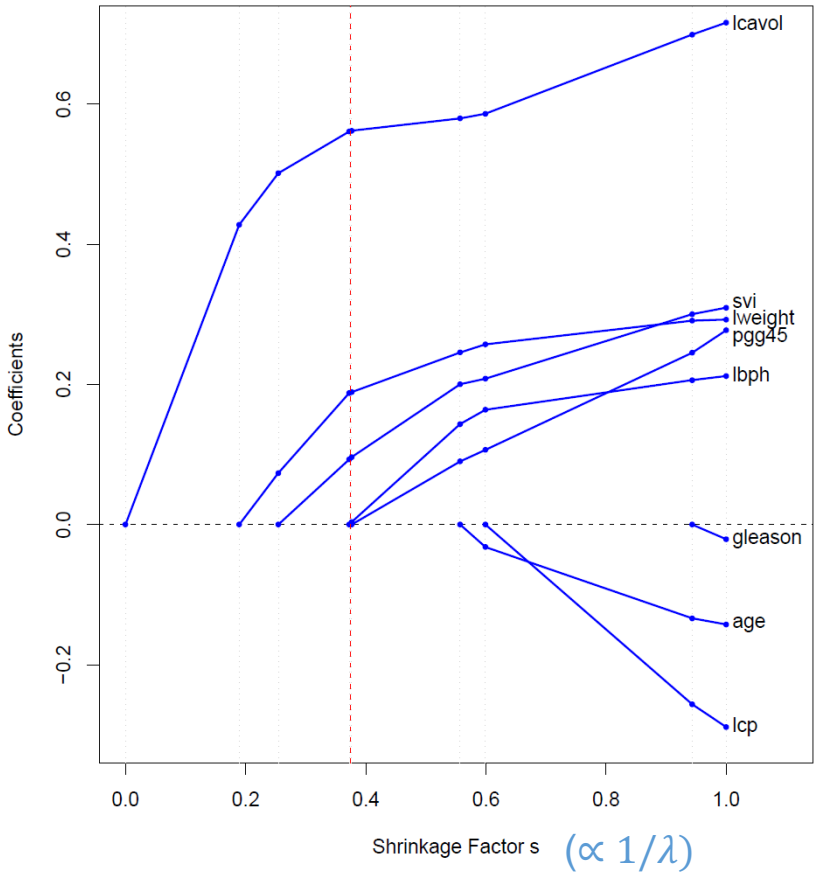
Task: predict logarithm of prostate specific antigen (PSA).

Best LASSO model learns to ignore several features (age, lcp, gleason, pgg45).

Wait…Is **age** really not a significant predictor of prostate cancer? What's going on here?

Age is highly correlated with other factors and thus *not significant* in the presence of those factors

Feature Weight Profiles

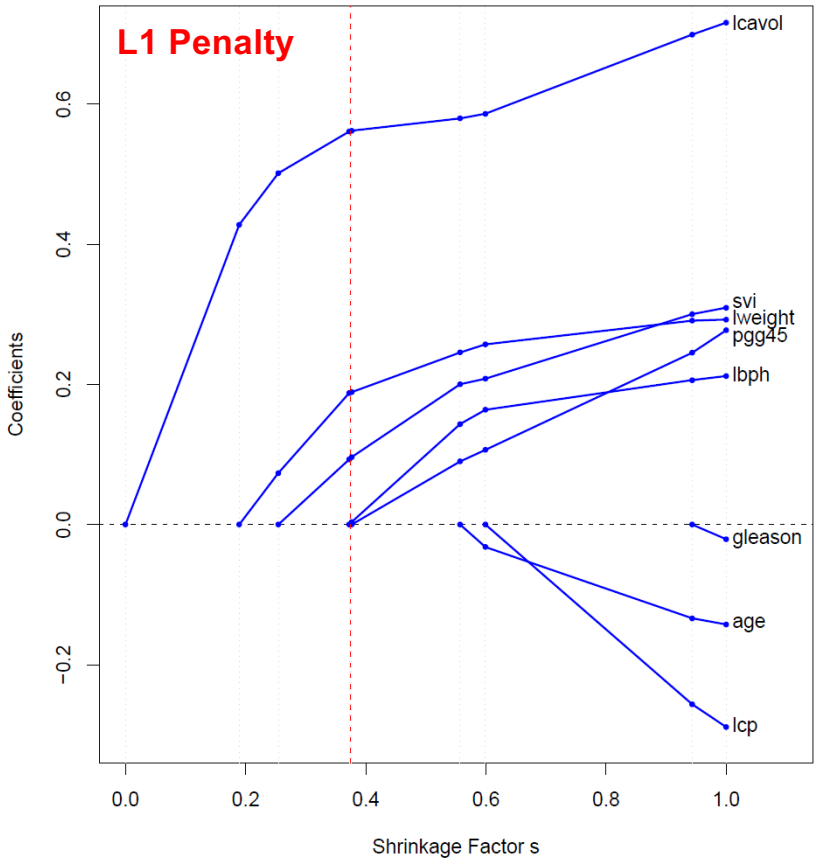


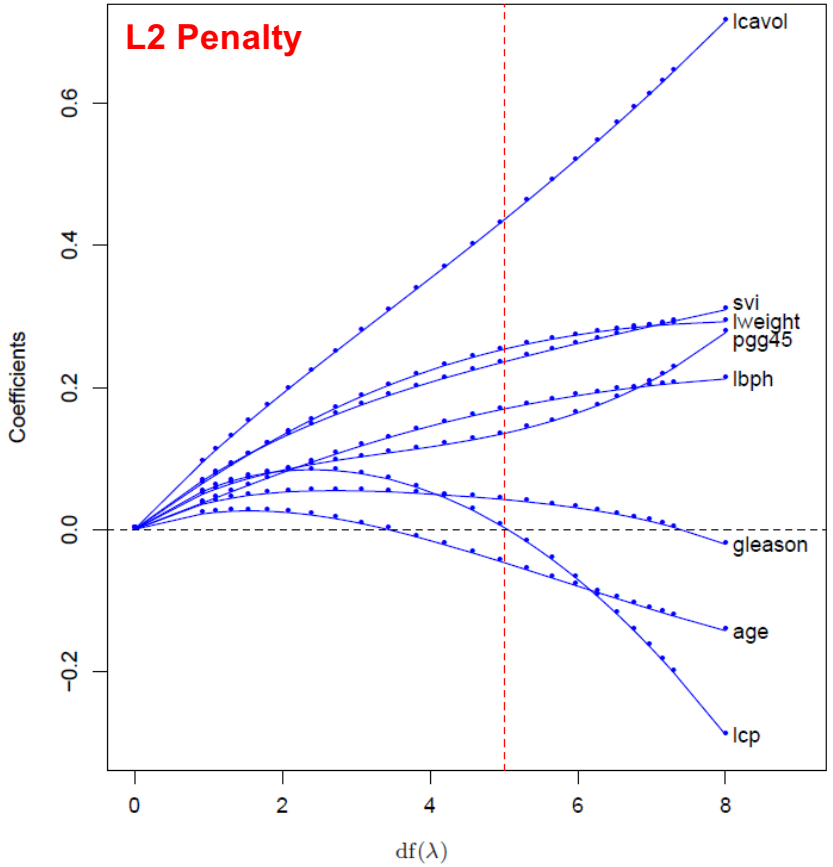
Varying regularization parameter adjusts *shrinkage factor*

For moderate regularization strength weights for many features go to zero

- Induces *feature sparsity*
- Ideal for high-dimensional settings since it reduced variance from having too many features!
- Gracefully handles D>m case, for D features and m training data

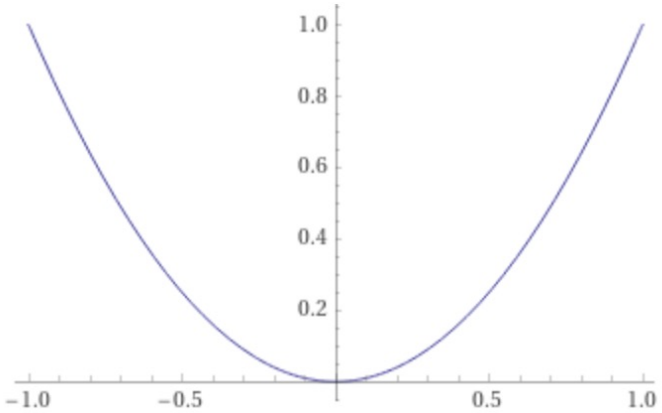
Feature Weight Profiles

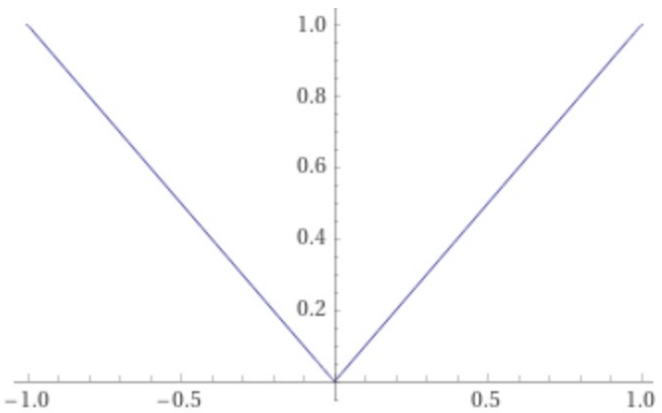




Why does this happen?

- Lasso, or L1 regularization tries to ‘eliminate’
  - Ridge (L2): punishes less when a parameter is ‘close’ to the optimal.
  - Lasso (L1): wherever you are, the magnitude of your slope is always the same – relatively larger penalty on the small difference.





Feature Selection

- **A big open problem in ML**: being able to throw in all the possible features, but still perform as good as knowing the truly relevant features ahead of time (i.e., not affected by irrelevant features)
  - Recall irrelevant features can be harmful.
  - E.g.) Among billions of genes, only hundreds of genes are related to breast cancer.

Best-Subset Selection

The optimal strategy for p features looks at models over *all possible combinations* of features,

```
For k in 1,…,p:
  subset = Compute all subset of k-features (p-choose-k)
  For kfeat in subset:
    model = Train model on kfeat features
    score = Evaluate model using cross-validation
Choose the model with best cross-validation score
```

Best-Subset Selection : Prostate Cancer Dataset

Time complexity

- Data have 8 features, there are 8-choose-k subsets for each k=1,…,8 for a total of 255 models
- Using 10-fold cross-val requires 10 x 255 = 2,550 training runs!
- In general, $O(2^p)$ time complexity

… who can afford exponential time complexity?

Feature Selection: Prostate Cancer Dataset

Best subset works well!

reasonably good test error, low standard deviation, and only based on two features!

Term	LS	Best Subset	Ridge	Lasso
Intercept	2.465	2.477	2.452	2.468
lcavol	0.680	0.740	0.420	0.533
lweight	0.263	0.316	0.238	0.169
age	−0.141		−0.046	
lbph	0.210		0.162	0.002
svi	0.305		0.227	0.094
lcp	−0.288		0.000	
gleason	−0.021		0.040	
pgg45	0.267		0.133	
Test Error	0.521	0.492	0.492	0.479
Std Error	0.179	0.143	0.165	0.164

[Source: Hastie et al. (2001)]

Forward Sequential Selection

An efficient method adds the most predictive feature one-by-one

```
featSel = empty
featUnsel = All features
For iter in 1,…,p:
  For kfeat in featUnsel:
    thisFeat = featSel + kfeat
     model = Train model on thisFeat features
     score = Evaluate model using cross-validation
  featSel = featSel + best scoring feature
  featUnsel = featUnsel – best scoring feature
Choose the model with best cross-validation score
```

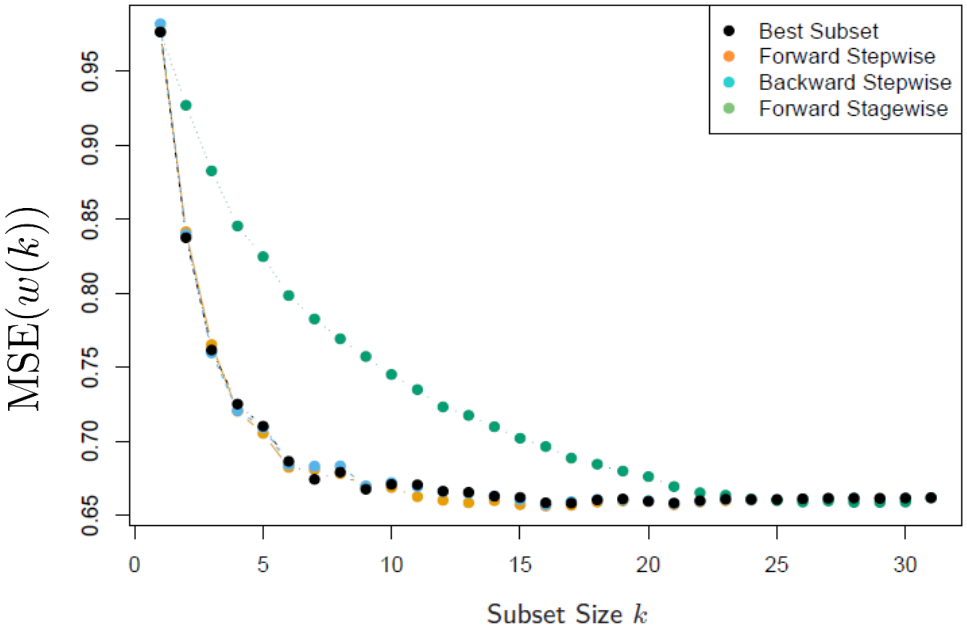
Backward Sequential Selection

Backwards approach starts with *all* features and removes one-by-one

```
featSel = All features
For iter in 1,…,p:
  For kfeat in featSel:
    thisFeat = featSel - kfeat
     model = Train model on thisFeat features
     score = Evaluate model using cross-validation
  featSel = featSel – worst scoring feature
Choose the model with best cross-validation score
```

Comparing Feature Selection Methods

Sequential selection is greedy, but often performs well…



Example Feature selection on synthetic model with p=30 features with pairwise correlations (0.85). True feature weights are all zero except for 10 features, with weights drawn from N(0,6.25).

Sequential selection with p features takes $O(p^2)$ time, compared to exponential time for best subset

Sequential feature selection available in Scikit-Learn under:
`feature_selection.SequentialFeatureSelector`

General Principles of Regularization

- From the loss function point of view

$$\text{Model} = \min_{\text{model}} \text{Loss}(\text{Model}, \text{Data}) + \lambda \cdot \text{Regularizer}(\text{Model})$$

Regularization Strength

Regularization Penalty

- We will see more examples of loss functions going forward.

CSC380: Principles of Data Science

Linear Models 3

Kyoungseok Jang

Announcement

- HW6 is out – due April. 12th
- HW5 score will be posted on April 7th
- New TA: Tugay Bilgis
  - We will be able to have office hours on Thursday 10 – 11 am.
- Sorry for the wrong instruction on HW5
  - Group submission is allowed.

Review – what I hope you to know

- Regularized Least Squares – We learned LASSO
  - Instead of L2 regularizer $||w||^2$, use $||w||_1$ (know the definition)
  - Properties of LASSO
    - Pros: tries to zero out the coefficients as much as possible - less features to care
    - Cons: not differentiable, no closed-form solution (but convex) → Iterative
- Feature selection: which features are meaningful
  - Best subset selection: check all possible combination – computationally expensive
  - Forward Sequential Selection
  - Backward Sequential Selection

Review – what I hope you to know

- Logistic Regression – Students said that this part was too hard to understand, so I will briefly talk about it again
  - This is a method to use linear regression in classification problems
    - Natural way – if $w^\top x \geq 0$, label +1 and else, -1.
  - Why linear regression is bad
    - Output can be much greater than the label.
    - Too sensitive to residuals – sensitive to outliers and points that are far from the actual decision boundary (though they are certain)
  - Understand the logistic function
    - Formula, the shape of the function – same sign with its input, (0,1) bounded
    - We use this function to use 'Bernoulli' likelihood instead of 'Gaussian'
    - Now it's not too sensitive to residuals, because of the shape of the function
    - **Linear decision boundary**

Review – what I hope you to know

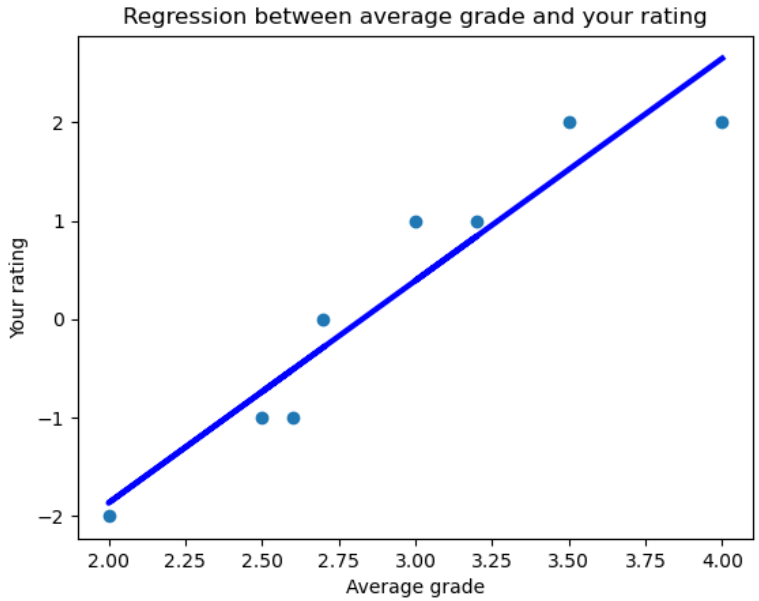
- Logistic Regression
  - Regularization
    - Tries to be a step function – weight w goes to infinity!
    - L1 or L2 Regularizer is used
  - Loss function perspective
    - Zero-one loss is intuitive for classification problem, but hard to optimize
    - We understand this logistic regression by using ‘upper bound function’ of this zero-one loss. Convex, differentiable, thus easy to optimize

Outline

- Linear Regression
- Least Squares Estimation
- Regularized Least Squares
- Logistic Regression

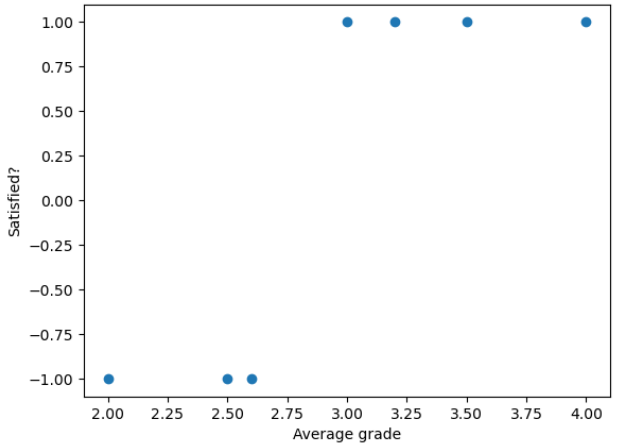
Classification as Regression

- Regression problem: suppose that you want to train a function that takes ‘average grade of that course’ as input and ‘**your rating**’ as its output.



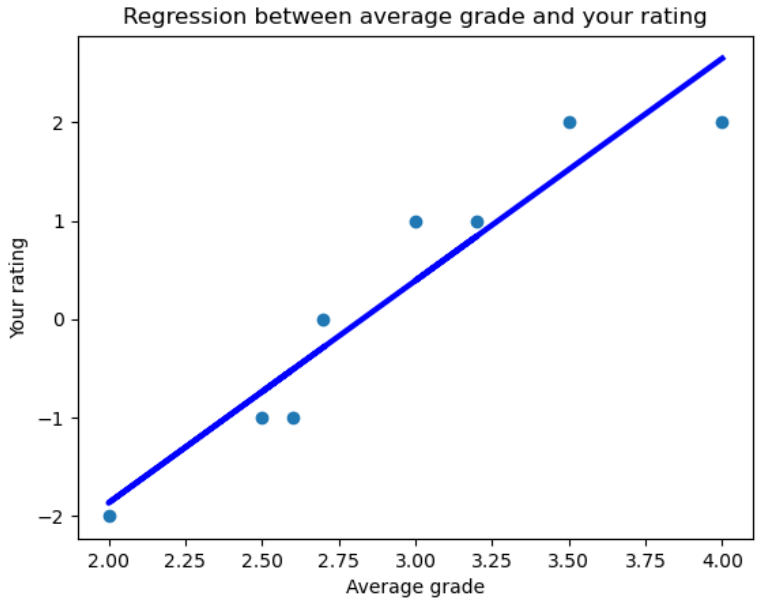
Classification as Regression

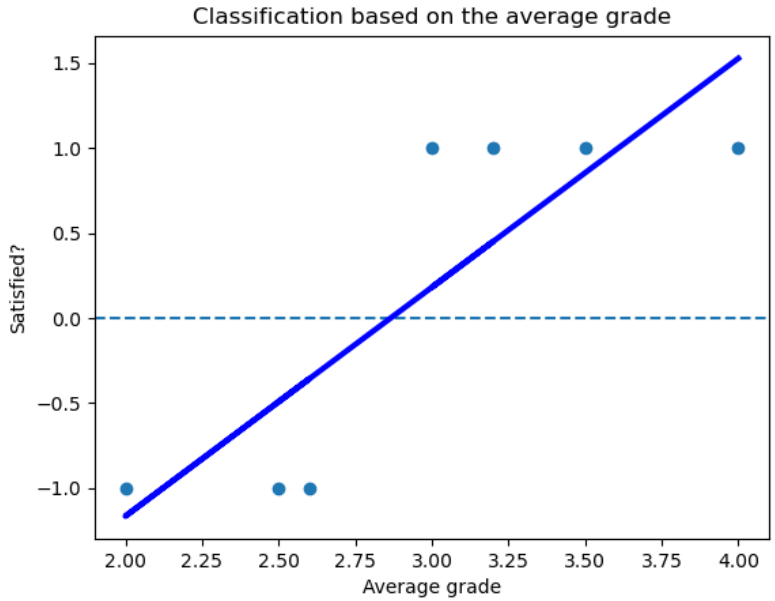
- Classification problem: suppose that you want to train a function that takes ‘average grade of that course’ as input and ‘**whether you satisfied or not**’ as its output.
  - We can only use the ‘sign’ of the rating, not the actual value.
- We can expect the ‘trend’ will not change much.
  - Positive when the average grade is high



Classification as Regression

- We could use regression for this Boolean label too.
- What will be a reasonable classification criterion?
  - If your prediction result is positive, classify it as ‘good’ class





Classification as Regression

- Maybe there might be a hidden ‘likeliness’ behind the classification problem. Simply we don’t have access to it.

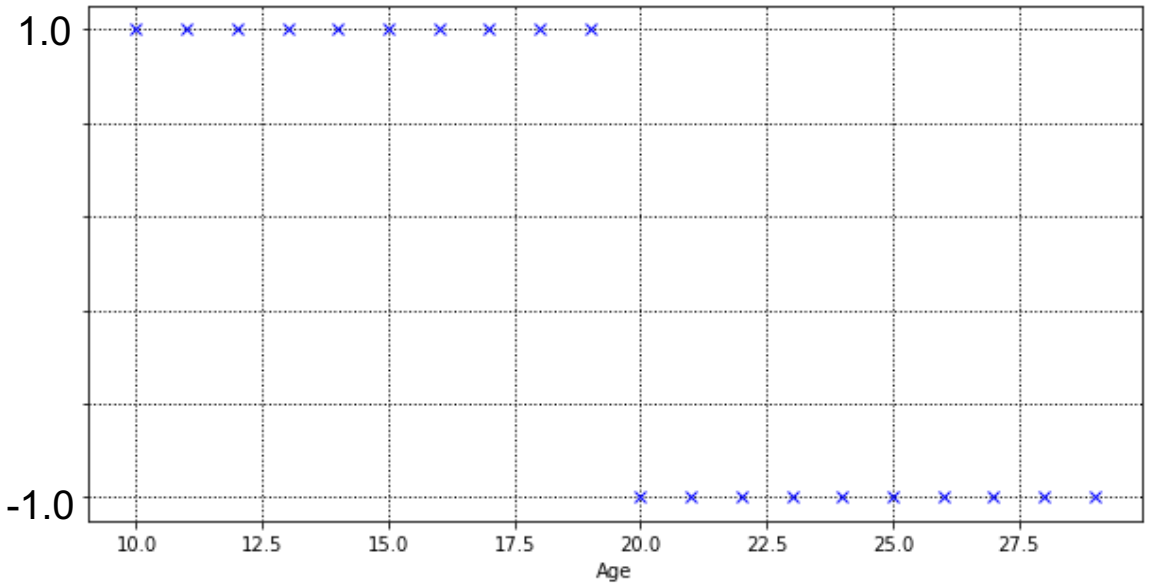
Totally a dog image

???

Totally a muffin image

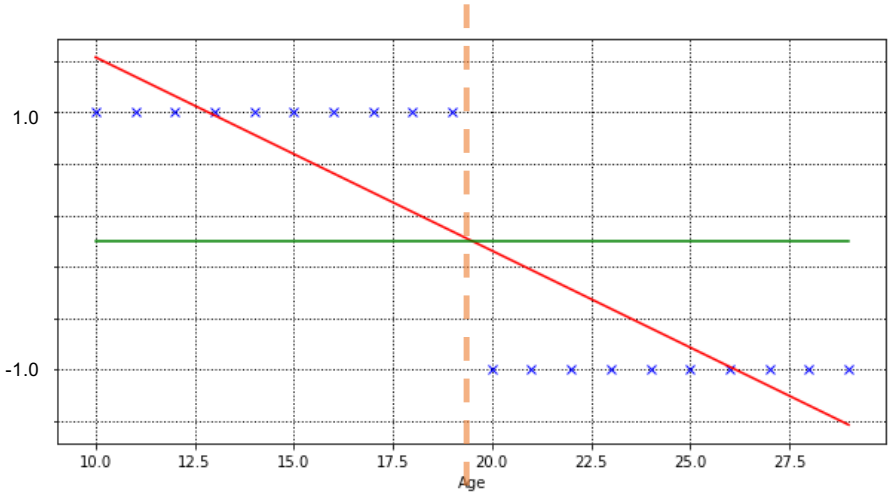
Classification as Regression

Suppose our response variables are binary y={-1,1}. How can we use linear regression ideas to solve this classification problem?



https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Classification as Regression



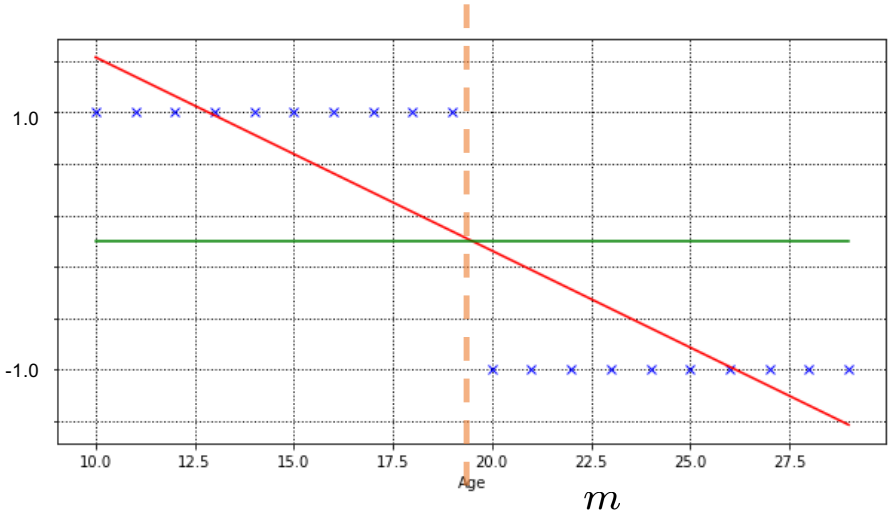
Idea Fit a regression function (red) to the data. Classify points based on whether they are *above* or *below* the (green).

predict 1 if $w^\top x \geq 0$
0 if $w^\top x < 0$

- This is called a *discriminant* function, since it discriminates between classes
- It is a linear function and so is a *linear discriminant*
- Decision boundary is the point at which $w^\top x$ becomes exactly 0 (orange dashed line)

https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Classification as Regression



Idea Fit a regression function (red) to the data. Classify points based on whether they are *above* or *below* the (green).

predict 1 if $w^\top x \geq 0$
0 if $w^\top x < 0$

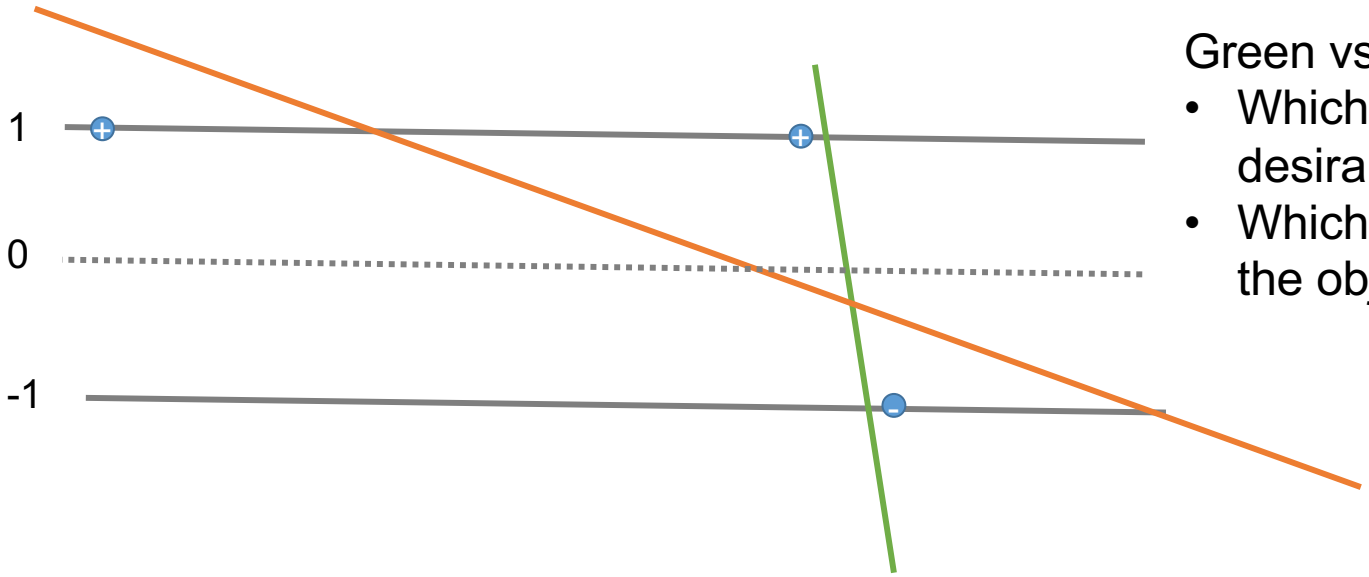
Recall: $$w^{\mathrm{L2}} = \arg\min_w \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 + \lambda \|w\|^2$$

Turns out, this is not a desirable approach. Any guess?

https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Why Classification as Regression is Not Desirable

Recall: $$w^{\mathrm{L2}} = \arg\min_{w} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2$$



Green vs orange:

- Which one is more desirable?
- Which one will minimize the objective function?

linear regression solution

this is what I would pick

Why Classification as Regression is Not Desirable (2)

Recall the probabilistic motivation for linear regression:

Assume $x \sim \mathcal{D}_X$ from some distribution. We then assume that

$$y = w^T x + \epsilon \quad \text{where } \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Equivalently,

$$p(y|x; w) = \mathcal{N}(w^T x, \sigma^2)$$

assuming Gaussian for binary variable!!

Q: What would be a reasonable alternative?

$$y \sim \text{Bernoulli}(p = w^\top x)$$

Q: Once we compute the estimate $\widehat{w}$, how do we make prediction for x^*

$$y^* = \arg \max_{y' \in \{0,1\}} p(y = y' \mid x^*; \widehat{w})$$

Q: But we will have to find w with $w^\top x^{(i)} \in (0,1)$ for all train data point. Why?

Otherwise, the log-likelihood is not well-defined. What is Bernoulli(1.5)?

Q: Say we do MLE with this to get $\widehat{w}$. Any issues for making predictions?

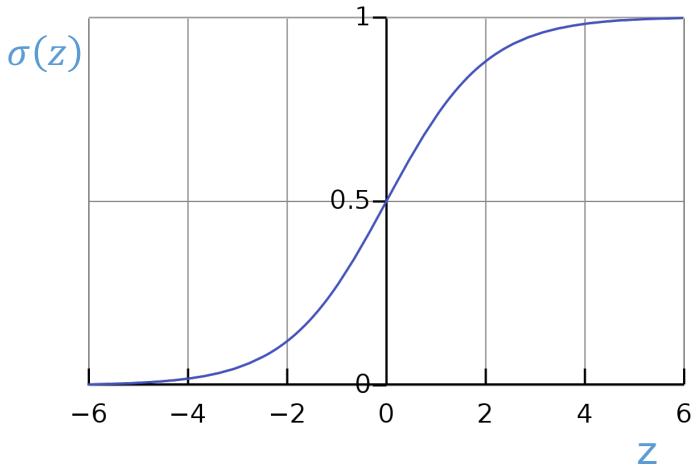
For test data point x^*, $\widehat{w}^\top x^*$ may lie outside [0,1]!

Logistic Regression

Idea Distort the prediction $w^\top x$ in some way to map to [0,1] so that it is always a probability.

$$\sigma(w^\top x) \text{ instead of } w^\top x$$

where $\sigma(w^T x) = \frac{\exp(w^T x)}{1+\exp(w^T x)}$



That is, assume

$$y \sim \text{Bernoulli}\big(p = \sigma(w^\top x)\big)$$

- **Logistic function** is a type of *sigmoid function*, since it maps any value to the range [0,1]
- Logistic also widely used in Neural Networks – for classification last layer is typically just a logistic regression

Logistic Regression

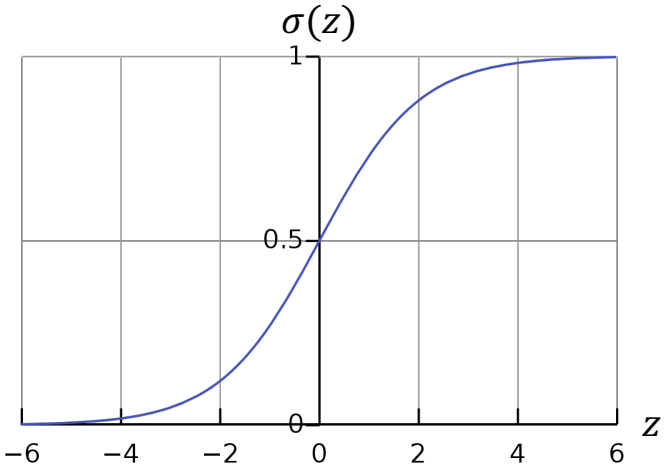
Model:

$$\mathrm{y} \sim \mathrm{Bernoulli}\big(p = \sigma(w^{\mathsf{T}} x)\big)$$

Train: compute the MLE $\hat{w}$

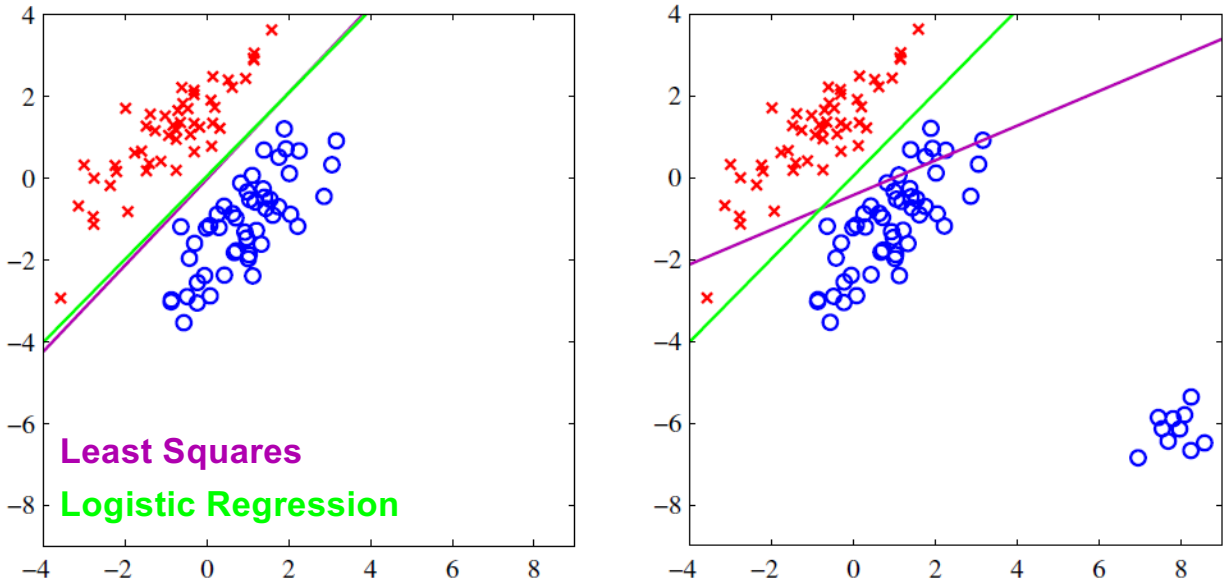
Test: Given test point x^* compute

$$y^* = \arg\max_{v \in \{0,1\}} p(y = v \mid x^*; \hat{w})$$



- **Equivalent to $y^* = \mathbf{I}\{\hat{w}^{\mathsf{T}} x^* \geq 0\}$ ➔ Linear boundary!**
- Why?
- $\max_{v \in \{0,1\}} p(y = v \mid x^*; \hat{w}) = \max(\sigma(\hat{w}^{\mathsf{T}} x^*), 1 - \sigma(\hat{w}^{\mathsf{T}} x^*))$
- $\sigma(\hat{w}^{\mathsf{T}} x^*) > 1 - \sigma(\hat{w}^{\mathsf{T}} x^*)$ when $\sigma(\hat{\mathbf{w}}^{\mathsf{T}} \mathbf{x}^*) \geq \mathbf{1/2}$, which is equivalent to $\hat{\mathbf{w}}^{\mathsf{T}} \mathbf{x}^* \geq \mathbf{0}$!

Least Squares vs. Logistic Regression

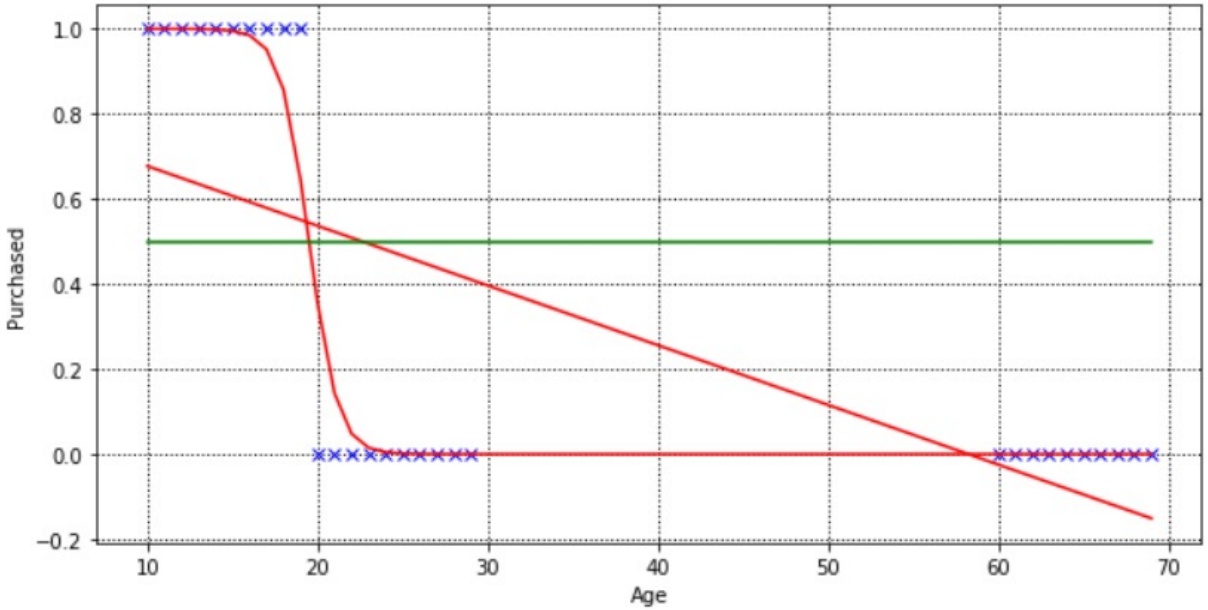


- Both models learn a linear decision boundary
- 😄Least squares can be solved in closed-form (convex objective)
- ☹Least squares is sensitive to outliers

[Source: Bishop “PRML”]

Least Squares vs. Logistic Regression

Similar results in 1-dimension



Fitting Logistic Regression

Fit by maximizing likelihood—start with the *binary* case

For the students who are not familiar with calculus and algebra:

- We use Bernoulli likelihood function
  - Note) We used Gaussian likelihood function for linear regression
- No closed-form solution – use iterative method
  - Fortunately, the objective function is concave.

Fitting Logistic Regression

Fit by maximizing likelihood—start with the *binary* case

Posterior probability of class assignment is Bernoulli,

$$p(y \mid x; w) = p(y = 1 \mid x; w)^y (1 - p(y = 1|x; w))^{(1-y)}$$

Given N iid training data pairs the log-likelihood function is,

$$\begin{aligned}
\mathcal{L}_m(w) &= \sum_{i=1}^{m} \log p(y_i \mid x_i; w) \\
&= \sum_i \{y_i \log p(y_i = 1 \mid x_i; w) + (1 - y_i) \log p(y_i = 0 \mid x_i; w)\} \\
\text{(algebra)} \qquad &= \sum_i \left\{ y_i w^T x_i - \log \left(1 + e^{w^T x_i}\right) \right\}
\end{aligned}$$

Fitting Logistic Regression

$$w^{\mathrm{MLE}} = \arg\max_{w} \sum_{i} \left\{ y^{(i)} w^T x^{(i)} - \log\left(1 + e^{w^T x^{(i)}}\right) \right\}$$

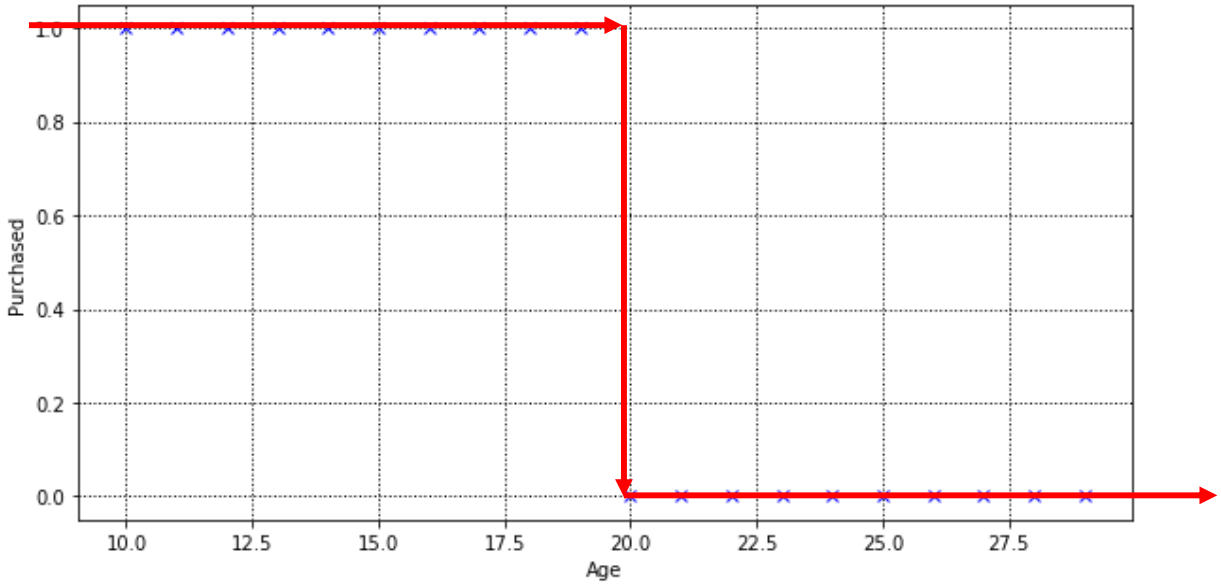
Computing the derivatives with respect to each element w_d,

$$\frac{\partial \mathcal{L}}{\partial w_d} = \sum_{i} x_d^{(i)} \left(y^{(i)} - \frac{e^{w^T x^{(i)}}}{1 + e^{w^T x^{(i)}}} \right) = 0$$

- For D features this gives us D equations and D unknowns
- Does not give a closed-form solution.
- Need to use iterative methods to solve it
- The objective function is concave => global solution can be found!

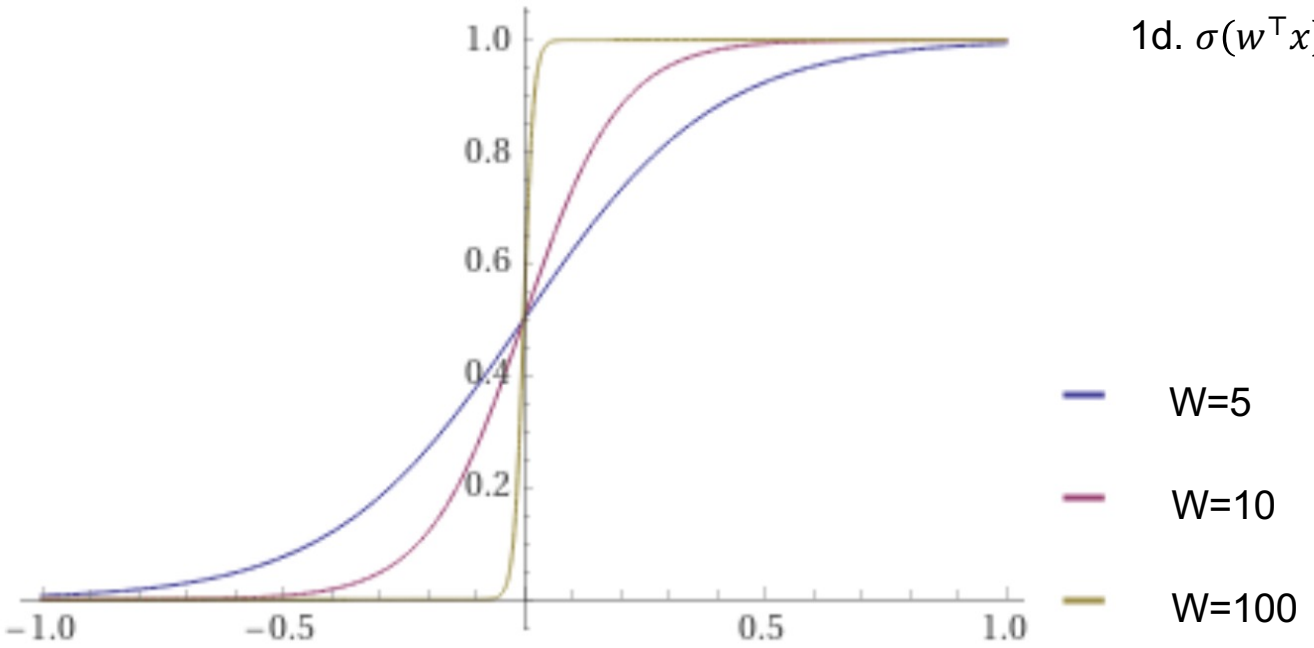
Potential Issues

- What is the ‘best function’ that explains this dataset?
  - Step function!



Potential Issues

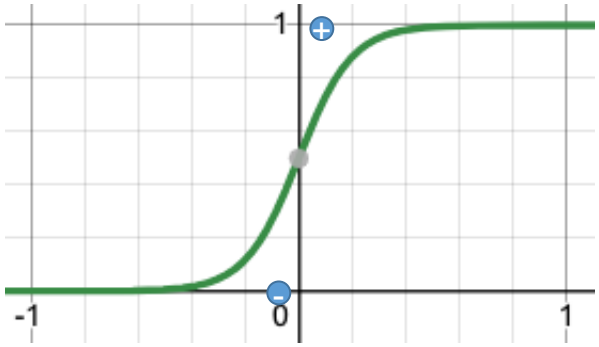
- As w gets larger, the logistic function converges to the ‘step function’



Potential Issues

- Imagine that we adjust w manually to make this sigmoid curve to be as close as possible to training point.
  - We will have to make it a step function!
  - This can be achieved by taking w to be infinity!
- In fact, this is what happens with logistic regression in this dataset.
  - In general, happens when the dataset is linearly separable
- Issue: **iterative algorithms will never converge**

linearly separable: there exists w that can have 0 train error



Q: Solution?

1d. $\sigma(w^\top x) = \frac{e^{wx}}{1+e^{wx}}$

Need for Regularization

Solution: add a regularization!

$$w^{\mathrm{L2}} = \arg\max_w \sum_i \left\{ y^{(i)} w^T x^{(i)} - \log\left(1 + e^{w^T x^{(i)}}\right) \right\} - \lambda \|w\|^2$$

$$= \arg\min_w \sum_i \left\{ -y^{(i)} w^T x^{(i)} + \log\left(1 + e^{w^T x^{(i)}}\right) \right\} + \lambda \|w\|^2$$

L1 regularization also possible

- Shares the same 'feature selection' property!

$$w^{\mathrm{L1}} = \arg\min_w \sum_i \left\{ -y^{(i)} w^T x^{(i)} + \log\left(1 + e^{w^T x^{(i)}}\right) \right\} + \lambda \|w\|_1$$

A Loss Function point of view

Recall: linear regression has two viewpoints: (1) loss function (2) probabilistic model

We used (2) probabilistic models and used Bernoulli distribution to compute likelihood.

Which means the output classes are 0 and 1.

What if we change it to -1 and 1, as our first example?

Classification: our version

predict 1 if $w^\top x \geq 0$
0 if $w^\top x < 0$

right if $w^\top x \geq 0 \; and \; y = 1$
or $w^\top x < 0 \; and \; y = 0$
wrong otherwise

messy

new version

predict 1 if $w^\top x \geq 0$
-1 if $w^\top x < 0$

right if $y \cdot w^\top x \geq 0$
wrong if $y \cdot w^\top x < 0$

neat

A Loss Function point of view

Recall: linear regression has two viewpoints: (1) loss function (2) probabilistic model

Reformulation: take positive label as +1 and negative label as -1

- Loss function $\ell(w, x, y)$ for x, y

$$\ell(w, x, y) = -yw^\top x + \log\left(1 + e^{w^\top x}\right)$$

- Set $y' = 2y - 1 \in \{-1, 1\}$. Then,

$$= \log\left(1 + e^{-y'w^\top x}\right)$$

exercise:
try plugging in y=0 above and y'=-1 above and verify the equality. try again with with y=1 and y'=1

- So, if the label is encoded as $y = \pm 1$, then

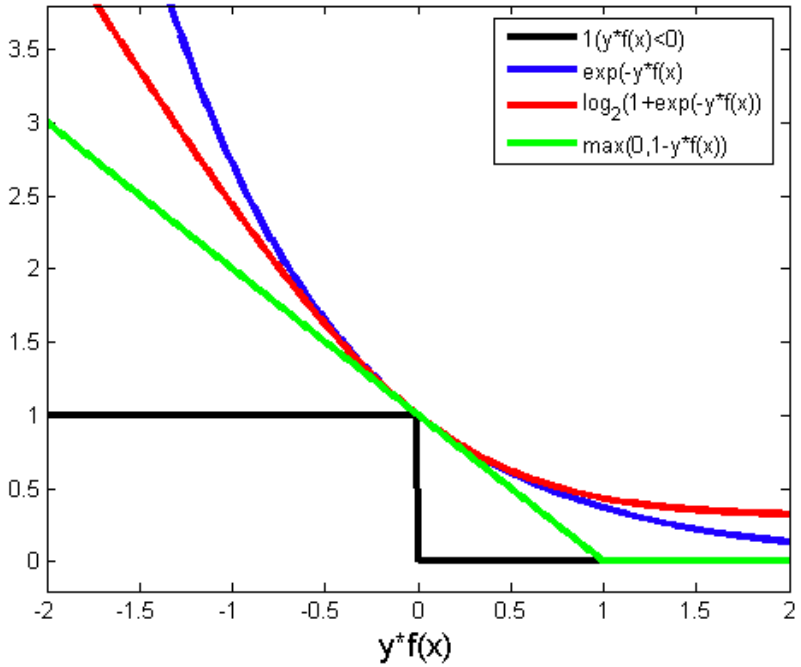
$$w^{\mathrm{L2}} = \arg\min_w \sum_i \left\{\log\left(1 + e^{-y^{(i)}w^\top x^{(i)}}\right)\right\} + \lambda\|w\|^2$$

Logistic Regression Motivated by a Loss Function

- Black: classification error (aka zero-one loss)
  $\ell(w, x, y') = \mathbf{I}\{y' \cdot w^\top x < 0\}$ with $y' \in \pm 1$
- Red: "logistic loss"
  $$\ell(w, x, y') = \log\left(1 + e^{-y' w^\top x}\right)$$

Interpretation

- The zero-one loss is not convex and hard to optimize
  - in fact, proven to be NP-hard)
- Logistic loss is a convex upper bound on the zero-one loss. => easier to optimize!



sklearn.linear_model.LogisticRegression

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]

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.

tol : *float, default=1e-4*

Tolerance for stopping criteria.

C : *float, default=1.0* $C = 1/\lambda$

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

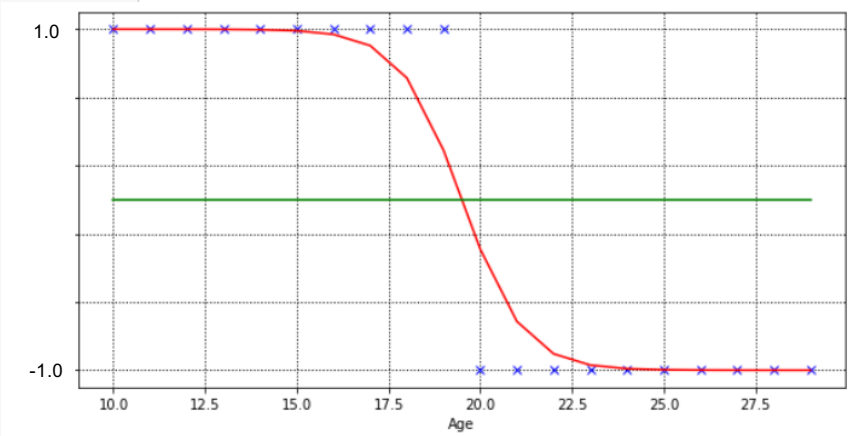
Scikit-Learn Logistic Regression


```
log_regression = sklearn.linear_model.LogisticRegression()
```

```
_ = log_regression.fit(pd.DataFrame(x), y)

y_pred = log_regression.predict_proba(pd.DataFrame(x))
log_y_pred_1 = [item[1] for item in y_pred]

fig = plt.figure(figsize=(10,5))
xlabel = 'Age'
ylabel = 'Purchased'
plt.xlabel(xlabel)
plt.ylabel(ylabel)
plt.grid(color='k', linestyle=':', linewidth=1)
plt.plot(x, y, 'xb')
plt.plot(x, log_y_pred_1, '-r')
_ = plt.plot(x, line_point_5,'-g')
```



Function `predict_proba(X)` returns prediction of class assignment probabilities for each class. It returns n by C matrix if n data points were provided as argument.

E.g.) probability of (cat, dog, horse) = (0.8, 0.1, 0.1)

https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

Extension: Multiclass

- What if we have more than 2 classes?
- For C classes,

$$y \mid x \sim \text{Categorical}(\pi) \quad \text{with} \quad \pi_j = \frac{\exp\left(w^{(j)\top} x\right)}{\sum_{c=1}^{C} \exp\left(w^{(c)\top} x\right)}$$

CD

- Alternatively, one can use

$$\pi_j = \frac{\exp\left(w^{(j)\top} x\right)}{1+\sum_{c=1}^{C-1} \exp\left(w^{(c)\top} x\right)} \text{ for } j = 1, \ldots, C-1, \text{ and then define } \pi_C = \frac{1}{1+\sum_{c=1}^{C-1} \exp\left(w^{(c)\top} x\right)}$$

(C-1)D

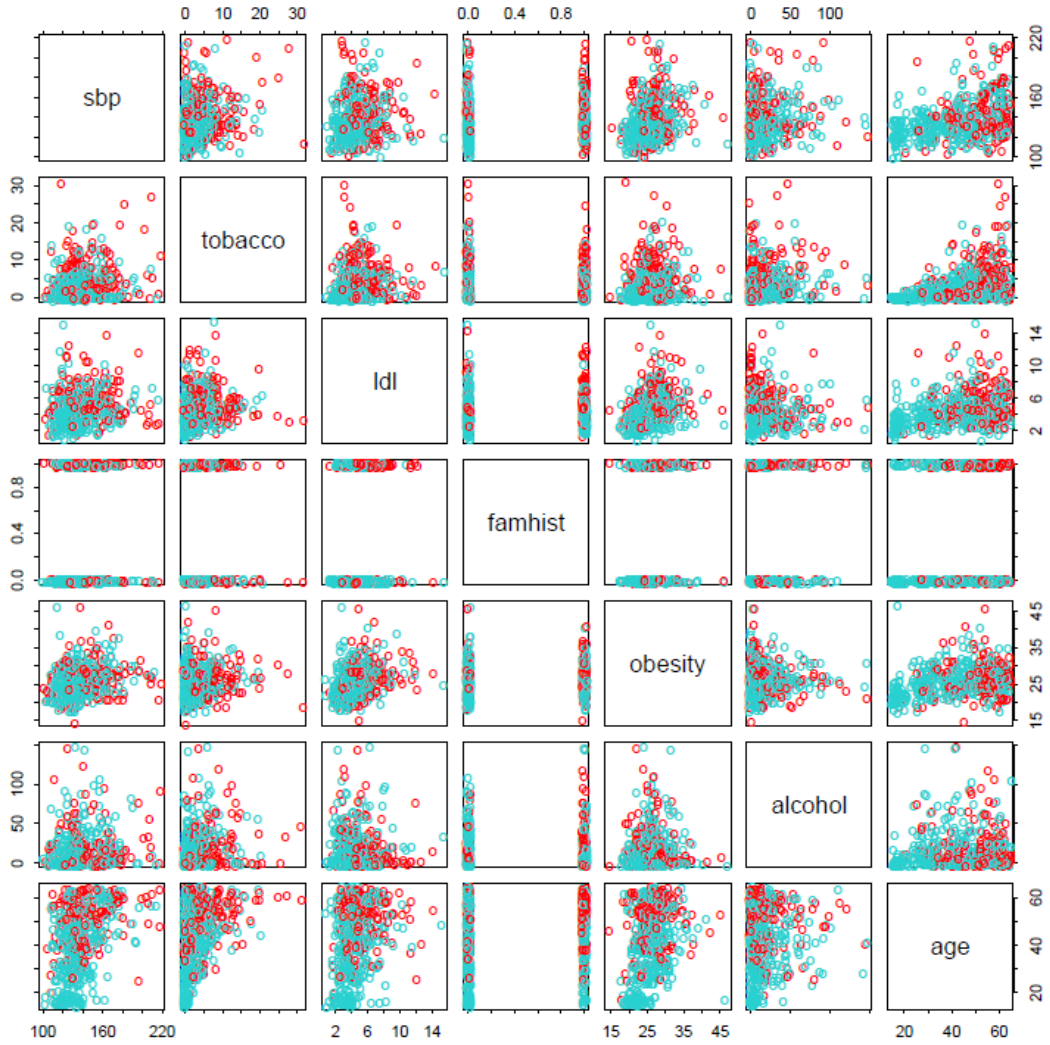
Q: Number of parameters for the top one and the bottom one (say D features)?

Using Logistic Regression

The role of Logistic Regression differs in ML and Data Science,

- In Machine Learning we use Logistic Regression for building **predictive** classification models
- In Data Science we often use it for **understanding** how features relate to data classes / categories

Example South African Heart Disease (Hastie et al. 2001)
Data result from Coronary Risk-Factor Study in 3 rural areas of South Africa. Data are from white men 15-64yrs and response is presence/absence of *myocardial infraction (MI).* How predictive are each of the features?



Looking at Data

Each scatterplot shows pair of risk factors. Cases **with** MI (**red**) and **without** (cyan)

Features

- Systolic blood pressure
- Tobacco use
- Low density lipoprotein (ldl)
- Family history (discrete)
- Obesity
- Alcohol use
- Age

[Source: Hastie et al. (2001)]

Example: African Heart Disease

	Coefficient	Std. Error	Z Score
(Intercept)	−4.130	0.964	−4.285
sbp	0.006	0.006	1.023
tobacco	0.080	0.026	3.034
ldl	0.185	0.057	3.219
famhist	0.939	0.225	4.178
obesity	-0.035	0.029	−1.187
alcohol	0.001	0.004	0.136
age	0.043	0.010	4.184

Goal: hypothesis testing on whether the coefficient is 0 or not (hope to reject the hypothesis that the coefficient is 0)

Fit logistic regression to the data using MLE estimate

Standard error is estimated standard deviation of the learned coefficients

Z-score of weights is a random variable from standard Normal,

$$w_d \div \mathrm{SE}(w_d) \sim \mathcal{N}(0, 1)$$

Thus, anything with Z-score > 2 is significant with 95% confidence.

Example: African Heart Disease

	Coefficient	Std. Error	Z Score
(Intercept)	−4.130	0.964	−4.285
sbp	0.006	0.006	1.023
tobacco	0.080	0.026	3.034
ldl	0.185	0.057	3.219
famhist	0.939	0.225	4.178
obesity	-0.035	0.029	−1.187
alcohol	0.001	0.004	0.136
age	0.043	0.010	4.184

E.g.) famhist's Z-score is 4.178.

which means that

if the true weight of $w_{famhist} = 0$,

then 'the probability that this much or large Z-score happening' is equal to $2\times\big(1-\Phi(4.178)\big) < 0.01\%$

Natural conclusion: $w_{famhist} \neq 0$ and it is significant

Z-score of weights is a random variable from standard Normal,

$$w_d \div \mathrm{SE}(w_d) \sim \mathcal{N}(0,1)$$

Thus, anything with Z-score > 2 is significant with 95% confidence.

Example: African Heart Disease

	Coefficient	Std. Error	Z Score
(Intercept)	−4.130	0.964	−4.285
sbp	0.006	0.006	1.023
tobacco	0.080	0.026	3.034
ldl	0.185	0.057	3.219
famhist	0.939	0.225	4.178
obesity	-0.035	0.029	−1.187
alcohol	0.001	0.004	0.136
age	0.043	0.010	4.184

Finding Systolic blood pressure (sbp) is **not a significant predictor**

Obesity is not significant and negatively correlated with heart disease in the model

Remember All correlations / significance of features are based on presence of *other features*. We must always consider that features are strongly correlated.

DO NOT INTERPRET IT AS CAUSALITY!

CSC380: Principles of Data Science

Nonlinear Models

Kyoungseok Jang

Outline

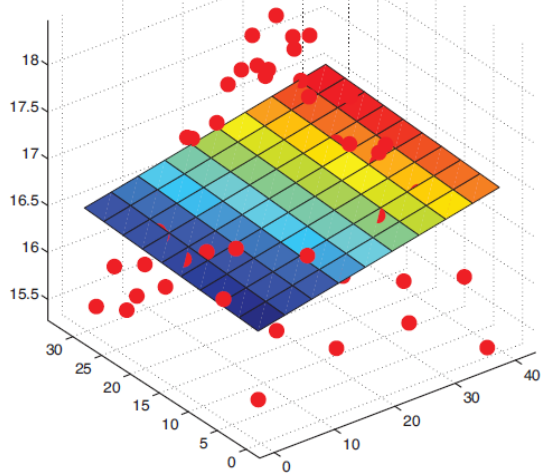
- Basis Functions
- Support Vector Machine Classifier
- Kernels
- Neural Networks

Outline

- Basis Functions
- Support Vector Machine Classifier
- Kernels
- Neural Networks

Linear Models

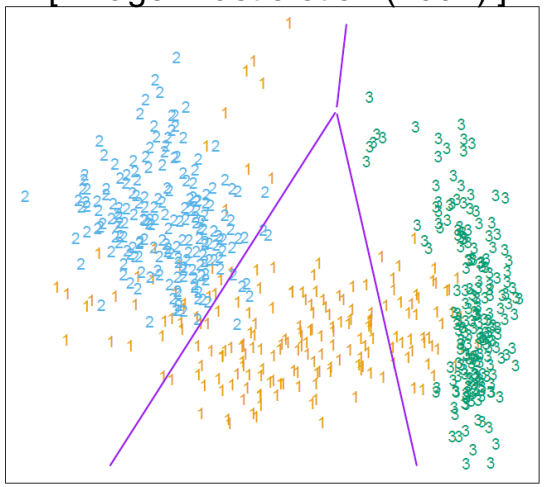
[Image: Murphy, K. (2012)]



Linear Regression Fit a *linear function* to the data,

$$y = w^T x$$

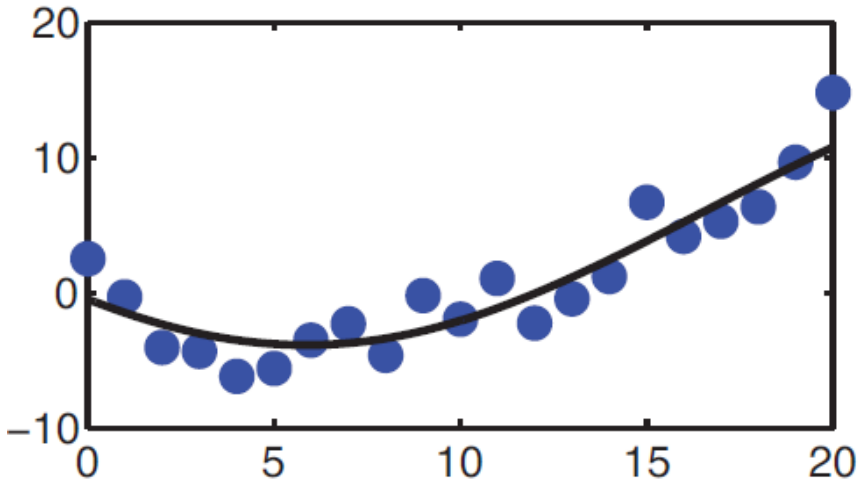
[Image: Hastie et al. (2001)]



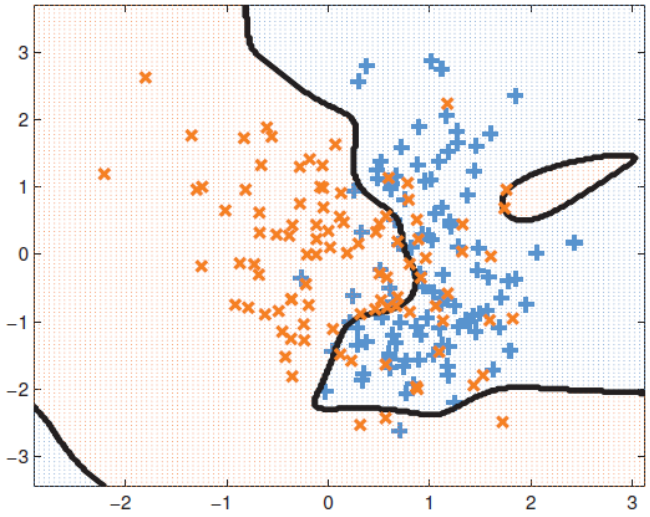
Logistic Regression Learn a decision boundary that is *linear in the data*,

$$y = \mathbf{I}\{w^\top x \geq 0\}$$

Nonlinear Data



What if our data are *not* well-described by a linear function?



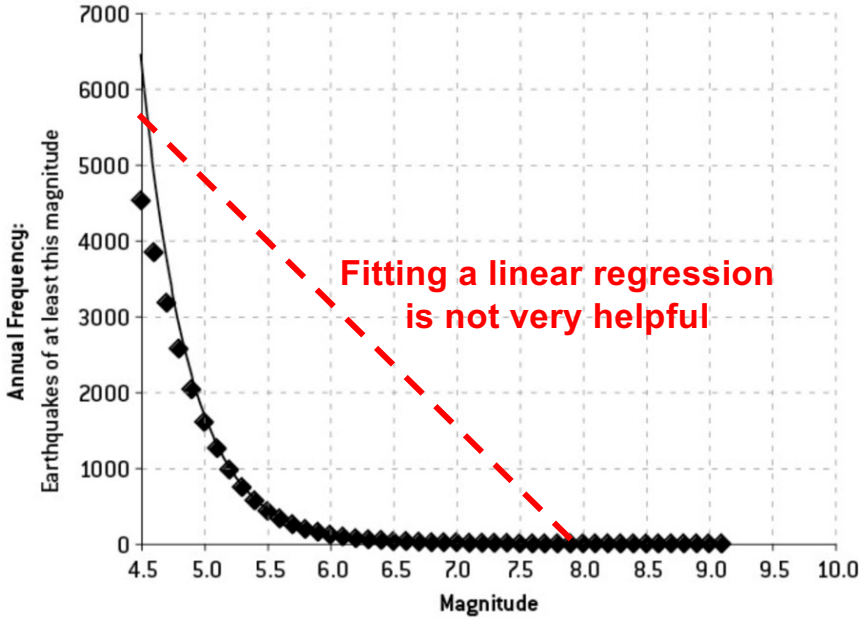
What if classes cannot be well-distinguished by a linear function?

[Source: Murphy, K. (2012)]

Example: Earthquake Prediction

Suppose that we want to predict the number of earthquakes that occur of a certain magnitude. Our data are given by,

FIGURE 5-3A: WORLDWIDE EARTHQUAKE FREQUENCIES, JANUARY 1964–MARCH 2012



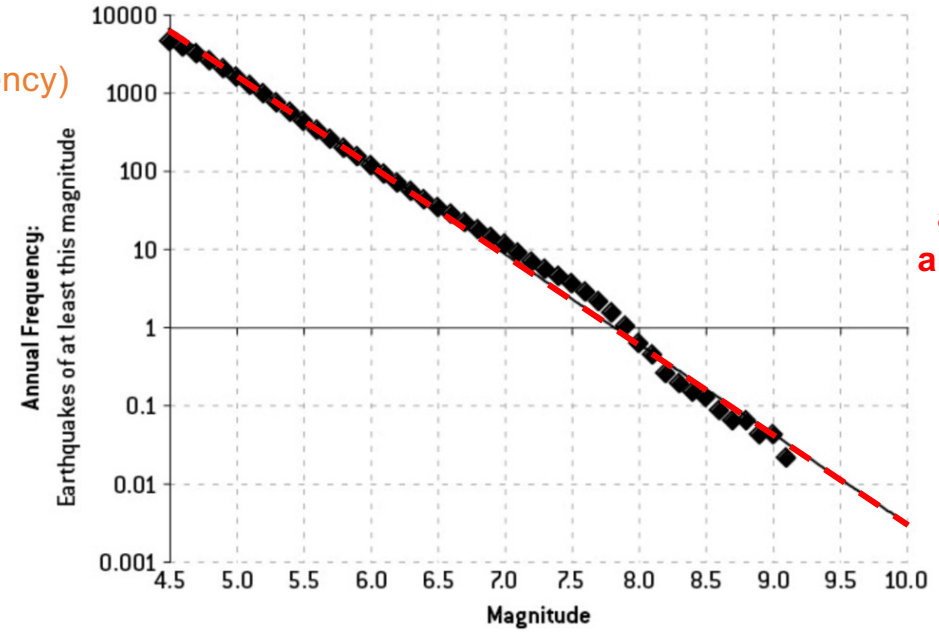
[Source: Silver, N. (2012)]

Example: Earthquake Prediction

Suppose that we want to predict the number of earthquakes that occur of a certain magnitude. Our data are given by,

FIGURE 5-3B: WORLDWIDE EARTHQUAKE FREQUENCIES, JANUARY 1964–MARCH 2012, LOGARITHMIC SCALE

(log frequency)



But plotting outputs on a logarithmic scale reveals a strong linear relationship…

it's like $y = e^{-ax+b}$

[Source: Silver, N. (2012)]

Beyond linearity: Transformation in x

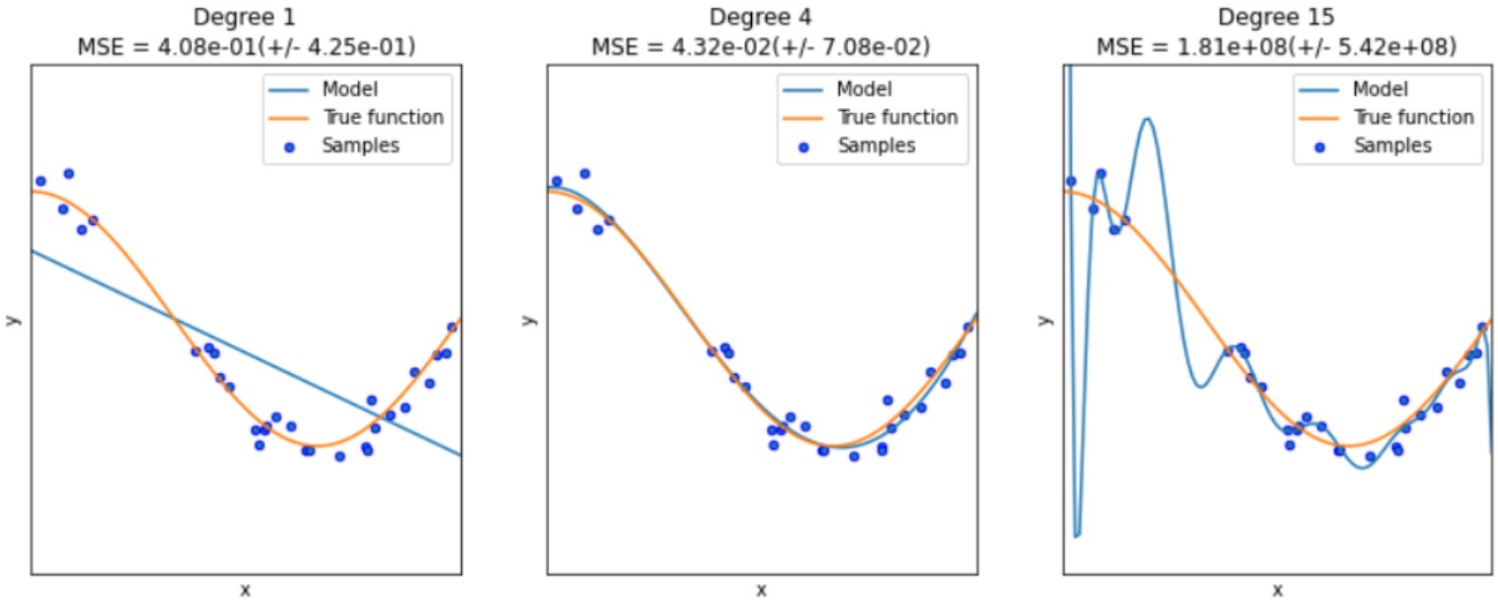
- Recall: for 1d problem, we embedded the feature: $x' = (x, 1) \in \mathbb{R}^2$ so we can encode the intercept term.

- Actually, the embedding trick is stronger.
  - $(x^2, x, 1)$: 2nd order polynomial with respect to x
  - $(x^d, x^{d-1}, \ldots, 1)$: d-th order polynomial (= degree d)

Beyond linearity: Transformation in x

- Example: suppose you have three data points
  - (x,y) = (0,1), (1,4), (2,9)
- True relationship: $y = (x+1)^2 = x^2 + 2x + 1$
- Linear regression cannot catch this relationship perfectly.
- Instead, create additional 'features' $x_0 = 1,\ x_2 = x^2$
- Now your dataset changes to
  - (x0, x, x2,y)=(1,0,0,1), (1,1,1,4), (1,2,4,9)
- Linear regression $y = w^{\mathsf{T}}\tilde{x} = w_0 x_0 + w_1 x + w_2 x_2$

Feature embedding trick



$(x, 1)$ $(x^4, x^3, x^2, x, 1)$ $(x^{15}, \ldots, x^3, x^2, x, 1)$

higher-order polynomial = higher complexity = prone to overfitting!

from https://datascience.foundation/sciencewhitepaper/underfitting-and-overfitting-in-machine-learning

Basis Functions

- A **basis function** can be any function of the input features X
- Define a set of B basis functions $\phi_1(x), \ldots, \phi_B(x)$
- Fit a linear regression model in terms of basis functions,

$$y = \sum_{b=1}^{B} w_b \phi_b(x) = w^T \phi(x)$$

notation:
$\phi(x) \coloneqq [\phi_1(x), \ldots, \phi_B(x)]$

- The model is *linear in the transformed basis/induced features* $\phi(x)$.
- The model is *nonlinear in the data X*

Common "All-Purpose" Basis Functions

- Linear basis functions recover the original linear model,

$$\phi_b(x) = x_b$$

Returns b[th] dimension of X

- Quadratic $\phi_b(x) = x_d^2$ or $\phi_b(x) = x_d x_e$ for any $d \neq e$ capture 2nd order interactions
- An order p polynomial $\phi(x) \rightarrow \left(x_d, x_d^2, \ldots, x_d^p\right)$ captures higher-order nonlinearities (but requires $O(d^p)$ parameters)
- You can mix and match:

$$\phi(x) = (\log x_d\,, \sqrt{x_d}, \log x_{d+1} \ldots)$$

- Indicator function is often useful:

$$\phi_b(x) = I(L_b \leq x_d \leq U_b)$$

recall $I(A)$ is 1 if A is true
0 otherwise

sklearn.preprocessing.PolynomialFeatures

degree : *int or tuple (min_degree, max_degree), default=2*

If a single int is given, it specifies the maximal degree of the polynomial features. If a tuple `(min_degree, max_degree)` is passed, then `min_degree` is the minimum and `max_degree` is the maximum polynomial degree of the generated features. Note that `min_degree=0` and `min_degree=1` are equivalent as outputting the degree zero term is determined by `include_bias`.

interaction_only : *bool, default=False*

If `True`, only interaction features are produced: features that are products of at most `degree` *distinct* input features, i.e. terms with power of 2 or higher of the same input feature are excluded:

- included: `x[0]`, `x[1]`, `x[0] * x[1]`, etc.
- excluded: `x[0] ** 2`, `x[0] ** 2 * x[1]`, etc.

include_bias : *bool, default=True*

If `True` (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).

order : *{'C', 'F'}, default='C'*

Order of output array in the dense case. `'F'` order is faster to compute, but may slow down subsequent estimators.

Example: Polynomial Basis Functions

Create three two-dimensional data points [0,1], [2,3], [4,5]:

```
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
```

Compute quadratic features $(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$,

```
>>> poly = PolynomialFeatures(degree=2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

These are now our new data and ready to fit a model…

Example: Polynomial Basis Functions

Create a 3-rd order polynomial (cubic) function,

```
f = lambda x: (x-1)*(x-2)*(x-3)
import numpy.random as ra
ra.seed(20)
train_x = np.arange(5)
train_y = f(train_x) + 1*ra.randn(len(train_x))
train_y
```

✓ 0.3s

```
array([-5.11610689,  0.19586502,  0.35753652, -2.34326191,  4.91516741])
```

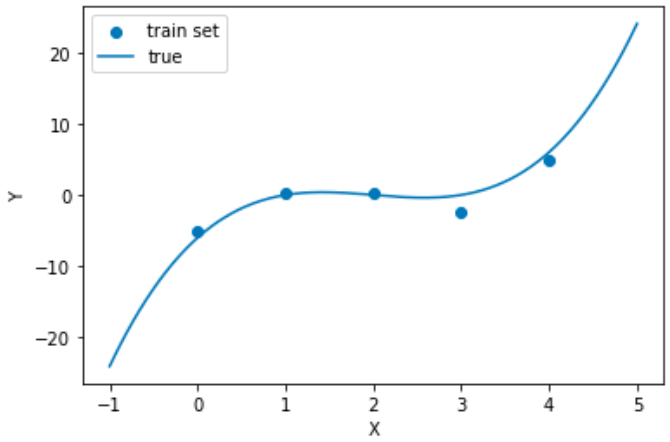
Plot train set and the actual function

```
test_x = np.linspace(-1,5,400)

from matplotlib import pyplot as plt
plt.scatter(train_x,train_y)

plt.plot(test_x, f(test_x))
plt.legend(['train set', 'true'])
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

✓ 0.4s



Example: Polynomial Basis Functions

Create cubic features $(1, x, x^2, x^3)$

```
poly = PolynomialFeatures(degree=3)
train_xx = poly.fit_transform(train_x[:,np.newaxis])
train_xx
```

✓ 0.4s

turns train_x (length 5 array) into a matrix (5 by 1 matrix)

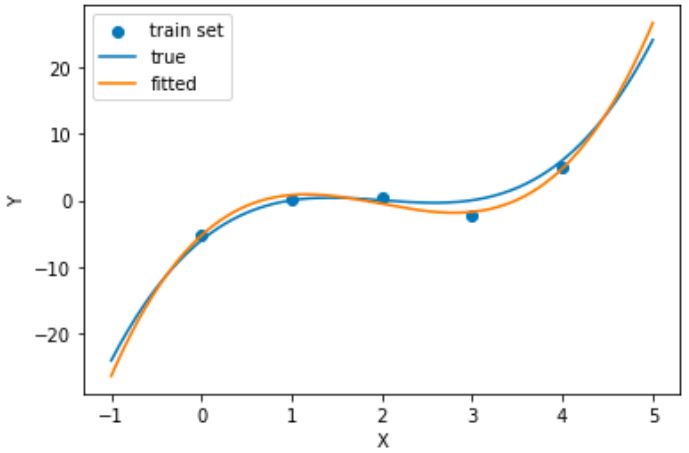
```
array([[ 1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  2.,  4.,  8.],
       [ 1.,  3.,  9., 27.],
       [ 1.,  4., 16., 64.]])
```

Perform linear regression; plot it

```
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(train_xx, train_y)
test_x = np.linspace(-1,5,400)
test_xx = poly.fit_transform(test_x[:,np.newaxis])
pred_y = model.predict(test_xx)

plt.scatter(train_x,train_y)
plt.plot(test_x, f(test_x))
plt.plot(test_x, pred_y)
plt.legend(['train set', 'true', 'fitted'])
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

✓ 0.2s



Recall the ordinary least squares solution is given by,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{m1} & \dots & x_{mD} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \qquad w^{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Design Matrix
(each training input on a column)

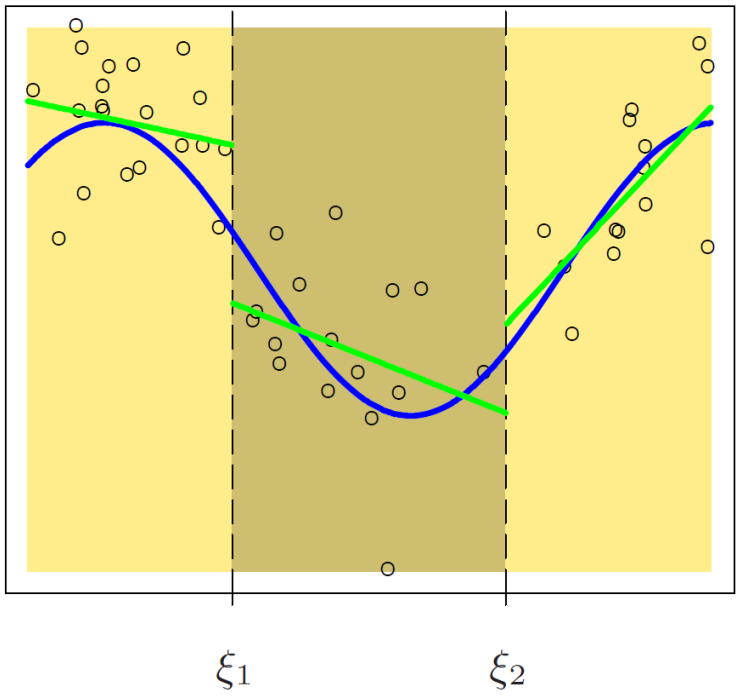
Vector of
Training labels

Can similarly solve in terms of basis functions,

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_1(x_1) & \dots & \phi_B(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_B(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_m) & \dots & \phi_B(x_m) \end{pmatrix} \qquad w^{\mathrm{OLS}} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{y}$$

Example: Piecewise Linear Regression

[Source: Hastie et al. (2001)]



Regression lines are discontinuous at boundary points

Decompose the input space into 3 regions with indicator basis functions,

$$\phi_1(x) = x \cdot I\{x < \xi_1\} \qquad \phi_4(x) = I\{x < \xi_1\}$$
$$\phi_2(x) = x \cdot I\{\xi_1 \leq x < \xi_2\} \qquad \phi_5(x) = I\{\xi_1 \leq x < \xi_2\}$$
$$\phi_3(x) = x \cdot I\{\xi_2 \leq x\} \qquad \phi_6(x) = I\{\xi_2 \leq x\}$$

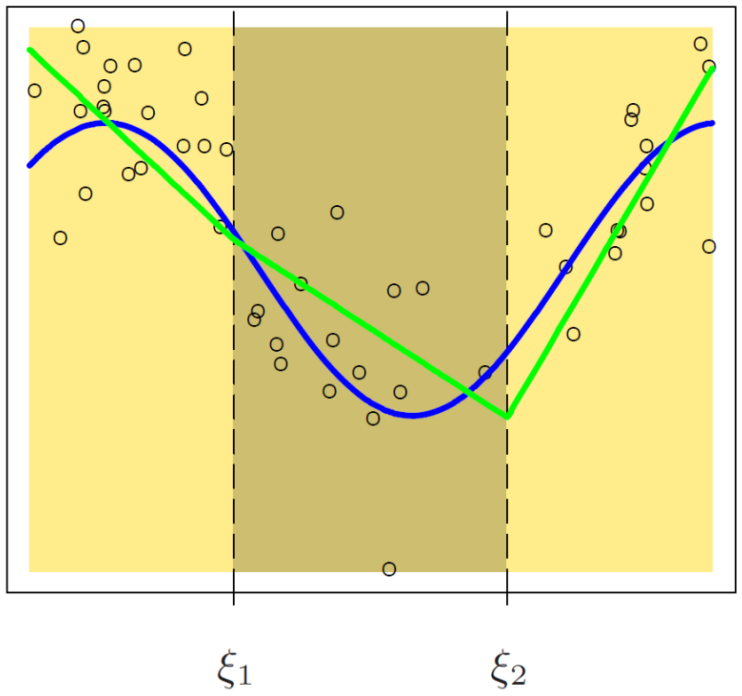
Fit linear regression model,

$$y = \sum_{i=1}^{6} w_i \phi_i(x)$$

Effectively fits 3 linear regressions independently to data in each region

Example: Piecewise Linear Regression

[Source: Hastie et al. (2001)]



Enforce constraint that lines agree at boundary points,

$$\phi_1(x) = 1$$

$$\phi_2(x) = x$$

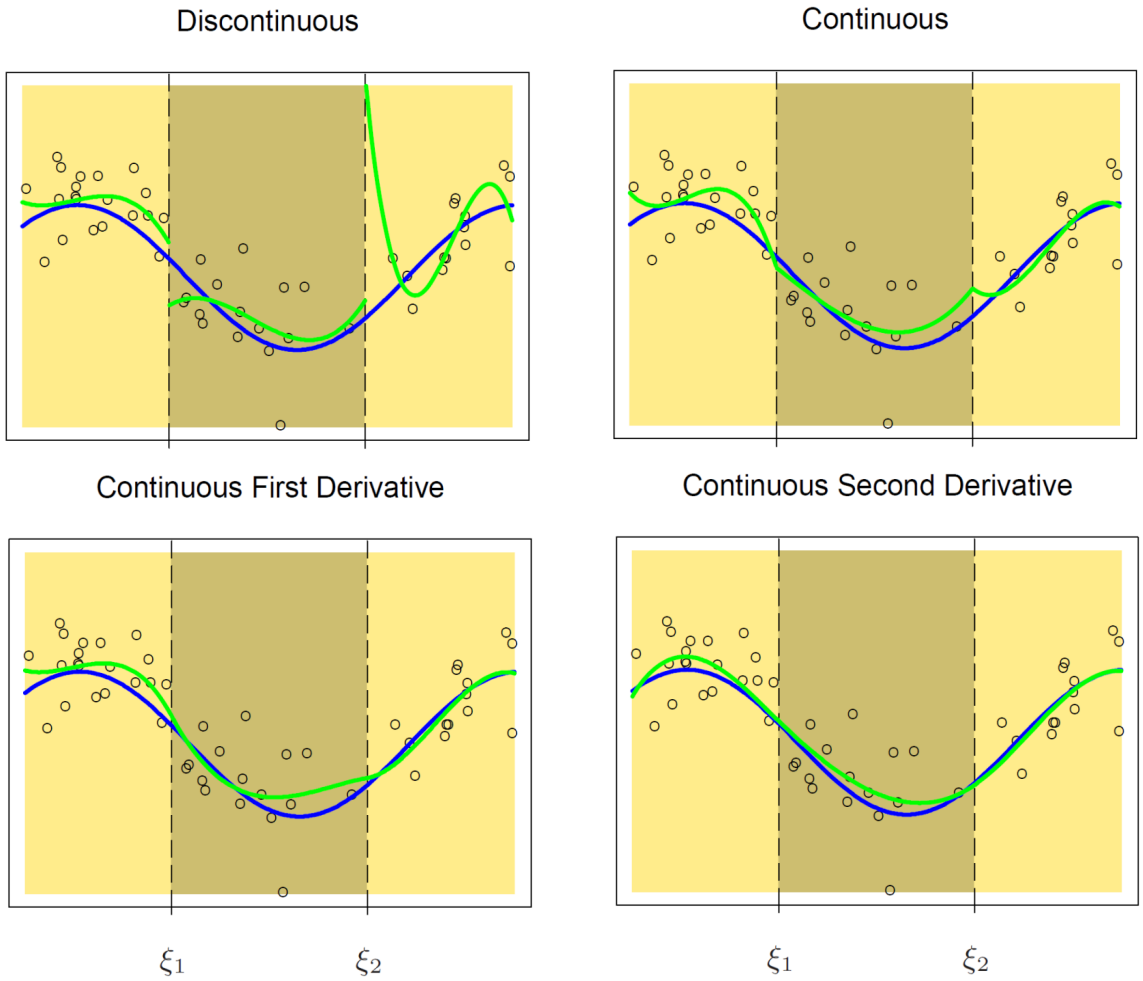
$$\phi_3(x) = (x - \xi_1)_+$$ <: activated only after $x \geq \xi_1$

$$\phi_4(x) = (x - \xi_2)_+$$ <: activated only after $x \geq \xi_2$

Where $(z)_+ := \max\{z, 0\}$.
I.e., the positive part of z

An improvement, but generally prefer *smoother* functions…

[Source: Hastie et al. (2001)]



Replace linear basis functions with polynomial,

$$\phi_1(x) = 1 \quad \phi_2(x) = x$$

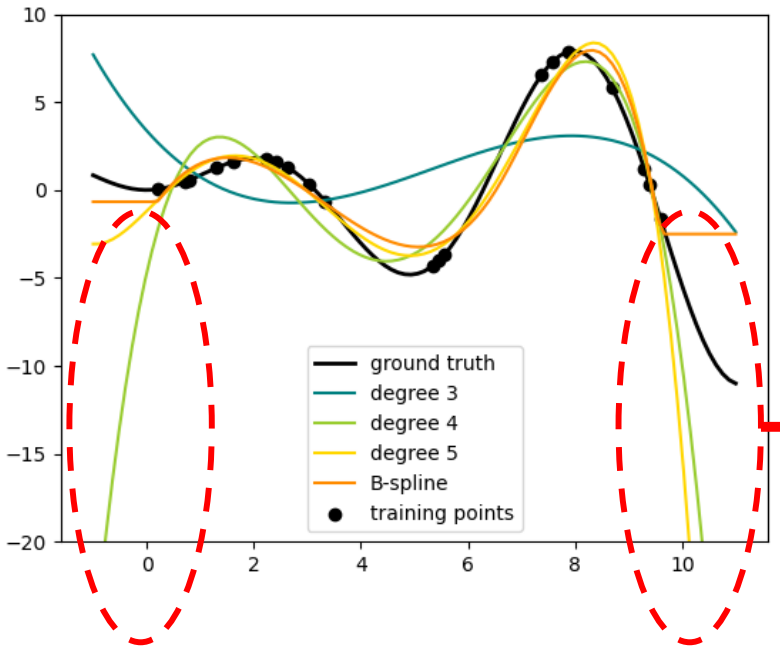
$$\phi_3(x) = x^2 \quad \phi_4(x) = x^3$$

$$\phi_5(x) = (x - \xi_1)_+^3$$

$$\phi_6(x) = (x - \xi_2)_+^3$$

Additional constraints ensure smooth 1st and 2nd derivatives at boundaries

Polynomial Splines



These piecewise regression functions are called *splines*

Supported in Scikit-Learn
`preprocessing.SplineTransformer`

Caution Polynomial basis functions often yield poor out-of-sample predictions with higher order producing more extreme predictions

Data Preprocessing

- Generally the first step in data science involves *preprocessing* or transforming data in some way
  - Filling in missing values (imputation)
  - Centering / normalizing / standardizing
  - Etc.
- We then fit our models to this preprocessed data
- One way to view preprocessing is simply as computing some basis function $\phi(x)$, nothing more

PROs

- More flexible modeling that is nonlinear in the original data
- Increases model expressivity

CONs

- Typically requires **more parameters** to be learned
- More sensitive to **overfitting** training data (due to expressivity)
- Requires more **regularization** to avoid overfitting
- Need to find *good* basis functions (feature engineering)

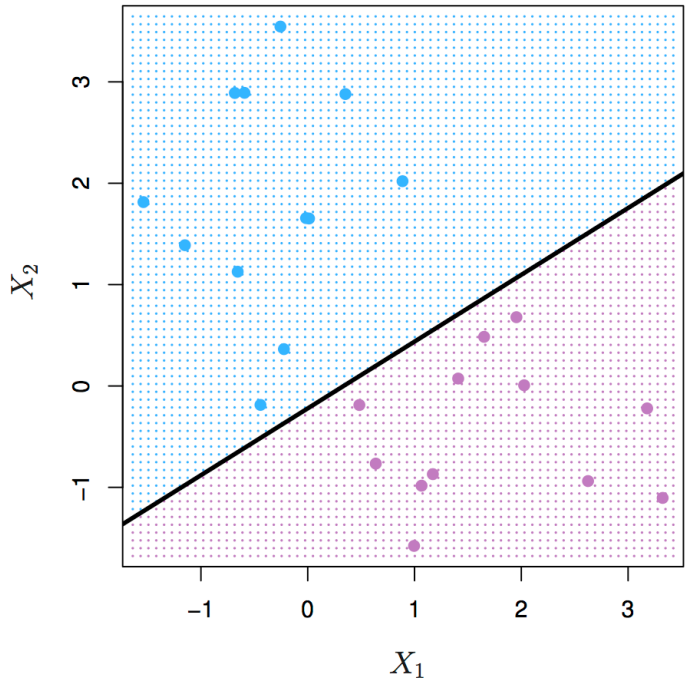
Outline

- Basis Functions
- Support Vector Machine Classifier
- Kernels
- Neural Networks

Linear Decision Boundary

Forget about the 'regression' point of view for now..

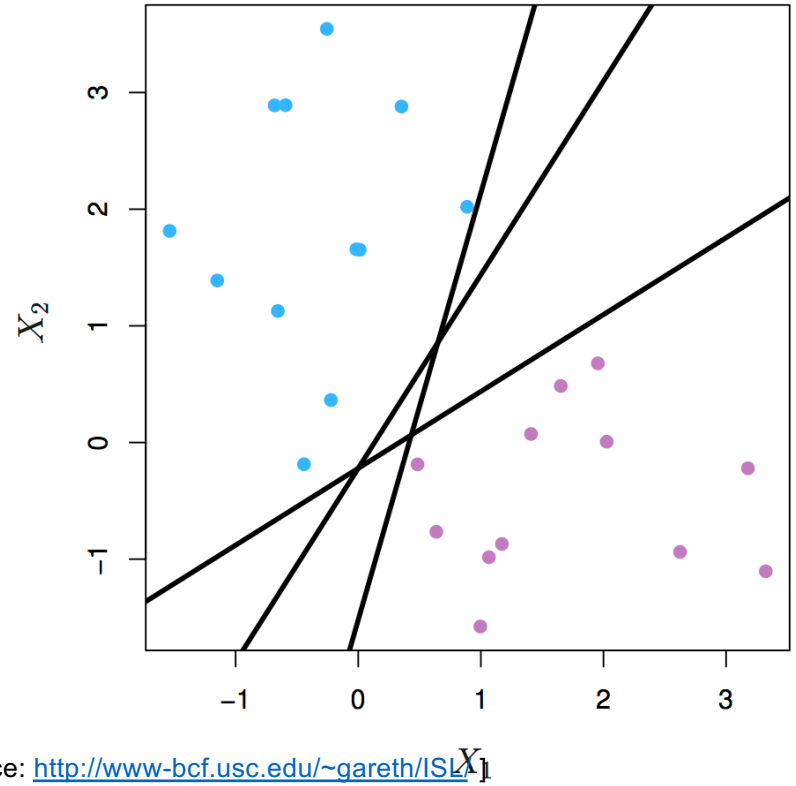
At the end of the day, we just want a line that separates the two classes well.



[Source: http://www-bcf.usc.edu/~gareth/ISL/]

Linear Decision Boundary

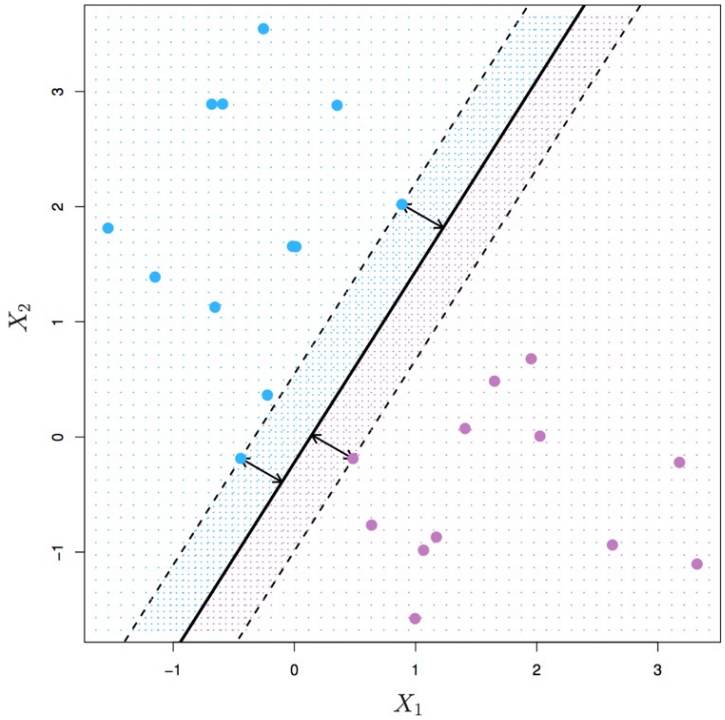
Note: Any boundary that separates classes is equivalently good on training data



Q: but if you have to choose one, which one will you choose?

[Source: http://www-bcf.usc.edu/~gareth/ISL]

Classifier Margin



*The **margin** measures minimum distance between each class and the decision boundary*

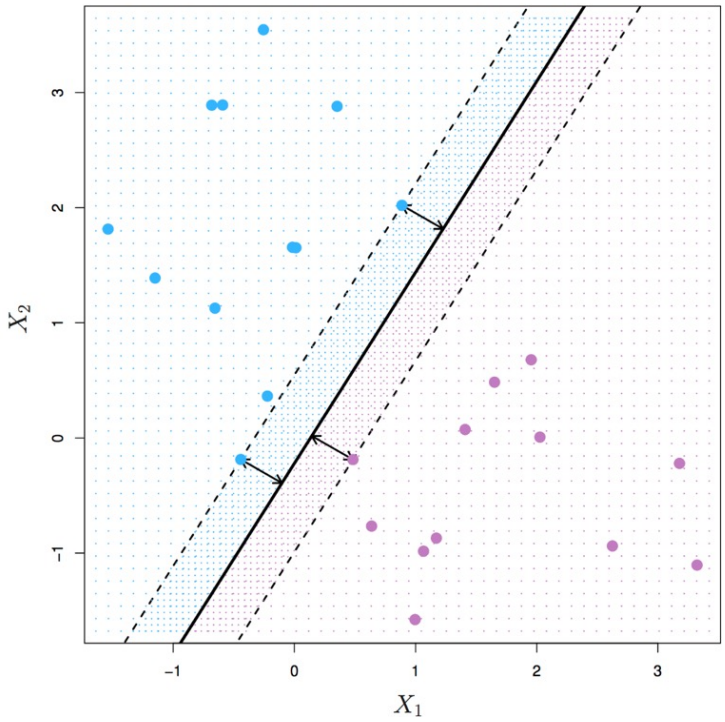
Observation Decision boundaries with larger margins are more likely to generalize to unseen data

Idea Learn the classifier with the largest margin that still separates the data…

…we call this a ***max-margin classifier***

[Source: http://www-bcf.usc.edu/~gareth/ISL/]

Max-Margin Classifier (Linear Seperable Case)



For now, let's focus on the case where the data is **linearly separable**

(Otherwise, there is no margin to talk about!)

[Source: http://www-bcf.usc.edu/~gareth/ISL/]

Max-Margin Classifier (Linear Separable Case)

Recall that the linear model is given by

$$f(x) = w^T x + b$$

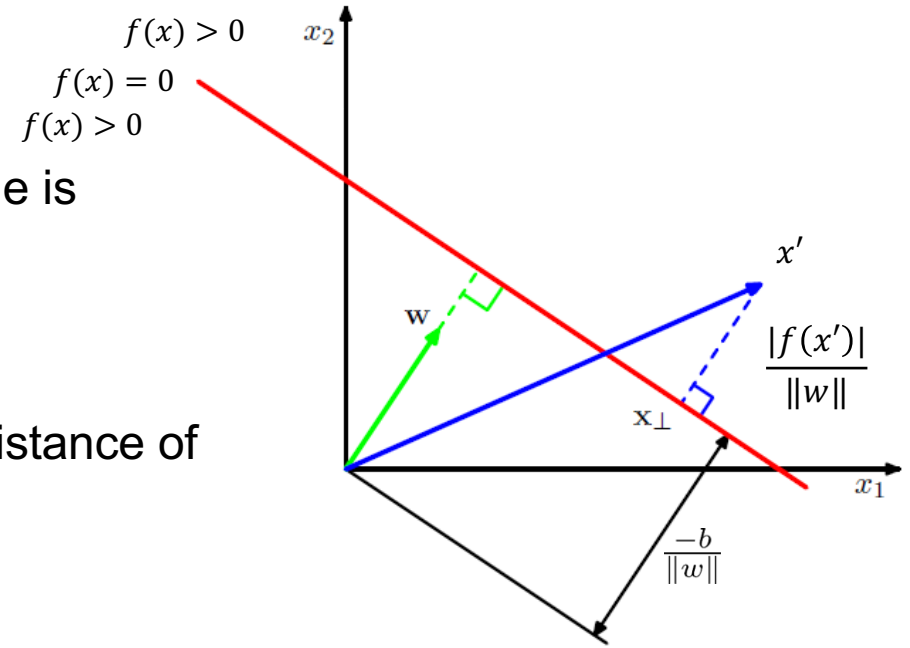
Let classes be $\{-1, 1\}$ so classification rule is

$$\text{Class} = \begin{cases} -1 & \text{if } f(x) < 0 \\ 1 & \text{if } f(x) >= 0 \end{cases}$$

Decision boundary is now at f(x) = 0 and distance of x' to the decision boundary (margin) is

$$\frac{|f(x')|}{\|w\|}$$

where $\|w\| = \sqrt{w^T w} = \sqrt{\sum_i w_i^2}$



Distance from a point to a plane equation:
wiki/Distance_from_a_point_to_a_plane

Max-Margin Classifier (Linear Separable Case)

For training data $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, a classifier $f(x) = w^\top x + b$ with 0 train error will satisfy

$$y^{(i)} f(x^{(i)}) = y^{(i)}(w^\top x^{(i)} + b) > 0$$

↓ negative margin when misclassifying it!

The margin for $(x^{(i)}, y^{(i)})$ is given by,

$$\frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

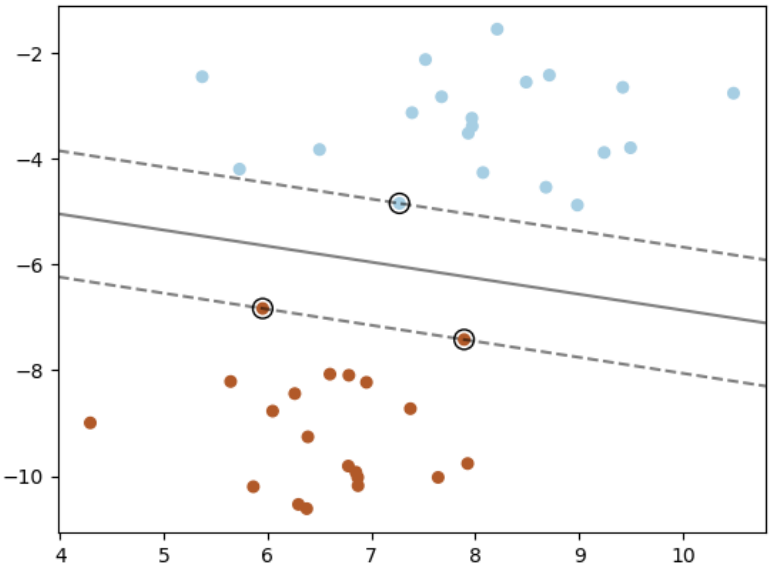
The margin of a classifier $f(x)$ is

$$\min_i \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

Find f that maximize margin

$$\arg\max_{w,b} \min_i \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

Max-Margin Classifier (Linear Separable Case)



Maximize the minimum margin

$$\arg\max_{w,b} \min_{i} \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

Minimum margin over all training data

Find the parameters (w,b) that **maximize** the **smallest margin** over all the training data

[Source: http://www-bcf.usc.edu/~gareth/ISL/]

Max-Margin Classifier (Linear Separable Case)

Learning objective is hard to solve in this form…

$$\arg\max_{w,b} \min_{i} \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

But we can scale parameters $w \to \alpha w$ and $b \to \alpha b$ without changing the margin

$\Rightarrow$ But then, there exists an infinitely many solution

$\Rightarrow$ Optimization packages will do a bad job.

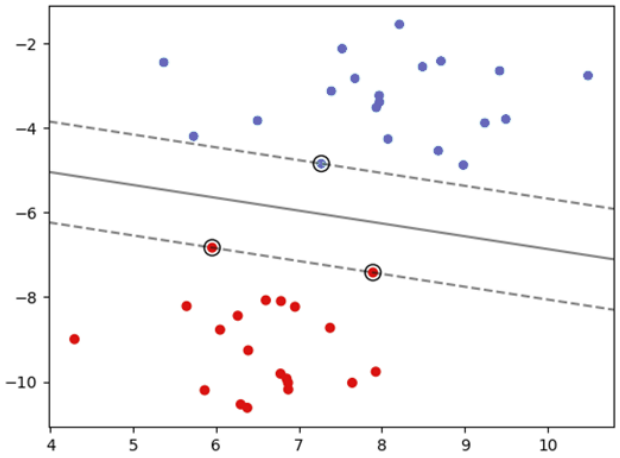
…we can pick the nearest point j to the margin and restrict $y^{(j)}\left(w^\top x^{(j)} + b\right) = 1$

This means that $y^{(i)}\left(w^\top x^{(i)} + b\right) \geq 1, \forall i$ $\Rightarrow \min_i y^{(i)}\left(w^\top x^{(i)} + b\right) = 1$!!!

We now just need to choose (w,b) that minimizes $\|w\|$ under this constraint!

Support Vector Machine (Hard Margin)

… it leads to



$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$\text{subject to}$$

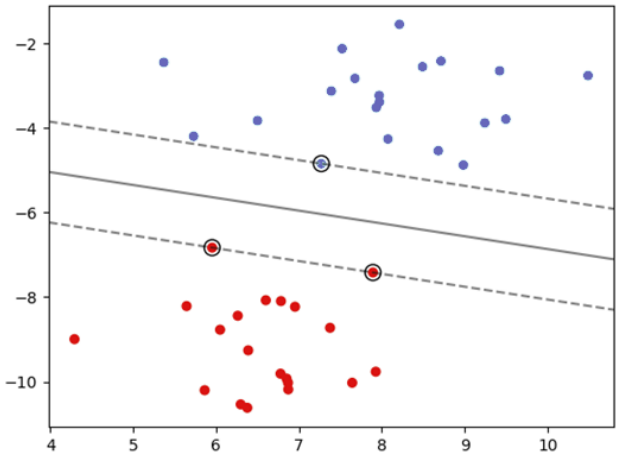
$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 \qquad \text{for } i = 1, \ldots, m$$

This is a convex (quadratic) optimization problem that can be solved efficiently

- Data are D-dimensional *vectors*
- Margins determined by nearest data points called *support vectors*
- We call this a *support vector machine* (SVM)

Support Vector Machine (Soft Margin)

If the data is linearly not separable,



$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{m} \xi_i$$

$$\text{subject to}$$

$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \quad \text{for } i = 1, \ldots, m$$

auxiliary 'slack' variable

C: tradeoff between margin and the slack!

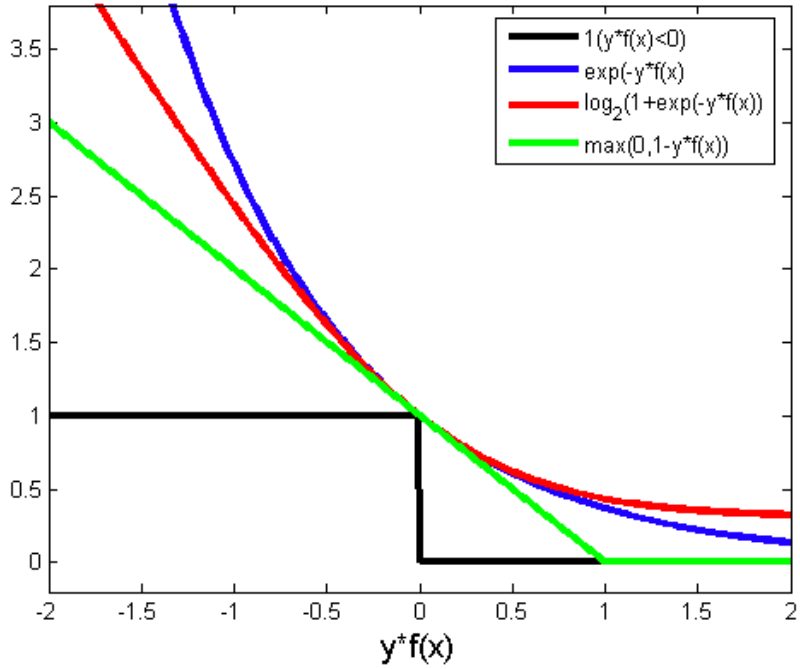
Support Vector Machine

$$\text{minimize } \frac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{m} \xi_i$$
$$\text{subject to}$$
$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0 \quad \text{for } i = 1, \ldots, m$$

Equivalent formulation

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{m} \left(1 - y^{(i)}(w^\top x^{(i)} + b)\right)_+$$

$$\ell\left(f; x^{(i)}, y^{(i)}\right) = \left(1 - y^{(i)} f\left(x^{(i)}\right)\right)_+$$



$$(X)_+ := \max(X, 0)$$

General Principle

$$\arg\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\left(1 - y^{(i)}(w^\top x^{(i)} + b)\right)_+$$

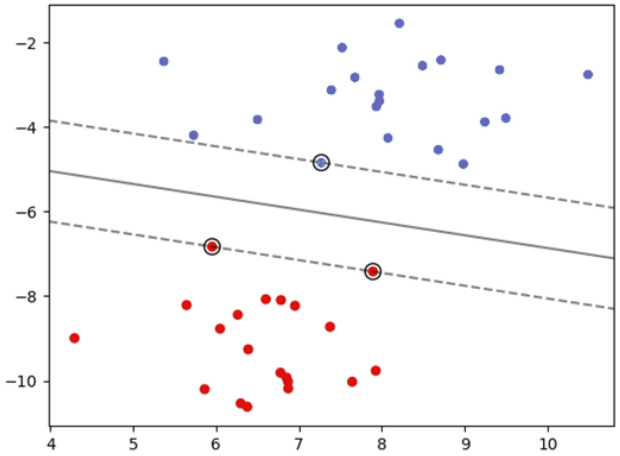
=> by setting $C = 1/\lambda$, it's equivalent to solve

$$\arg\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{m}\left(1 - y^{(i)}(w^\top x^{(i)} + b)\right)_+$$

SVM belongs to the general loss-oriented formulation!

$$\text{Model} = \arg\min_{\text{model}} \text{Loss}(\text{Model}, \text{Data}) + \lambda \cdot \text{Regularizer}(\text{Model})$$

Support Vectors



$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{m} \xi_i$$

$$\text{subject to}$$

$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0 \quad \text{for } i = 1, \ldots, m$$

Those data points achieving equality $y^{(i)}\left(w^\top x^{(i)} + b\right) = 1 - \xi_i$ are called **support vectors**.

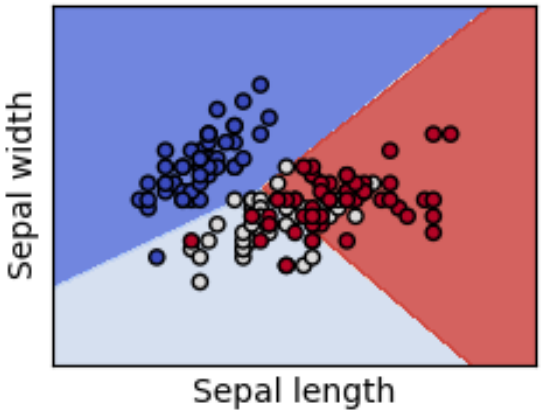
Turns out, if you knew support vectors already, solving the optimization problem above with **just the support vectors as train set** leads to the same solution.

⇒ Leave-one-out cross validation can be done fast!

SVM in Scikit-Learn

SVM with linear decision boundaries,

`sklearn.svm.LinearSVC`



Call options include…

penalty : *{'l1', 'l2'}, default='l2'*

Specifies the norm used in the penalization. The 'l2' penalty is the standard used in SVC. The 'l1' leads to `coef_` vectors that are sparse.

dual : *bool, default=True*

Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when n_samples > n_features.

C : *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.

Only showing linear for a reason that will be clear soon…

Other options for controlling optimizer (e.g. convergence tolerance 'tol')