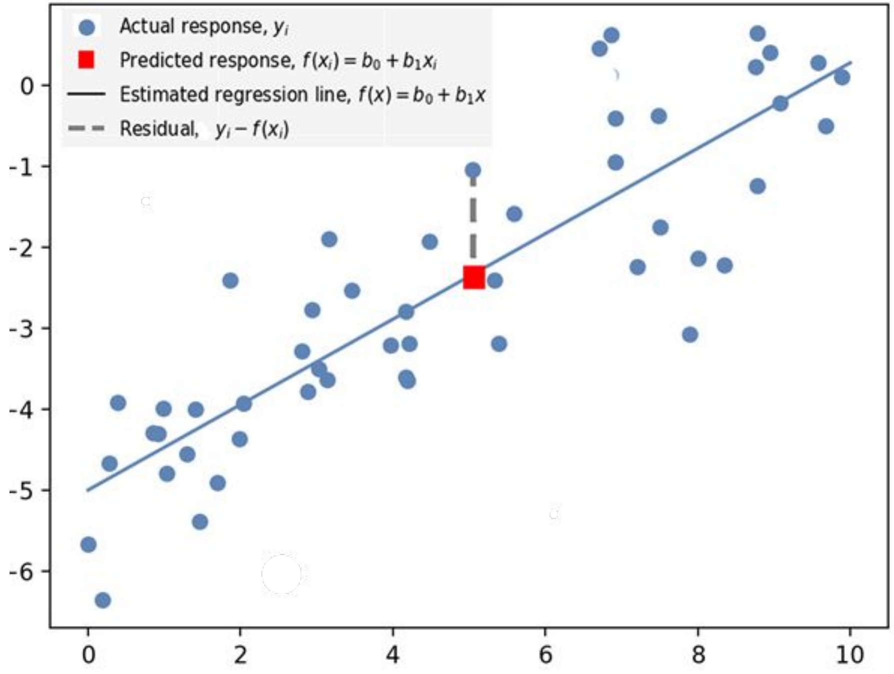
Computer Science

CSC380: Principles of Data Science

Linear Models 2

Xinchen Yu

Review: 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/

Review: 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

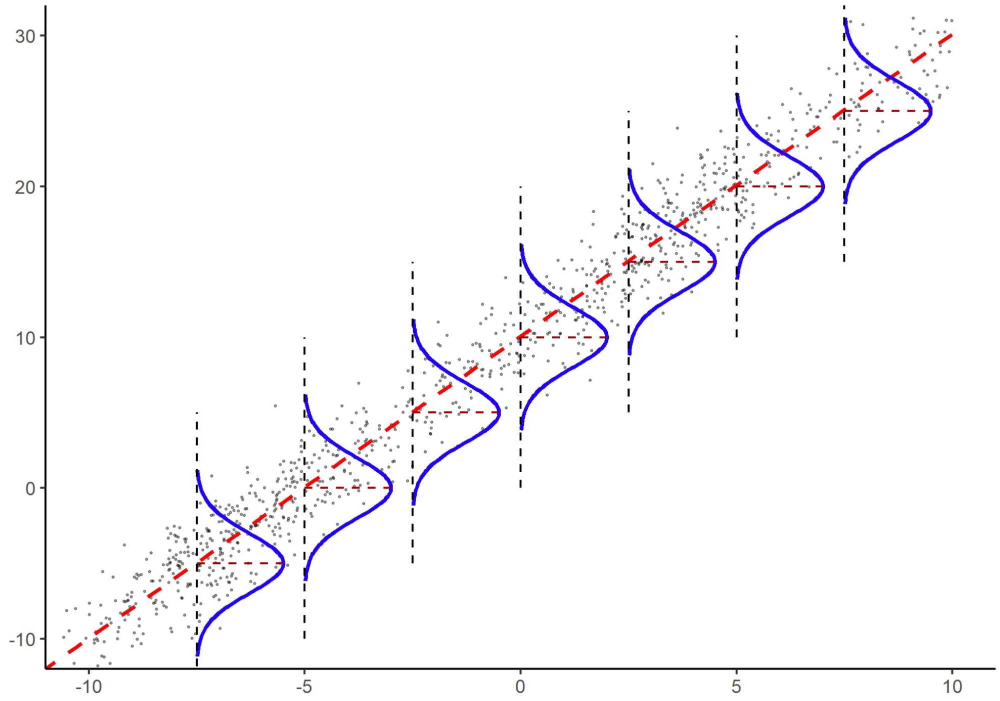
$$w = \frac{\sum_i y^{(i)} x^{(i)}}{\sum_j (x^{(j)})^2}$$

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

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

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,

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

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!

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

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

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

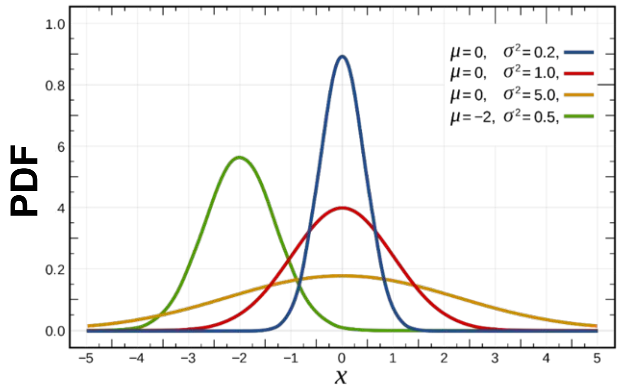
note model assumption! $p(y|x;w) = \mathcal{N}(w^T x, \sigma^2)$

Univariate Gaussian (Normal) Distribution

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

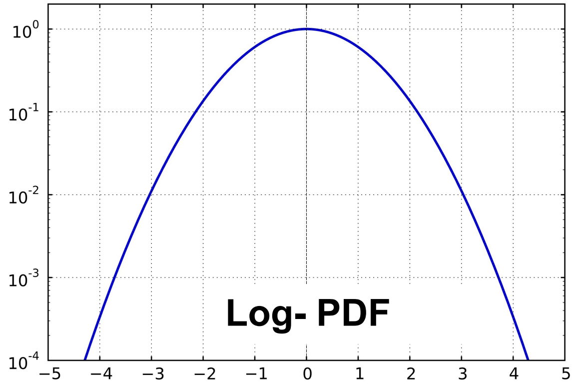
Gaussian (a.k.a. Normal) distribution with mean (location) μ and variance (squared scale) σ^2 parameters,

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

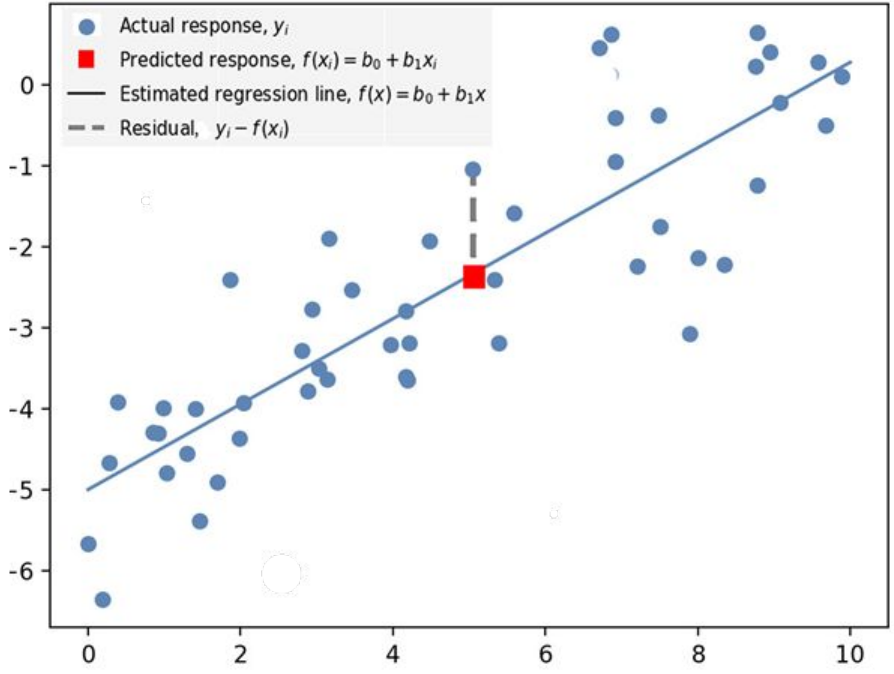


The logarithm of the PDF if just a negative quadratic,

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



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,

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

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 is the issue?

May be non-invertible!

Invertible matrix

Invertible matrix: a matrix A of dimension n x n is called invertible if and only if there exists another matrix B of the same dimension, such that AB = BA = I, where I is the identity matrix of the same order.

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix}$$

$$AB = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$BA = \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

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 non-invertible 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

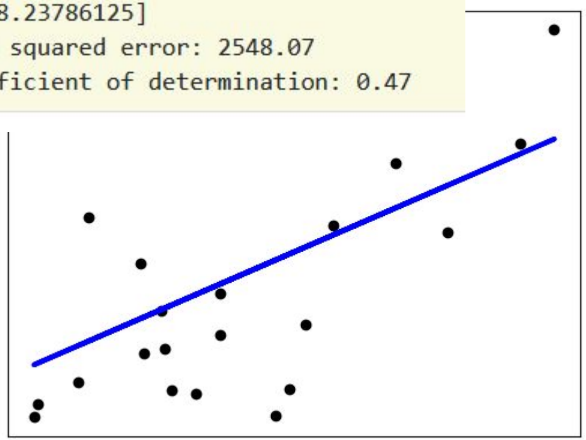
Train (fit) and predict,

```
# 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

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

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)

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

Q: 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

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

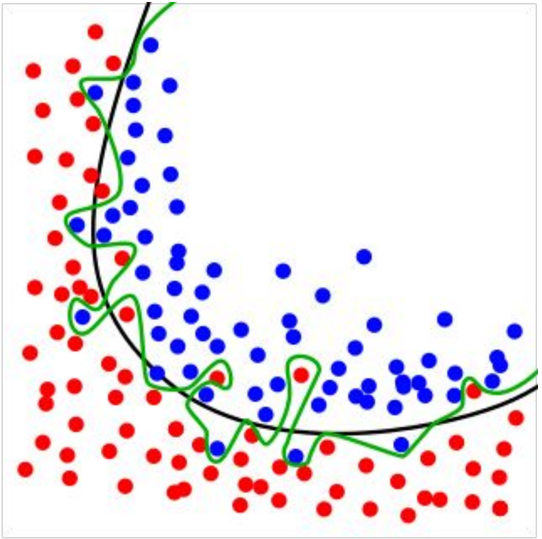
Challenges in ML

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)

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

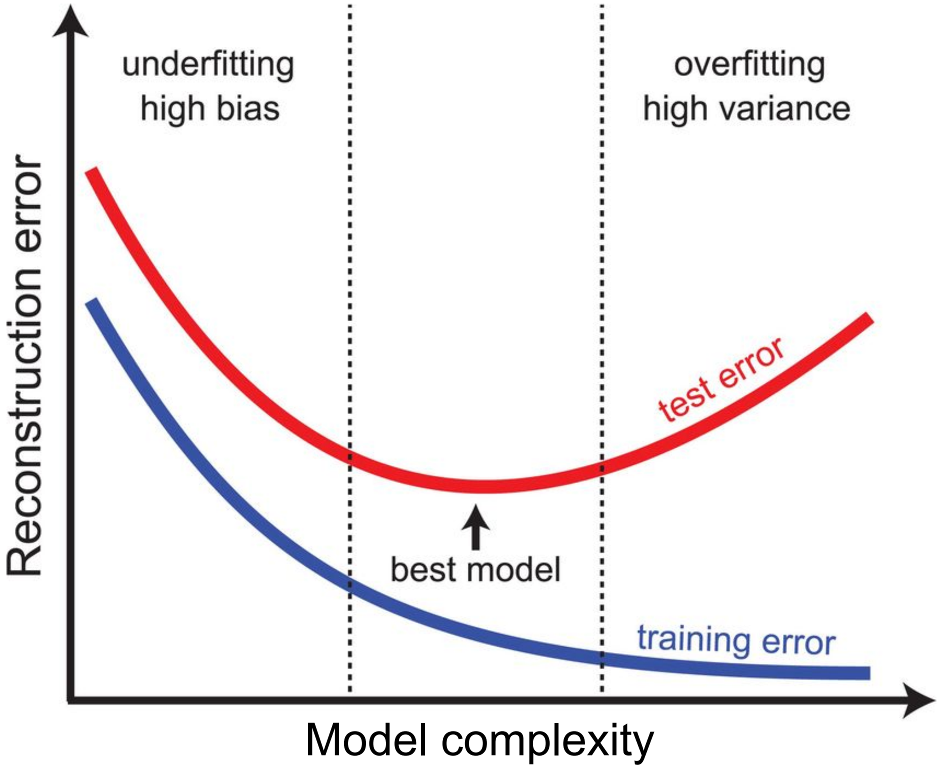
Doesn't generalize to unseen data – called *overfitting* the training data.

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



green: almost memorization
black: true decision boundary

Bias-Variance Tradeoff



Regularization

- 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)!

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

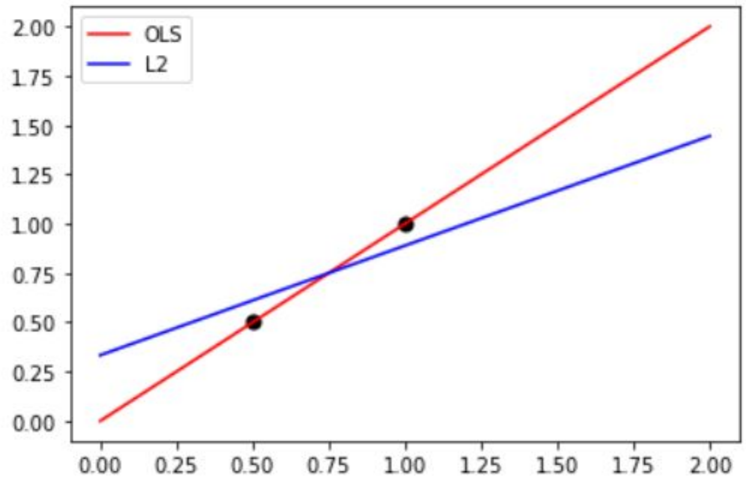
alpha : *{float, ndarray of shape (n_targets,)}, default=1.0*

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

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,

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

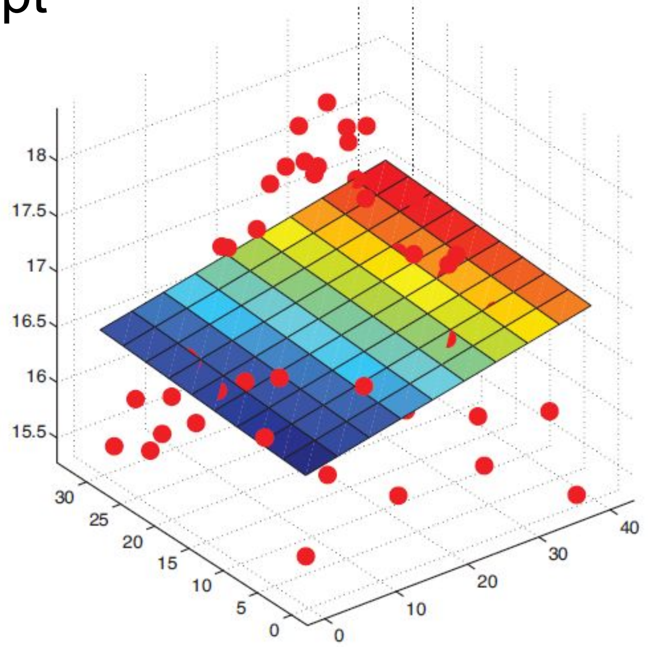
Recall: we enforced $x_D^{(i)} = 1$ so that w_D encodes the intercept

  - Penalizing intercept will make solution depend on origin for Y. i.e., add a constant c to $y^{(i)}$'s $\Longrightarrow$ the solutions changes!

Moving to higher dimensions…

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

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



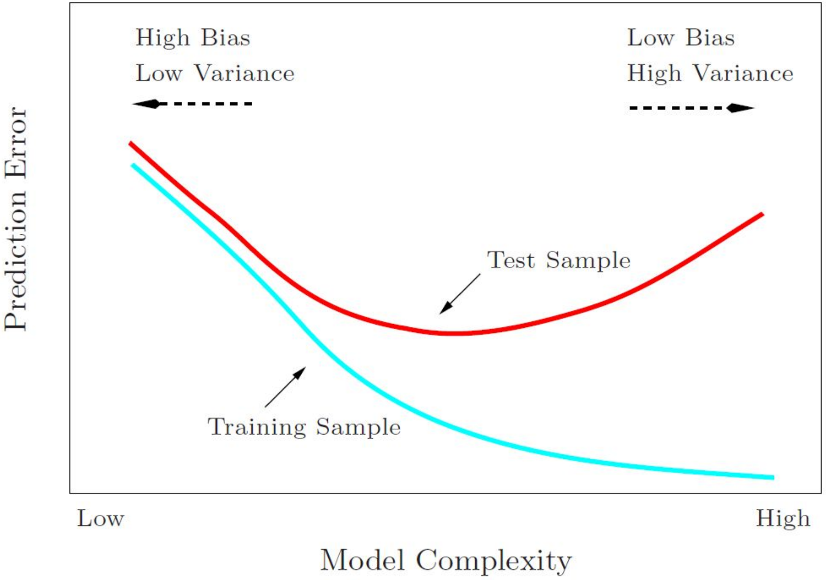
Since:

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

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!