Supervised learning

labeled data

- Historical data with labels
- Experiments to get labeled data A-B test
- Crowd-sourcing labeled data

in Python scikit-learn/sklearn TensorFlow keras

Iris dataset

```
from sklearn import datasets
    iris = datasets.load_iris()
    print(iris.keys())
```

```
type(iris)
```

```
sklearn.datasets.base.Bunch
```

```
dict_keys(['data', 'target_names', 'DESCR', 'feature_names', 'target'])
```

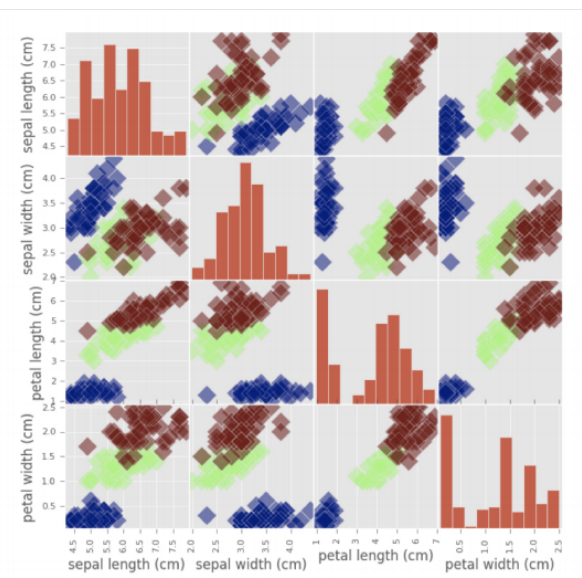
EDA)

```
X = iris.data
y = iris.target
df = pd.DataFrame(X, columns=iris.feature_names)
print(df.head())
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Visual EDA

```
_ = pd.plotting.scatter_matrix(df, c = y, figsize = [8, 8],  s=150,
marker = 'D')
```

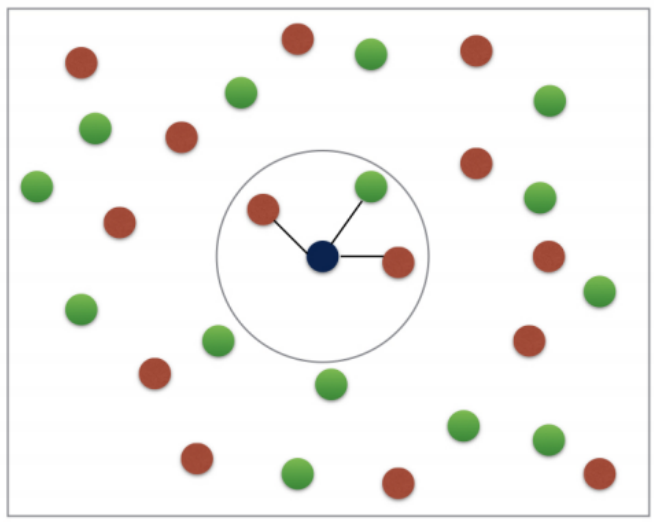


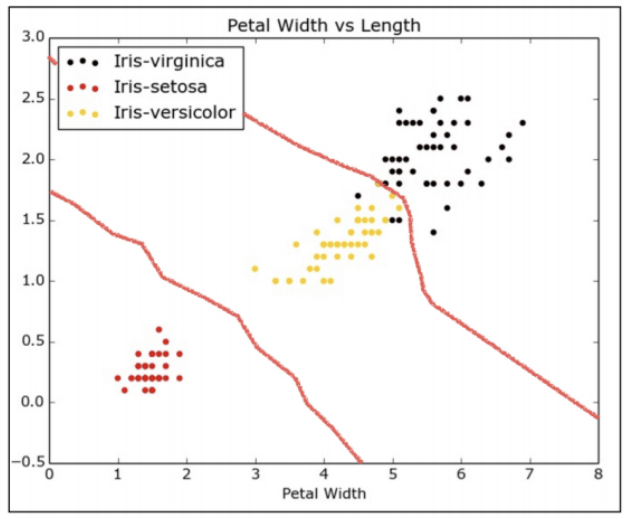
k-Nearest Neighbors

Basic idea: Predict the label of a data point by

Looking at the ‘k’ closest labeled data points

Intuition





.fit()

predict()

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(iris['data'], iris['target'])
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30,
   metric='minkowski',metric_params=None, n_jobs=1,
   n_neighbors=6, p=2,weights='uniform')
```

```
prediction = knn.predict(X_new)
```

```
Prediction: [1 1 0]
```

model performance

In classi×cation, accuracy is a commonly used metric

Training

generalize

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.3,
                     random_state=21, stratify=y)
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(\"Test set predictions:\\n {}\".format(y_pred))
```

```
knn.score(X_test, y_test)
```

.score(X,y) is accuracy

Python classes

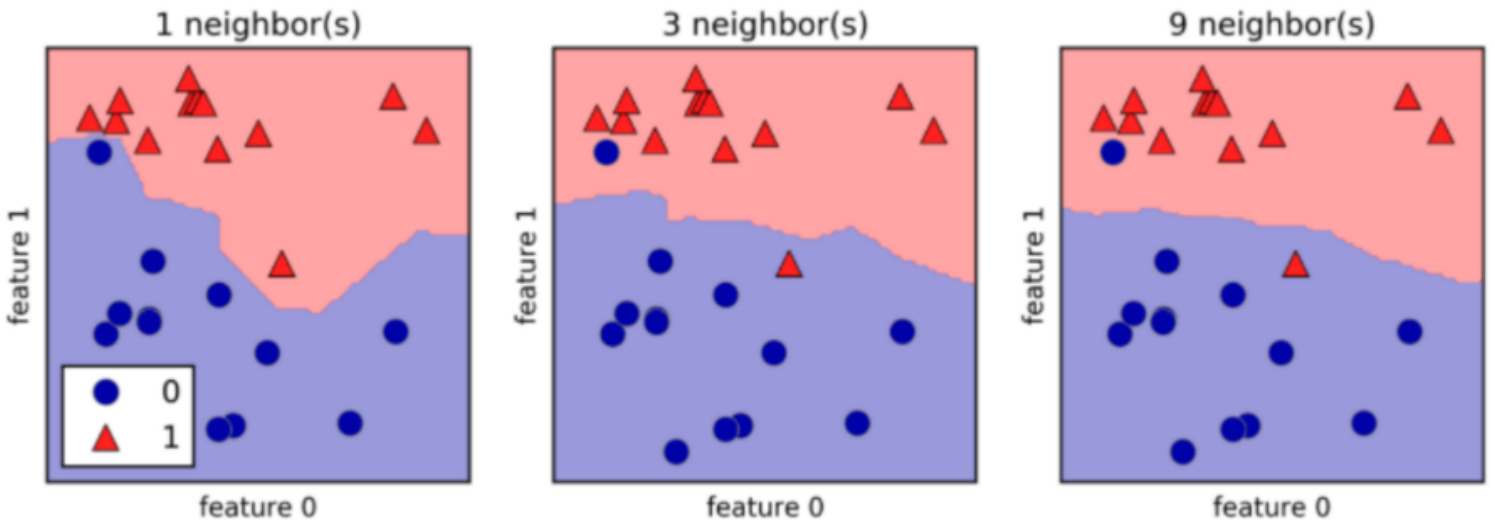
All machine learning models implemented as Python classes

They implement the algorithms for learning and predicting

Store the information learned from the data

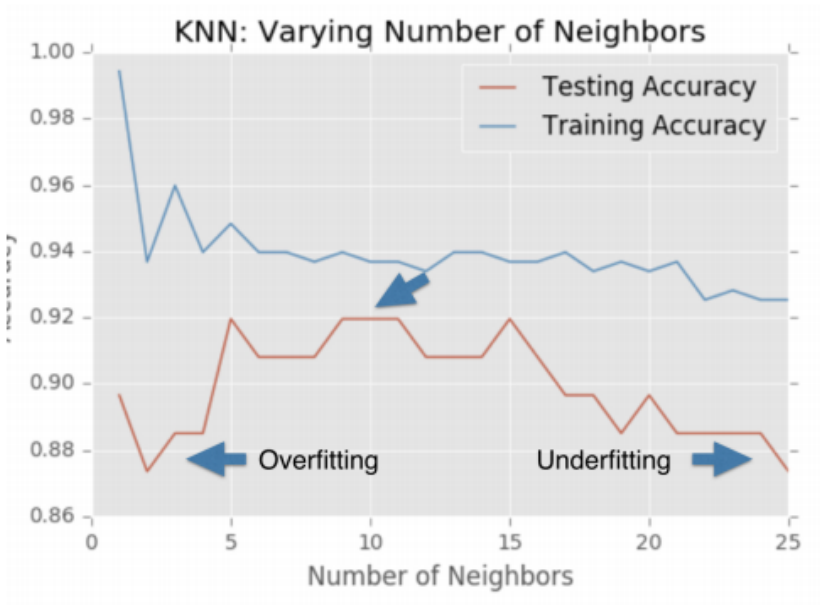
k-NN

Model complexity



Larger k = smoother decision boundary = less complex model

Smaller k = more complex model = can lead to over×tting



Develop k-Nearest Neighbors in Python From Scratch

3 parts

- **Step 1**: Calculate Euclidean Distance.
- **Step 2**: Get Nearest Neighbors.
- **Step 3**: Make Predictions.

```python
# Example of making predictions
from math import sqrt

# calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return sqrt(distance)

# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Make a classification prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

# Test distance function
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]
prediction = predict_classification(dataset, dataset[0], 3)
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

```
Expected 0, Got 0.
```

```python
def distance(instance1, instance2):
    # just in case, if the instances are lists or tuples:
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

    return np.linalg.norm(instance1 - instance2)
```

```python
def get_neighbors(training_set,
                  labels,
                  test_instance,
                  k,
                  distance=distance):
    """
    get_neighors calculates a list of the k nearest neighbors
    of an instance 'test_instance'.
    The list neighbors contains 3-tuples with
    (index, dist, label)
    where
    index    is the index from the training_set,
    dist     is the distance between the test_instance and the
             instance training_set[index]
    distance is a reference to a function used to calculate the
             distances
    """
    distances = []
    for index in range(len(training_set)):
        dist = distance(test_instance, training_set[index])
        distances.append((training_set[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return neighbors
```

```python
from collections import Counter

def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]
```

```python
def vote_prob(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    labels, votes = zip(*class_counter.most_common())
    winner = class_counter.most_common(1)[0][0]
    votes4winner = class_counter.most_common(1)[0][1]
    return winner, votes4winner/sum(votes)
```

USING SKLEARN FOR KNN

KNeighborsClassifier

RadiusNeighborsClassifier

Parameters

- n_neighbors
- weights
- algorithm
- leaf_size
- metric
- metric_params
- n_jobs

```
# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(learnset_data, learnset_labels)
KNeighborsClassifier(algorithm='auto',
                     leaf_size=30,
                     metric='minkowski',
                     metric_params=None,
                     n_jobs=1,
                     n_neighbors=5,
                     p=2,
                     weights='uniform')

print("Predictions form the classifier:")
print(knn.predict(testset_data))
print("Target values:")
print(testset_labels)
```

```
Predictions form the classifier:
[1 2 1 1 2 2 0 1 1 0 1 2]
Target values:
[1 2 1 1 2 2 0 1 2 0 1 2]
```

regression

Linear Regression

For a collection of data points **X**, the predictions yˆ can be expressed via the matrix-vector product:

Loss Function

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2.$$

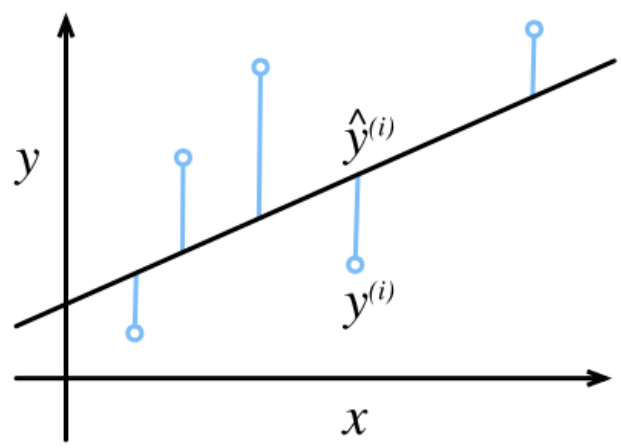


Fig. 3.1.1: Fit data with a linear model.

$$\mathbf{w}^*, b^* = \underset{\mathbf{w},b}{\operatorname{argmin}}\ L(\mathbf{w}, b).$$

Analytic Solution

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y.$$

Gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \partial_{\mathbf{w}} l^{(i)}(\mathbf{w}, b) \quad = w - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \mathbf{x}^{(i)}\left(\mathbf{w}^\top\mathbf{x}^{(i)} + b - y^{(i)}\right),$$

$$b \leftarrow b - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \partial_b l^{(i)}(\mathbf{w}, b) \quad = b - \frac{\eta}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \left(\mathbf{w}^\top\mathbf{x}^{(i)} + b - y^{(i)}\right).$$

B| represents the number of examples in each minibatch (the *batch size*)

η denotes the *learning rate*.

hyper-parameters.

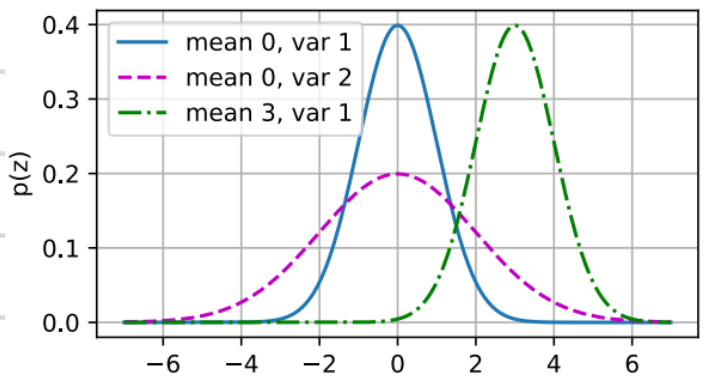
We emphasize that the values of the batch size and learning rate are manually pre-specified and not typically learned through model training.

These parameters that are tunable but not updated in the training loop are called *hyper-parameters*. *Hyperparameter tuning* is the process by which these are chosen, and typically requires that we adjust the hyperparameters based on the results of the inner (training) loop as assessed on a separate *validation* split of the data.

Normal Distribution and Squared Loss

square loss objective

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(z-\mu)^2\right).$$



noise is

assume that observations arise from noisy observations, where the noise is normally distributed

然而, 现实任务中 $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 往往不是满秩矩阵. 例如在许多任务中我们会遇到大量的变量, 其数目甚至超过样例数, 导致 $\mathbf{X}$ 的列数多于行数, $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 显然不满秩. 此时可解出多个 $\hat{\boldsymbol{w}}$, 它们都能使均方误差最小化. 选择哪一个解作为输出将由学习算法的归纳偏好决定, 常见的做法是引入正则化 (regularization)项.

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

likelihood

seeing a particular *y* for a given **x** via

$$p(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x} - b)^2\right).$$

maximum likelihood principle,

$$P(Y \mid X) = \prod_{i=1}^{n} p(y^{(i)}|\mathbf{x}^{(i)}).$$

$$-\log p(\mathbf{y}|\mathbf{X}) = \sum_{i=1}^{n} \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2.$$

σ is some fixed constant

second term is identical to the squared error objective introduced earlier, but for the multiplicative constant 1σ2

minimizing squared error is equivalent to maximum likelihood estimation of a linear model under the assumption of additive Gaussian noise.

Backpropagation

5. Linear regression -线性代数

- The solution to linear regression that we just provided, $\theta = (X^T X)^{-1} X^T Y$, is called the *closed-form* solution to the problem.
  - We are able to use mathematics to come up with a direct solution to the minima/maxima problem.
- However, for some problems a closed-form solution/equation may not exist and/or if our matrix is large it may become infeasible to compute inverse
  - Or inverse may not exist in some cases

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

300000 × 60000

Same goal: minimize J

closed form: 方程 minima

$\frac{\partial J}{\partial \theta} = -2X^T Y + 2X^T X\theta = 0$

$\theta = (X^T X)^{-1} X^T Y$

Find θ

Gradient descent:

$\frac{\partial J}{\partial \theta} = 2X^T(X\theta - y)$

$\theta = \theta - 2X^T(X\theta - y)$ → Find θ

Gradient Descent

- Termination Criteria:
  - Max number of iterations reached
  - Parameters change very little
  - Change in the training error is very little.

1. How is loss changing with weights
2. Using 1 to minimize loss...

Least square estimate

55

Least Square Estimate

$\|a\|^2 \Rightarrow a^T a$

- So now we want to minimize:

$$J = \sum_{i=1}^{N} (Y_i - X_i\theta)^2$$

- Which we can write in matrix form as

$$J = (Y - X\theta)^T (Y - X\theta)$$

$\theta = (X^T X)^{-1} X^T Y$

- How can we find the minimum of this?
  - Take the derivative with respect to θ, set it equal to zero, and solve for θ!

$$\frac{\partial J}{\partial \theta} = 0$$

closed form

$\frac{\partial J}{\partial \theta}$

Least square gradient descent

2

Least Squares Gradient Descent

$J = (y - x\theta)^2$

- Now let's take the gradient of this with respect to one of the parameters, θ_i

$$\frac{\partial J}{\partial \theta_i} = -2x_i y + 2x_i x\theta = 2x_i(x\theta - y)$$

- We can then vectorize this to get all the gradients at once for this observation:

$$\frac{\partial J}{\partial \theta} = 2x^T(x\theta - y) \Rightarrow 2x^T x\theta - 2x^T y = 0$$

$\theta = (X^T X)^{-1} X^T Y$ $X^T X\theta = X^T y$

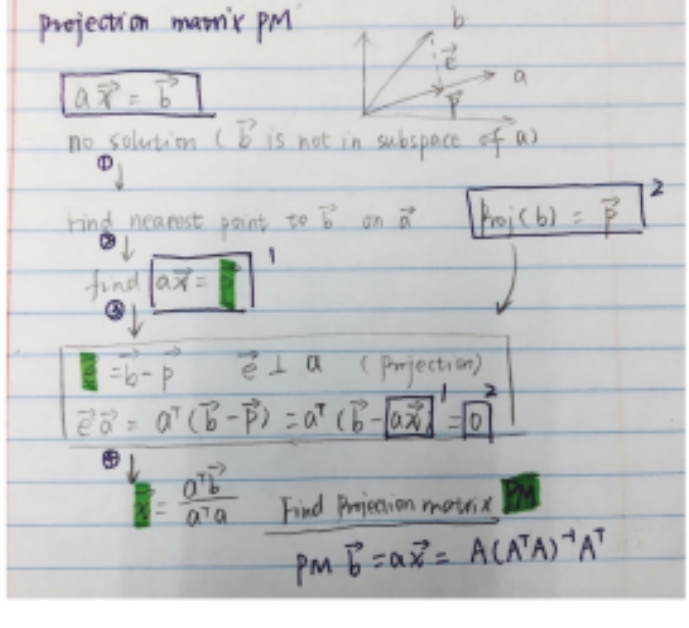
$\theta = (X^T X)^{-1} X^T y$

- And update your parameters as:

$$\theta = \theta - 2x^T(x\theta - y)$$

Closed form- analytical solution

- $J = (Y - X\theta)^T (Y - X\theta)$
- $J = (Y^T - (X\theta)^T)(Y - X\theta)$ //distribution of transpose
- $J = (Y^T - \theta^T X^T)(Y - X\theta)$ //distribution of transpose
- $J = Y^T Y - Y^T X\theta - \theta^T X^T Y + \theta^T X^T X\theta$//distribution
- $\frac{dJ}{d\theta} = -(Y^T X)^T - (X^T Y) + 2X^T X\theta = 0$ //derivative
- $\Rightarrow -X^T Y - X^T Y + 2X^T X\theta = 0$ //simplification
- $\Rightarrow -2X^T Y + 2X^T X\theta = 0$ //simplification
- $\Rightarrow X^T X\theta = X^T Y$ //algebra
- $\theta = (X^T X)^{-1} X^T Y$ //algebra



Regularized regression

Why regularize

Recall: Linear regression minimizes a loss function

It chooses a coefficient for each feature variable

Large coefficients can lead to overfitting

Penalizing large coefficients: Regularization

Ridge regression

Loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} {a_i}^2$

```
from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, random_state=42)
ridge = Ridge(alpha=0.1, normalize=True)
ridge.fit(X_train, y_train)
ridge_pred = ridge.predict(X_test)
ridge.score(X_test, y_test)
```

Lasso regression

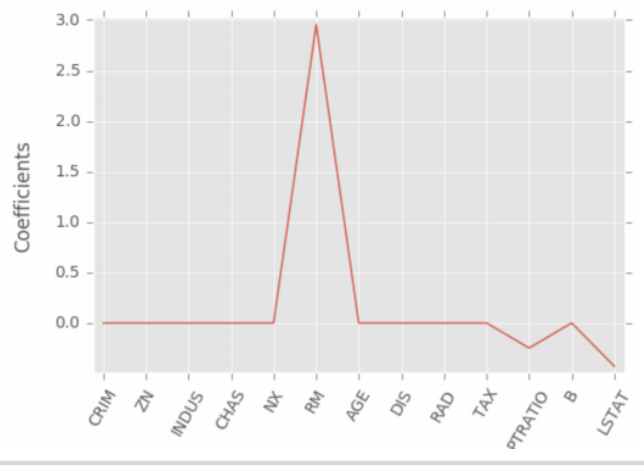
Loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} |a_i|$

```
from sklearn.linear_model import Lasso
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, random_state=42)
lasso = Lasso(alpha=0.1, normalize=True)
lasso.fit(X_train, y_train)
lasso_pred = lasso.predict(X_test)
lasso.score(X_test, y_test)
```

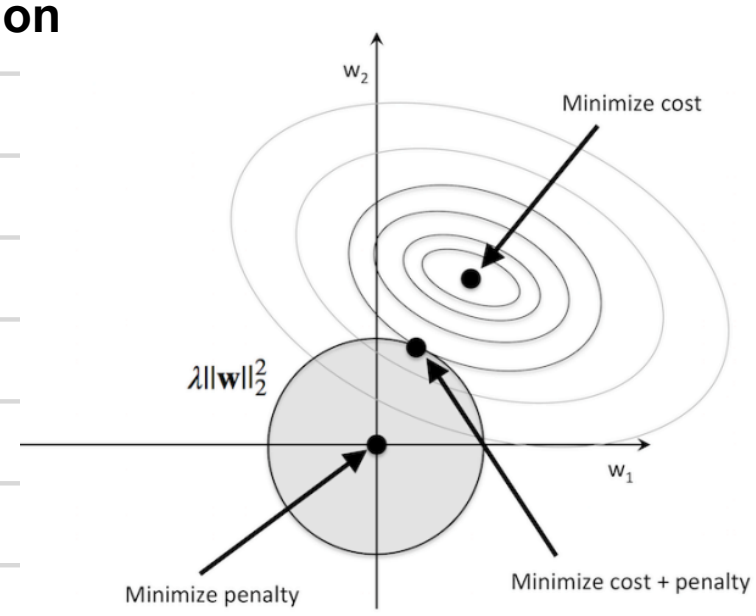
Can be used to select important features of a dataset

Shrinks the coef×cients of less important features to exactly 0

```
from sklearn.linear_model import Lasso
names = boston.drop('MEDV', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
_ = plt.plot(range(len(names)), lasso_coef)
_ = plt.xticks(range(len(names)), names, rotation=60)
_ = plt.ylabel('Coefficients')
plt.show()
```



L2 regularization

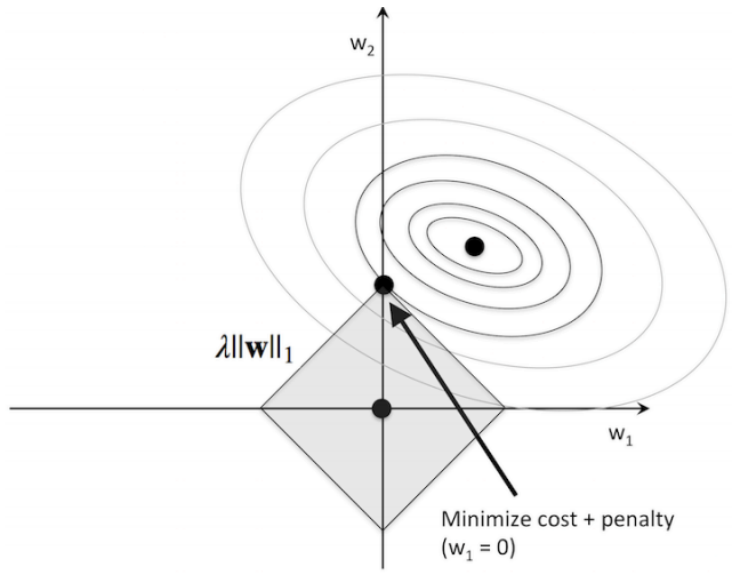


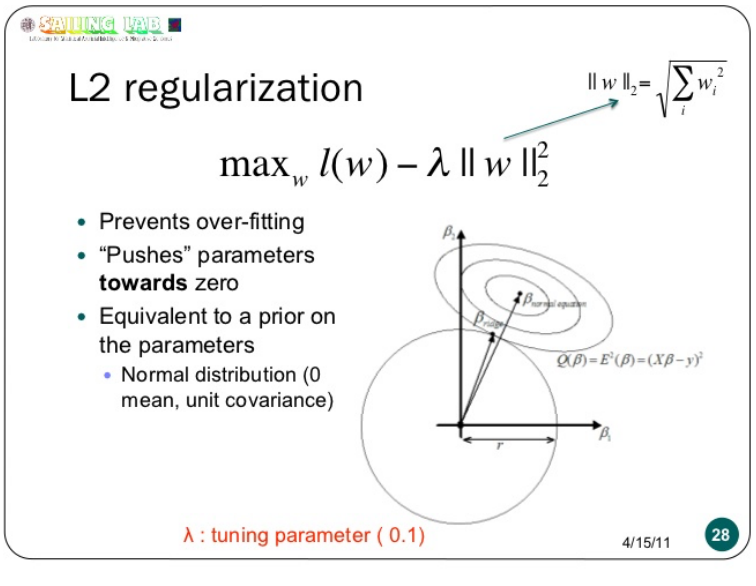
$(wx_1+b-y_1)^2$
$+(wx_2+b-y_2)^2$
$=C$ ↓
椭圆

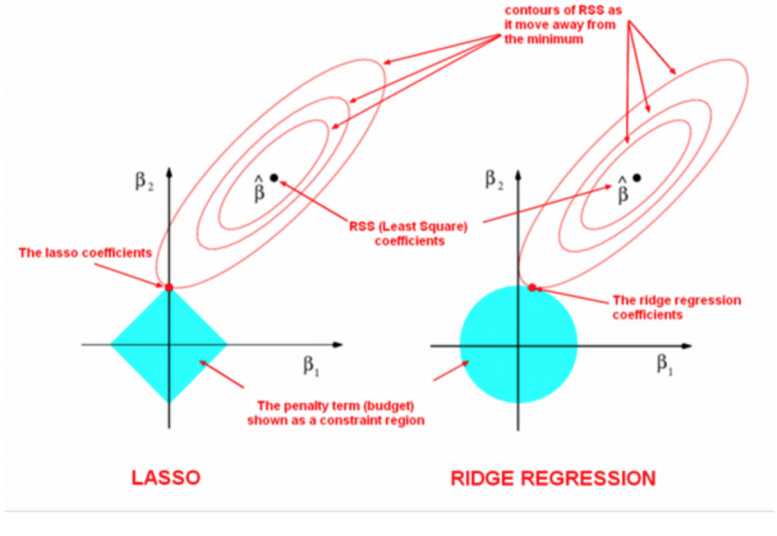
Without regularization, our objective is to find the global cost minimum.

adding a regularization penalty

our objective becomes to minimize the cost function under the constraint that we have to stay within our "budget" (the gray-shaded ball).







Cross-validation

motivation

Model performance is dependent on way the data is split

Not representative of the model's ability to generalize

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=5)
print(cv_results)
```

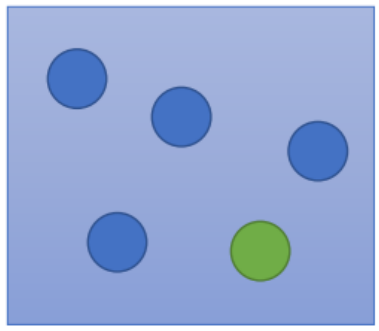
```
[ 0.63919994  0.71386698  0.58702344  0.07923081 -0.25294154]
```

```
np.mean(cv_results)
```

```
0.353275924395870 58
```

Classification: Probabilistic

Generative Model



Box 1

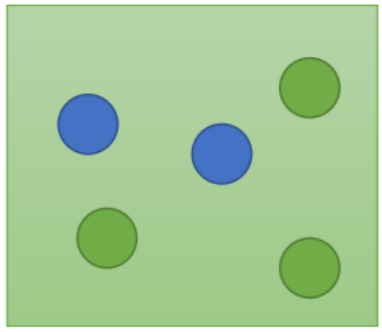
$P(B_1) = 2/3$

Class 1

$P(C_1)$

$P(\text{Blue}|B_1) = 4/5$

$P(\text{Green}|B_1) = 1/5$



Box 2

$P(B_2) = 1/3$

Class 2

$P(C_2)$

$P(\text{Blue}|B_1) = 2/5$

$P(\text{Green}|B_1) = 3/5$

● from one of the boxes

Where does it come from?

$$P(B_1 \mid \text{Blue}) = \frac{P(\text{Blue}|B_1)P(B_1)}{P(\text{Blue}|B_1)P(B_1) + P(\text{Blue}|B_2)P(B_2)}$$

Given an x, which class does it belong to

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

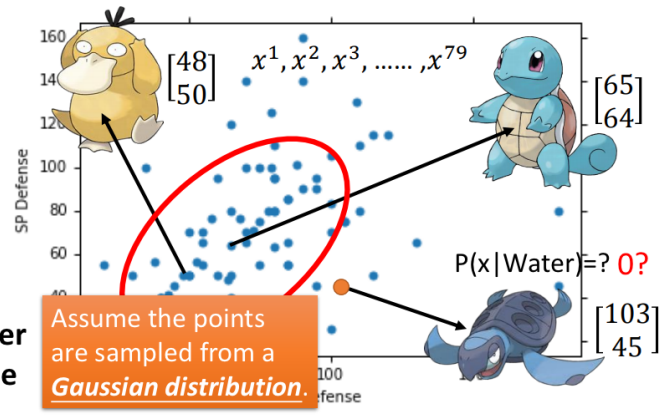
Generative Model $P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$

Estimating the Probabilities
From training data

Gaussian Distribution

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(z-\mu)^2\right).$$

- Considering **Defense** and **SP Defense**

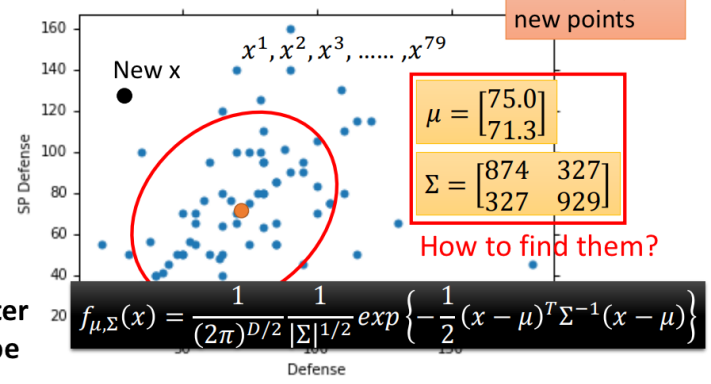


Water Type

Assume the points are sampled from a ***Gaussian distribution***.

Assume the points are sampled from a Gaussian distribution

Find the Gaussian distribution behind them ➡ Probability for new points



Water Type

Maximum Likelihood

step 1

We have the "Water" type Pokémons: $x^1, x^2, x^3, \ldots\ldots, x^{79}$

We assume $x^1, x^2, x^3, \ldots\ldots, x^{79}$ generate from the Gaussian (μ^*, Σ^*) with the ***maximum likelihood***

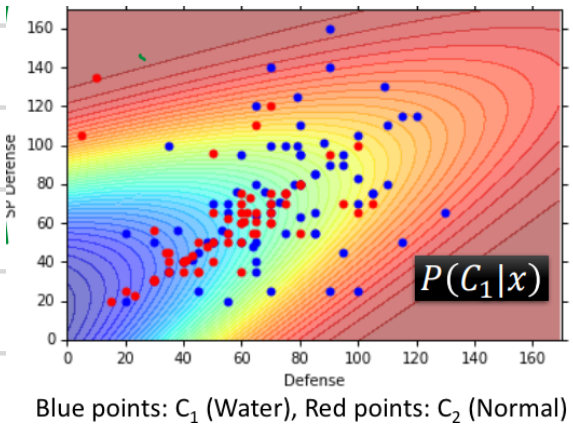
P(x|C1)

$$L(\mu,\Sigma) = f_{\mu,\Sigma}(x^1) f_{\mu,\Sigma}(x^2) f_{\mu,\Sigma}(x^3) \ldots\ldots f_{\mu,\Sigma}(x^{79})$$

$$f_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

$$\mu^*, \Sigma^* = arg \max_{\mu,\Sigma} L(\mu,\Sigma)$$

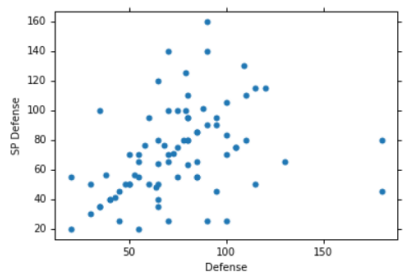
$$\mu^* = \frac{1}{79}\sum_{n=1}^{79} x^n \quad \text{(average)} \qquad \Sigma^* = \frac{1}{79}\sum_{n=1}^{79}(x^n-\mu^*)(x^n-\mu^*)^T$$



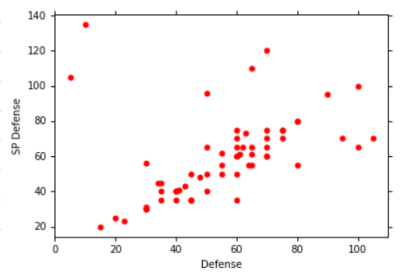
Blue points: C_1 (Water), Red points: C_2 (Normal)

Maximum Likelihood step 2 P(x|G)

Class 1: Water



Class 2: Normal



$\mu^1 = \begin{bmatrix}75.0\\71.3\end{bmatrix}$ $\Sigma^1 = \begin{bmatrix}874 & 327\\327 & 929\end{bmatrix}$ $\mu^2 = \begin{bmatrix}55.6\\59.8\end{bmatrix}$ $\Sigma^2 = \begin{bmatrix}847 & 422\\422 & 685\end{bmatrix}$

step 3. P(C|X)

$$f_{\mu^1,\Sigma^1}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^1|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu^1)^T(\Sigma^1)^{-1}(x-\mu^1)\right\}$$

P(C1) = 79 / (79 + 61) =0.5

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$f_{\mu^2,\Sigma^2}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma^2|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu^2)^T(\Sigma^2)^{-1}(x-\mu^2)\right\}$$

P(C2) = 61 / (79 + 61) =0.44

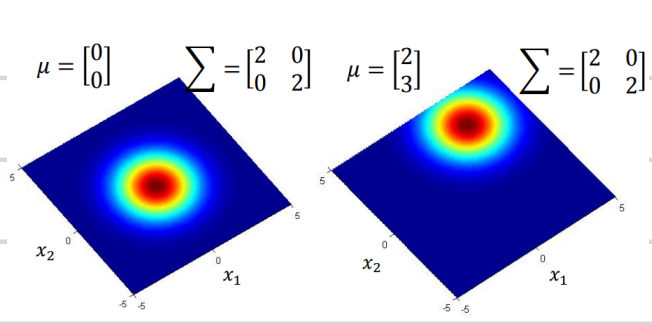
$\mu^2 = \begin{bmatrix}55.6\\59.8\end{bmatrix}$ $\Sigma^2 = \begin{bmatrix}847 & 422\\422 & 685\end{bmatrix}$

If $P(C_1|x) > 0.5$ ➡ x belongs to class 1 (Water)

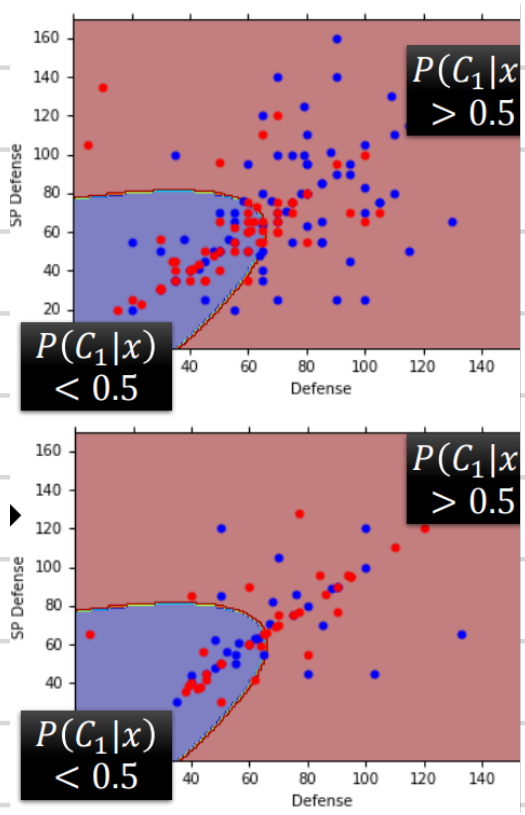
$$f_{\mu,\Sigma}(x) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

Input: vector x, output: probability of sampling x

The shape of the function determines by **mean μ** and **covariance matrix Σ**



Modifying Model



分别求两个class的 $P(x|C)$

- Maximum likelihood

"Water" type Pokémons: $x^1, x^2, x^3, \ldots\ldots, x^{79}$ → μ^1

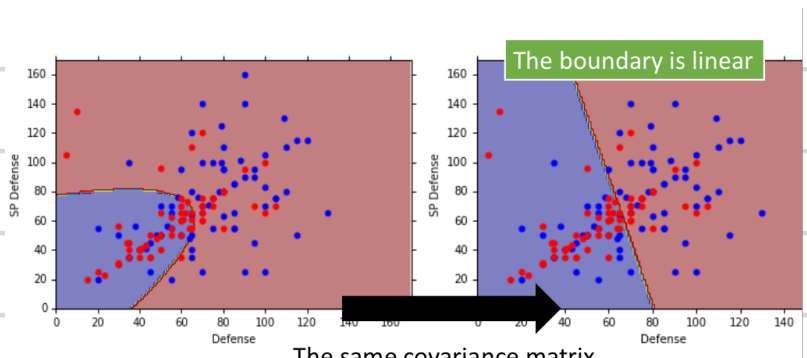
"Normal" type Pokémons: $x^{80}, x^{81}, x^{82}, \ldots\ldots, x^{140}$ → μ^2

Σ

Find μ^1, μ^2, Σ maximizing the likelihood $L(\mu^1,\mu^2,\Sigma)$

$$L(\mu^1,\mu^2,\Sigma) = f_{\mu^1,\Sigma}(x^1) f_{\mu^1,\Sigma}(x^2) \cdots f_{\mu^1,\Sigma}(x^{79}) \times f_{\mu^2,\Sigma}(x^{80}) f_{\mu^2,\Sigma}(x^{81}) \cdots f_{\mu^2,\Sigma}(x^{140})$$

μ^1 and μ^2 is the same $\quad \Sigma = \frac{79}{140}\Sigma^1 + \frac{61}{140}\Sigma^2$



The same covariance matrix

All: total, hp, att, sp att, de, sp de, speed

54% accuracy → 73% accuracy

covariace Σ 相同在同一个式子里求

Three Steps

Function Set (Model)

x →

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

If $P(C_1|x) > 0.5$, output: class 1

Otherwise, output: class 2

Goodness of a function:

- The mean μ and covariance Σ that maximizing the likelihood (the probability of generating data)

Find the best function

$$P(x|C_1) = P(x_1|C_1)\ P(x_2|C_1)\ \cdots\cdots\ P(x_k|C_1)\ \cdots\cdots$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ \vdots \\ x_K \end{bmatrix}$$

1-D Gaussian

For binary features, you may assume they are from Bernoulli distributions.

If you assume all the dimensions are independent, then you are using *Naive Bayes Classifier*.

线性模型

$y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$

样例 $(\boldsymbol{x}, y)$

线性模型(3.2) 的预测值逼近真实标记 y

预测值逼近 y 的衍生物

$y = g^{-1}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)$ $\ln y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$

$g(\cdot)$ 称为“联系函数”(link function).

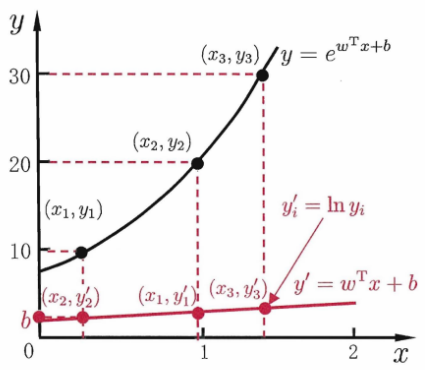


图 3.1 对数线性回归示意图

logistic function)

$$\ln \frac{y}{1-y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$$

$$y = \frac{1}{1 + e^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}}$$

若将 y视为样本 z 作为正例的可能性，则 1-y 是其反例可能性，两者的比值

$$y = \frac{1}{1 + e^{-z}} .$$

$$\ln \frac{p(y=1 \mid \boldsymbol{x})}{p(y=0 \mid \boldsymbol{x})} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b .$$

$$p(y=1 \mid \boldsymbol{x}) = \frac{e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}}{1 + e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}} ,$$

$$p(y=0 \mid \boldsymbol{x}) = \frac{1}{1 + e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}} .$$

Logistic Regression

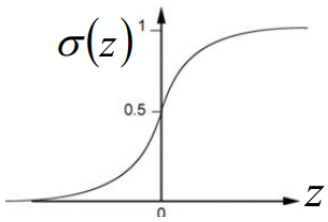
Posterior Probability

$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)}$$

$$= \frac{1}{1 + \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}} = \frac{1}{1 + exp(-z)} = \sigma(z)$$

Sigmoid function

$$z = ln\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}$$



$$\ln\frac{y}{1-y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$$

$$p(y=1 \mid \boldsymbol{x}) = \frac{e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}}{1+e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}},$$

$$p(y=0 \mid \boldsymbol{x}) = \frac{1}{1+e^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}}.$$

Step 1: Function Set

$$f_{w,b}(x) = P_{w,b}(C_1|x)$$

Including all different w and b

$$P_{w,b}(C_1|x) = \sigma(z)$$

$$\sigma(z) = \frac{1}{1+exp(-z)}$$

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

If $P_{w,b}(C_1|x) \geq 0.5$, output C_1

Otherwise, output C_2

Goodness of a Function

Training Data

x^1	x^2	x^3	……	x^N
C_1	C_1	C_2		C_1

Assume the data is generated based on $f_{w,b}(x) = P_{w,b}(C_1|x)$

Given a set of w and b, what is its probability of generating the data?

$$L(w,b) = f_{w,b}(x^1)f_{w,b}(x^2)\left(1 - f_{w,b}(x^3)\right)\cdots f_{w,b}(x^N)$$

The most likely w^* and b^* is the one with the largest $L(w,b)$.

$$w^*, b^* = arg\max_{w,b} L(w,b)$$

x^1	x^2	x^3	……	
C_1	C_1	C_2		

→

x^1	x^2	x^3	……	
$\hat{y}^1 = 1$	$\hat{y}^2 = 0$	$\hat{y}^3 = 1$	……	

$\hat{y}^n$: 1 for class 1, 0 for class 2

$$-lnL(w,b)$$

$$= -lnf_{w,b}(x^1) \rightarrow -[1\ lnf(x^1) + 0\ ln(1-f(x^1))]$$

$$-lnf_{w,b}(x^2) \rightarrow -[1\ lnf(x^2) + 0\ ln(1-f(x^2))]$$

$$-ln\left(1 - f_{w,b}(x^3)\right) \rightarrow -[0\ lnf(x^3) + 1\ ln(1-f(x^3))]$$

$$\vdots$$

Cross entropy

$$L(w,b) = f_{w,b}(x^1)f_{w,b}(x^2)\left(1 - f_{w,b}(x^3)\right)\cdots f_{w,b}(x^N)$$

$$-lnL(w,b) = lnf_{w,b}(x^1) + lnf_{w,b}(x^2) + ln\left(1 - f_{w,b}(x^3)\right)\cdots$$

$\hat{y}^n$: 1 for class 1, 0 for class 2

$$= \sum_n -\left[\hat{y}^n lnf_{w,b}(x^n) + (1-\hat{y}^n)ln\left(1 - f_{w,b}(x^n)\right)\right]$$

Cross entropy between two Bernoulli distribution

	Logistic Regression	***Linear Regression***
Step 1:	$f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$ Output: between 0 and 1	$f_{w,b}(x) = \sum_i w_i x_i + b$ Output: any valu
Step 2:	Training data: $(x^n, \hat{y}^n)$ $\hat{y}^n$: 1 for class 1, 0 for class 2 $L(f) = \sum_n C(f(x^n), \hat{y}^n)$	Training data: $(x^n, \hat{y}^n)$ $\hat{y}^n$: a real number $L(f) = \frac{1}{2}\sum_n (f(x^n) - \hat{y}^n)^2$
Step 3:	Logistic regression: $w_i \leftarrow w_i - \eta \sum_n -\left(\hat{y}^n - f_{w,b}(x^n)\right) x_i^n$	Linear regression: $w_i \leftarrow w_i - \eta \sum_n -\left(\hat{y}^n - f_{w,b}(x^n)\right) x_i^n$

Discriminative v.s. Generative

$P(C_1|x) = \sigma(w \cdot x + b)$

directly find ***w*** and b

Find μ^1, μ^2, Σ^{-1}

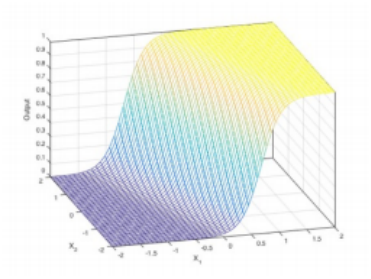
$w^T = (\mu^1 - \mu^2)^T \Sigma^{-1}$

$b = -\frac{1}{2}(\mu^1)^T(\Sigma^1)^{-1}\mu^1 + \frac{1}{2}(\mu^2)^T(\Sigma^2)^{-1}\mu^2 + ln\frac{N_1}{N_2}$

Will we obtain the same set of w and b?

The same model (function set), but different function is selected by the same training data.

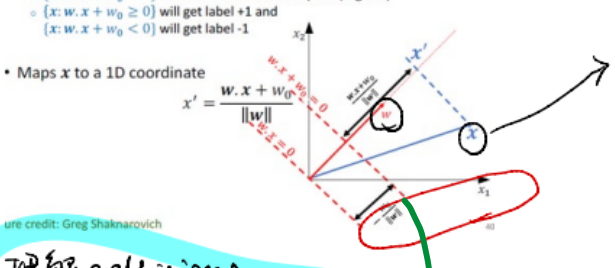
understand z & p → activation function

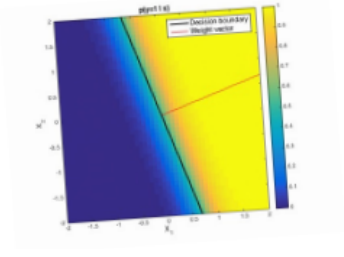


$\hat{w}, \hat{w}_0 = \underset{w,w_0}{\operatorname{argmin}} \sum_i \log\left(1 + \exp(-(w.x^{(i)} + w_0)y^{(i)})\right)$

- Learns a linear decision boundary
  - $\{x: w.x + w_0 = 0\}$ is a hyperplane in $\mathbb{R}^d$ - decision boundary
  - $\{x: w.x + w_0 = 0\}$ divides $\mathbb{R}^d$ into two halfspace (regions)
  - $\{x: w.x + w_0 \geq 0\}$ will get label +1 and $\{x: w.x + w_0 < 0\}$ will get label -1
- Maps x to a 1D coordinate

$x' = \frac{w.x + w_0}{\|w\|}$





ure credit: Greg Shaknarovich

理解coefficient及距离，boundary线

$z = w^T x + w_0$

是一个投射过程，投射到 $w^T x + w_0$ 的垂线上

$\vec{x} \times \vec{w} = w^T x =$

$\frac{\vec{x} \times \vec{w}}{|w_0|}$ 是 $\vec{x}$ 在 $\vec{w}$ 的投影，到 $wx=0$ 的距离

$\frac{w^T x + w_0}{\|w\|}$

是点到 $wx+w_0=0$ 的距离

$z = 0$ $\quad P = \frac{1}{1+e^{(-z)}} = 0.5$

understand discriminative

→ $z = w^T x + b$

→ mapping z to [0,1] $\quad z \rightarrow y = P(C_1|x)$

$y = \frac{1}{1+\exp(-z)}$

$z = 0$, $w^T x + b = 0$, $y = \frac{1}{1+\exp(-z)} = \frac{1}{2}$

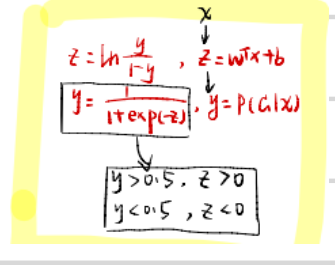
→ calculate z from y $\quad y \rightarrow z = w^T x + b$

understand coefficients of logistic regression

$ln\frac{y}{1-y} = z = w^T x + b$

$ln\frac{p}{1-p}$ log odd

$x \rightarrow z \rightarrow y\,(p)$



Model selection = 实验方法 + metric

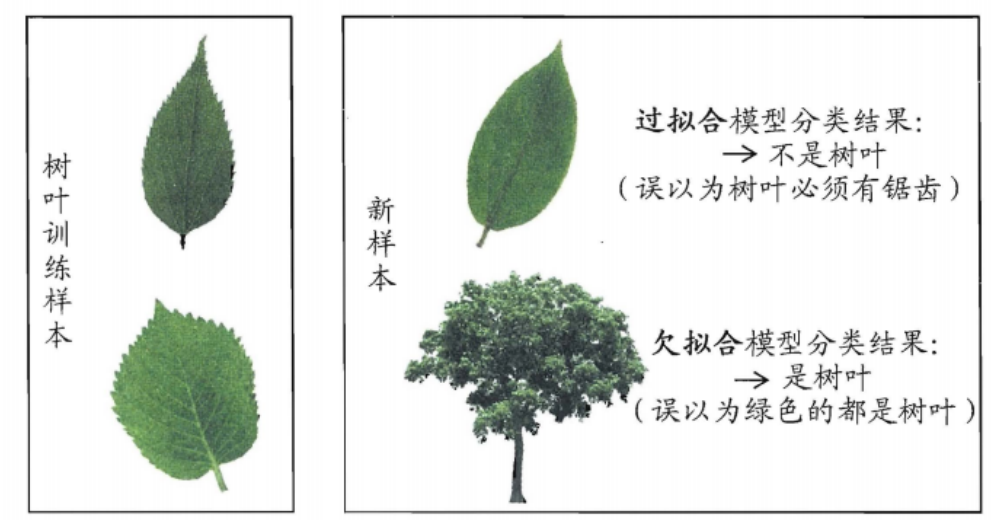


图 2.1 过拟合、欠拟合的直观类比

model-selection

我们该选用 哪 -个

学习算法、

使用哪一种参数配置呢

评估方法

实验方法

(hold-out)

交叉验证法" (cross validation)

自助法 bootstrapping

调参与最终模型

parameter tuning

performance measure)

性能度量

模型的"好坏"是相对的，什么样的模型 是好的?不仅取决于算法和数据，还决定于任务需求.

性能度量

均方误差" (mean squared error)

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}\left(f\left(\boldsymbol{x}_i\right)-y_i\right)^2 .$$

对于数据分布 $\mathcal{D}$ 和概率密度函数 $p(\cdot)$,

$$E(f;\mathcal{D})=\int_{\boldsymbol{x}\sim\mathcal{D}}\left(f\left(\boldsymbol{x}\right)-y\right)^2 p(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$$

分类任务中　　confusion matrix

$TP+FP+TN+FN$ =样例总数.

表 2.1 分类结果混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP (真正例)	FN (假反例)
反例	FP (假正例)	TN (真反例)

查准率 P　　$P=\frac{TP}{TP+FP}$,　　检索出的信息中有多少比例是用户感兴趣的"

查全率 R　　$R=\frac{TP}{TP+FN}$　　用户感兴趣的信息中有多少被检索出来了

若希望将好瓜尽可能多地选出来， 则可通过增加选瓜的数量来实现，

如果将所有西瓜都选上，那么所有的好瓜也　　必然都被选上了，但这样查准率就会较低;

若希望选出 的瓜中好瓜比例尽可能

高，则可只挑选最有把握的瓜， 但这样就难免会漏掉不少好瓜，使得查全率较低.

查准率- 查全率曲线　　P-R曲线"

在很多情形下, 我们可根据学习器的预测结果对样例进行排序, 排在前面的是学习器认为“最可能”是正例的样本, 排在最后的则是学习器认为“最不可能”是正例的样本. 按此顺序逐个把样本作为正例进行预测, 则每次可以计算出当前的查全率、查准率. 以查准率为纵轴、查全率为横轴作图, 就得到了查准率-查全率曲线, 简称“P-R曲线”, 显示该曲线的图称为“P-R图”. 图 2.3 给出了一个示意图.

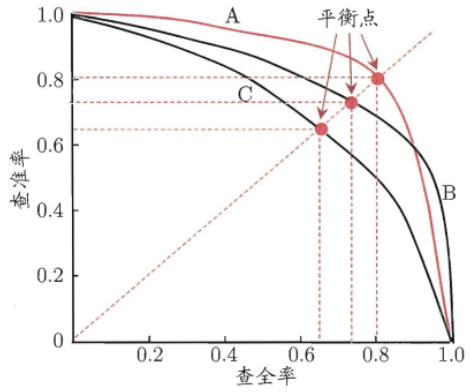


图 2.3 P-R曲线与平衡点示意图

P-R 图直观地显示出学习器在样本总体上的查全率、查准率.在进行比较时, 若一个学习器的 P-R 曲线被另一个学习器的曲线完全“包住”, 则可断言后者的性能优于前者, 例如图 2.3 中学习器 A 的性能优于学习器 C ; 如果两个学习器的 P-R 曲线发生了交叉, 例如图 2.3 中的 A 与 B , 则难以一般性地断言两者孰优孰劣, 只能在具体的查准率或查全率条件下进行比较. 然而, 在很多情形下, 人们往往仍希望把学习器 A 与 B 比出个高低. 这时一个比较合理的判据是比较 P-R 曲线下面积的大小, 它在一定程度上表征了学习器在查准率和查全率上取得相对“双高”的比例. 但这个值不太容易估算, 因此, 人们设计了一些综合考虑查准率、查全率的性能度量.

“平衡点”(Break-Event Point, 简称 BEP)就是这样一个度量, 它是“查准率=查全率”时的取值, 例如图 2.3 中学习器 C 的 BEP 是 0.64, 而基于 BEP 的比较, 可认为学习器 A 优于 B .

但 BEP 还是过于简化了些, 更常用的是 $F1$ 度量:

$$F1 = \frac{2 \times P \times R}{P + R} = \frac{2 \times TP}{\text{样例总数} + TP - TN}.$$

$F1$ 是基于查准率与查全率的调和平均(harmonic mean)定义的:

$$\frac{1}{F1} = \frac{1}{2} \cdot \left(\frac{1}{P} + \frac{1}{R} \right).$$

在一些应用中, 对查准率和查全率的重视程度有所不同. 例如在商品推荐系统中, 为了尽可能少打扰用户, 更希望推荐内容确是用户感兴趣的, 此时查准率更重要; 而在逃犯信息检索系统中, 更希望尽可能少漏掉逃犯, 此时查全率更重要. $F1$ 度量的一般形式——F_β, 能让我们表达出对查准率/查全率的不同偏好, 它定义为

$$F_\beta = \frac{(1 + \beta^2) \times P \times R}{(\beta^2 \times P) + R}, \tag{2.11}$$

其中 $\beta > 0$ 度量了查全率对查准率的相对重要性 [Van Rijsbergen, 1979]. $\beta = 1$ 时退化为标准的 $F1$; $\beta > 1$ 时查全率有更大影响; $\beta < 1$ 时查准率有更大影响.

$$P = \frac{TP}{TP + FP},$$

$$R = \frac{TP}{TP + FN}.$$

比较检验

有了实验评估方法和性能度量, 看起来就能对学习器的性能进行评估比较了: 先使用某种实验评估方法测得学习器的某个性能度量结果, 然后对这些结果进行比较. 但怎么来做这个“比较”呢? 是直接取得性能度量的值然后“比大小”吗? 实际上, 机器学习中性能比较这件事要比大家想象的复杂得多. 这里面涉及几个重要因素: 首先, 我们希望比较的是泛化性能, 然而通过实验评估方法我们获得的是测试集上的性能, 两者的对比结果可能未必相同; 第二, 测试集上的性能与测试集本身的选择有很大关系, 且不论使用不同大小的测试集会得到不同的结果, 即便用相同大小的测试集, 若包含的测试样例不同, 测试结果也会有不同; 第三, 很多机器学习算法本身有一定的随机性, 即便用相同的参数设置在同一个测试集上多次运行, 其结果也会有不同. 那么, 有没有适当的方法对学习器的性能进行比较呢?

统计假设检验(hypothesis test)为我们进行学习器性能比较提供了重要依据. 基于假设检验结果我们可推断出, 若在测试集上观察到学习器 A 比 B 好, 则 A 的泛化性能是否在统计意义上优于 B, 以及这个结论的把握有多大. 下面

泛化 error

以错误率为性能度量,

假设检验　　交叉验证 t 检验　　McNemar 检验

Friedman 检验 与 Nemenyi 后续检验

(bias-variance decomposition)

偏差与方差

对学习算法除了通过实验估计其泛化性能, 人们往往还希望了解它“为什么”具有这样的性能. “偏差-方差分解”(bias-variance decomposition)是解释学习算法泛化性能的一种重要工具.

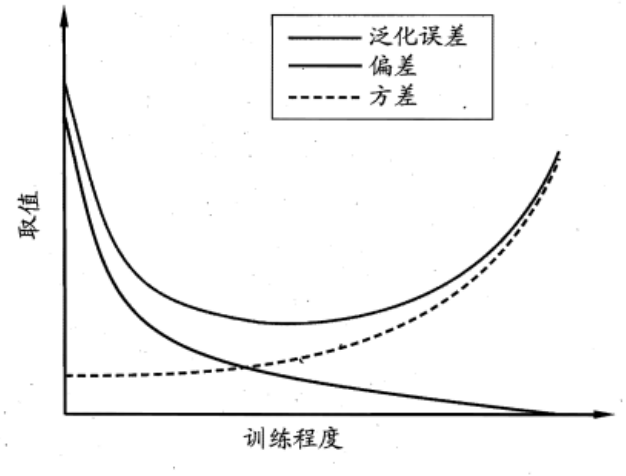


图 2.9 泛化误差与偏差、方差的关系示意图

ROC 与 AUC

很多学习器是为测试样本产生一个实值或概率预测, 然后将这个预测值与一个分类阈值(threshold)进行比较, 若大于阈值则分为正类, 否则为反类. 例如, 神经网络在一般情形下是对每个测试样本预测出一个 $[0.0, 1.0]$ 之间的实值, 然后将这个值与 0.5 进行比较, 大于 0.5 则判为正例, 否则为反例. 这个实值或概率预测结果的好坏, 直接决定了学习器的泛化能力. 实际上, 根据这个实值或概率预测结果, 我们可将测试样本进行排序, "最可能"是正例的排在最前面, "最不可能"是正例的排在最后面. 这样, 分类过程就相当于在这个排序中以某个"截断点"(cut point)将样本分为两部分, 前一部分判作正例, 后一部分则判作反例.

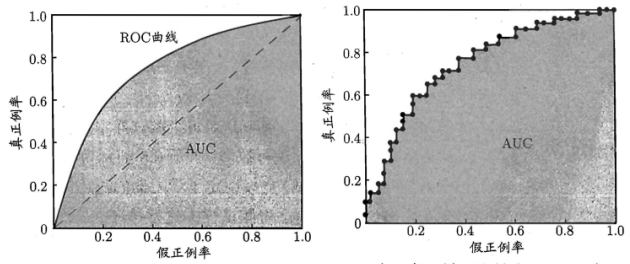
在不同的应用任务中, 我们可根据任务需求来采用不同的截断点, 例如若我们更重视"查准率", 则可选择排序中靠前的位置进行截断; 若更重视"查全率", 则可选择靠后的位置进行截断. 因此, 排序本身的质量好坏, 体现了综合考虑学习器在不同任务下的"期望泛化性能"的好坏, 或者说, "一般情况下"泛化性能的好坏. ROC 曲线则是从这个角度出发来研究学习器泛化性能的有力工具.

绘制出如图 2.4(b)所示的近似 ROC 曲线. 绘图过程很简单: 给定 m^+ 个正例和 m^- 个反例, 根据学习器预测结果对样例进行排序, 然后把分类阈值设为最大, 即把所有样例均预测为反例, 此时真正例率和假正例率均为 0, 在坐标 $(0, 0)$ 处标记一个点. 然后, 将分类阈值依次设为每个样例的预测值, 即依次将每个样例划分为正例. 设前一个标记点坐标为 (x, y), 当前若为真正例, 则对应标记点的坐标为 $(x, y + \frac{1}{m^+})$; 当前若为假正例, 则对应标记点的坐标为 $(x + \frac{1}{m^-}, y)$, 然后用线段连接相邻点即得.

进行学习器的比较时, 与 P-R 图相似, 若一个学习器的 ROC 曲线被另一个学习器的曲线完全"包住", 则可断言后者的性能优于前者; 若两个学习器的 ROC 曲线发生交叉, 则难以一般性地断言两者孰优孰劣. 此时如果一定要进行比较, 则较为合理的判据是比较 ROC 曲线下的面积, 即 AUC (Area Under ROC Curve), 如图 2.4 所示.

$$\mathrm{TPR} = \frac{TP}{TP + FN},$$

$$\mathrm{FPR} = \frac{FP}{TN + FP}.$$



(a) ROC 曲线与 AUC

(b) 基于有限样例绘制的 ROC 曲线与 AUC

Precision

$$P = \frac{TP}{TP + FP},$$

$$R = \frac{TP}{TP + FN}.$$

Recall

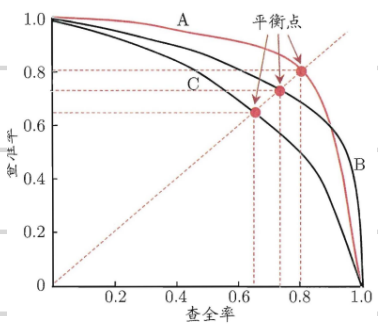


图 2.3 P-R曲线与平衡点示意图

Accuracy

$$\frac{tp + tn}{tp + tn + fp + fn}$$

表 2.1 分类结果混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP (真正例)	FN (假反例)
反例	FP (假正例)	TN (真反例)

Classification metrics

accuracy: Fraction of correctly classi×ed samples

Class imbalance 99% of emails are real; 1% of emails are spam

Could build a classi×er that predicts ALL emails as real

99% accurate!

Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Accuracy:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Precision

$$\frac{tp}{tp + fp}$$

High precision:

Not many real emails predicted as spam

Recall

$$\frac{tp}{tp + fn}$$

High recall:

Predicted most spam emails correctly

F1score:

$$2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

```
knn.score(X_test, y_test)
```

```
print(confusion_matrix(y_test, y_pred))
```

```
print(classification_report(y_test, y_pred))
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
knn = KNeighborsClassifier(n_neighbors=8)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

ROC curve

Modle_selection

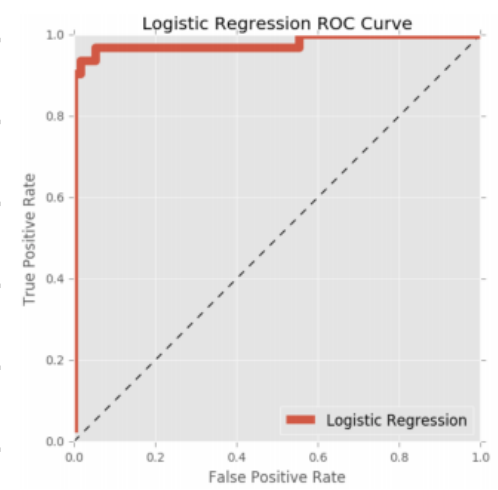
probabilities — Logistic regression outputs probabilities

threshold — By default, logistic regression threshold = 0.5

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

ROC curve

```
from sklearn.metrics import roc_curve
y_pred_prob = logreg.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show();
```



Area under the ROC curve (AUC)

Larger area under the ROC curve = better model

logreg.predict_proba

```
from sklearn.metrics import roc_auc_score
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
logreg.fit(X_train, y_train)
y_pred_prob = logreg.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_pred_prob)
```

AUC using cross-validation

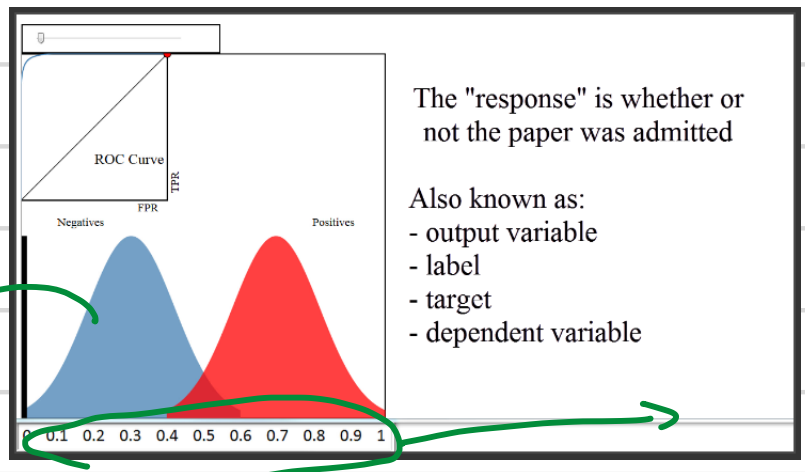
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(logreg, X, y, cv=5,
                            scoring='roc_auc')
print(cv_scores)
```

ROC curves and Area Under the Curve

binary classifier

An ROC curve is the most commonly used way to visualize the performance of a binary classifier, and AUC is (arguably) the best way to summarize its performance in a single number.

Data you know



Your model gives P>0.5 for some of the unadmitted papers

your pre probability

blue and red distributions

250 red pixels: papers that were actually admitted,

250 blue pixels : papers not admitted.

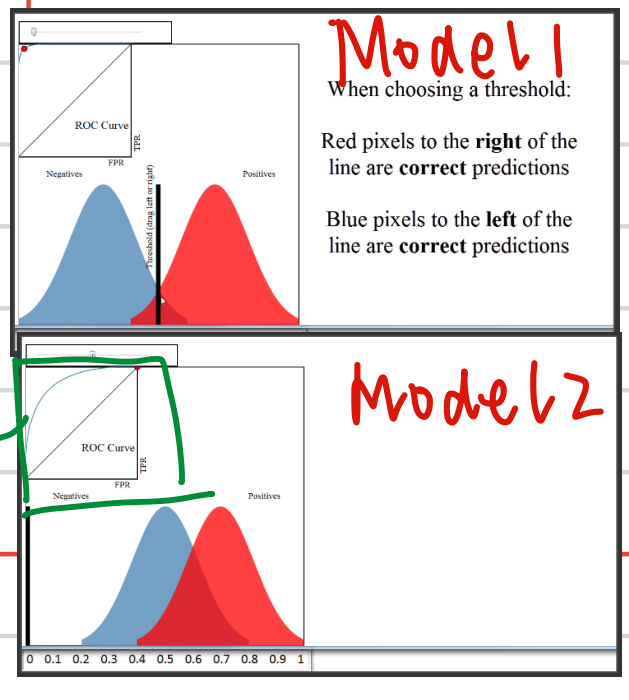
This is your validation (or "hold-out") set, so you know the true admission status of each paper.

x-axis: your predicted probabilities, and

x-axis

y-axis: represents a count of observations,

Model 1



Model 1

your accuracy rate would be above 90%, threshold" at 0.5

ROC

there is a lot more overlap here, and regardless of where you set your threshold, your classification accuracy will be much lower than before.

ROC curve

https://www.dataschool.io/roc-curves-and-auc-explained/

ROC curve

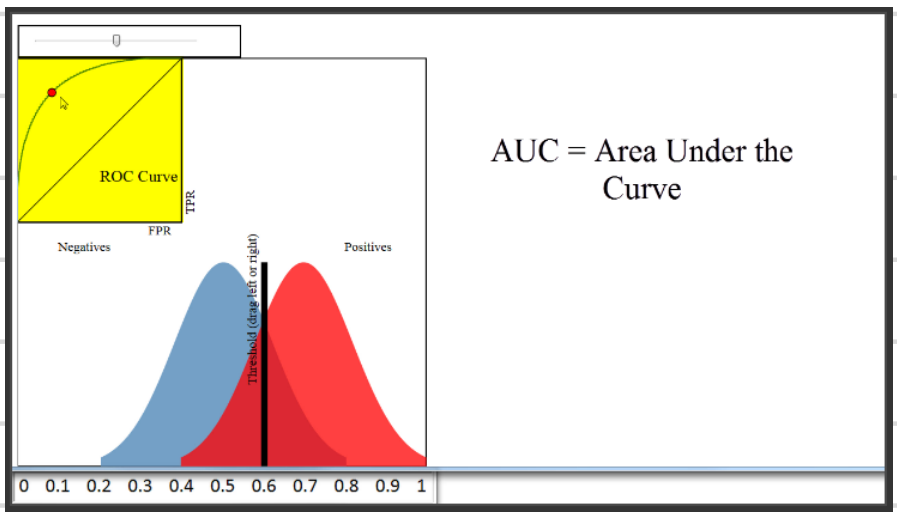
True Positive Rate (on the y-axis)

When the actual classification is positive (meaning admitted), how often does the classfier predict positive?"

False Positive Rate (on the x-axis)

When the actual classification is negative (meaning not admitted), how often does the classifier incorrectly predict positive?"

Area Under the Curve. A



You can think of AUC as representing the probability that a classifier will rank a randomly chosen positive observation higher than a randomly chosen negative observation, and thus it is a useful metric even for datasets with highly unbalanced classes.

threshold

That's actually more of a business decision, in that you have to decide whether you would rather minimize your False Positive Rate or maximize your True Positive Rate.

Hyperparameter tuning

Hyperparameters cannot be learned by ×tting the model

Hyperparameter

- Linear regression: Choosing parameters
- Ridge/lasso regression: Choosing alpha
- k-Nearest Neighbors: Choosing n_neighbors
- Parameters like alpha and k: Hyperparameters

Choosing

- Try a bunch of different hyperparameter values
- Fit all of them separately
- See how well each performs
- Choose the best performing one

It is essential to use cross-validation

Grid search cross-validation

```
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': np.arange(1, 50)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid, cv=5)
knn_cv.fit(X, y)
knn_cv.best_params_
```

```
{'n_neighbors': 12}
```

```
knn_cv.best_score_
```

```
0.933216168717
```

Hold-out set

Split data into training and hold-out set at the beginning

Perform grid search cross-validation on training set

Choose best hyperparameters and evaluate on hold-out set

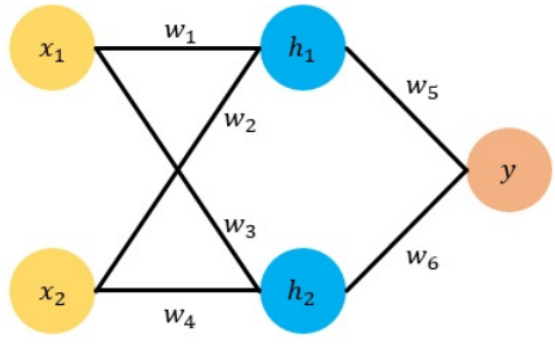
Back Propagation

$$\min_{w} \sum_{x} ||f_w(x) - y||^2$$

Stochastic Gradient Descent

小。求解这类问题有个经典的方法叫做梯度下降法（SGD, Stochastic Gradient Descent），这个算法一开始先随机生成一个 w，然后用下面的公式不断更新 w 的值，最终能够逼近真实结果。

$$w^+ = w - \eta \cdot \frac{\partial E}{\partial w}$$



我们令 $x_1 = 1, x_2 = 0.5$，然后我们令 w_1, w_2, w_3, w_4 的真实值分别是 $1, 2, 3, 4$，令 w_5, w_6 的真实值是 $0.5, 0.6$。这样我们可以算出 y 的真实目标值是 $t = 4$。

随机为他们初始化 $w_1 = 0.5, w_2 = 1.5, w_3 = 2.3, w_4 = 3, w_5 = 1, w_6 = 1$

Feed Forward Pass

$$h_1 = w_1 \cdot x_1 + w_2 \cdot x_2 = 1.25$$
$$h_2 = w_3 \cdot x_1 + w_4 \cdot x_2 = 3.8$$
$$y = w_5 \cdot h_1 + w_6 \cdot h_2 = 5.05$$
$$E = \frac{1}{2}(y - t)^2 = 0.55125$$

要更新 w_5 的值我们就要算 $\frac{\partial E}{\partial w_5}$

$$E = \frac{1}{2}(t - y)^2$$

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_5}$$

$$\frac{\partial E}{\partial y} = 2 \cdot \frac{1}{2} \cdot (t - y) \cdot (-1)$$
$$= y - t$$
$$= 5.05 - 4 = 1.05$$

$$y = w_5 * h1 + w_6 * h2$$

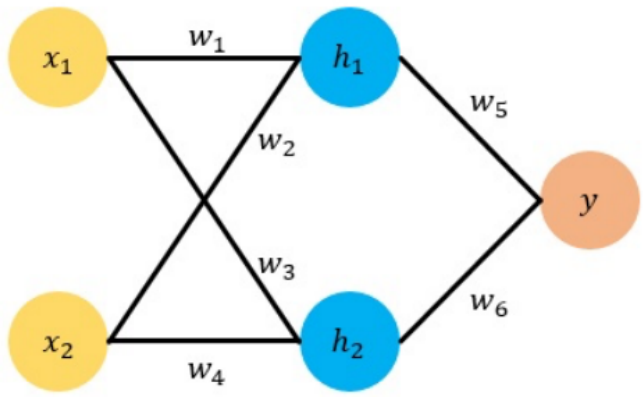
$$\frac{\partial y}{\partial w_5} = h_1 + 0 = h_1 = 1.25$$

$$w_5^+ = w_5 - \eta \cdot \frac{\partial E}{\partial w_5}$$
$$= 1 - 0.1 \cdot 1.3125$$
$$= 0.86875$$

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_5} = (y - t) \cdot h_1$$
$$= 1.05 \cdot 1.25 = 1.3125$$

w_6^+

$$
\begin{aligned}
w_6^+ &= w_6 - \eta \cdot \frac{\partial E}{\partial w_6} \\
&= 1 - 0.1 \cdot 3.99 \\
&= 0.601
\end{aligned}
$$



w_1,

$$
\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}
$$

$$
\frac{\partial y}{\partial h_1} = w_5 + 0 = w_5 \qquad y = w_5 \cdot h_1 + w_6 \cdot h_2
$$

$$
\frac{\partial h_1}{\partial w_1} = x_1 + 0 = x_1 \qquad h_1 = w_1 \cdot x_1 + w_2 \cdot x_2
$$

$$
\begin{aligned}
w_1^+ &= w_1 - \eta \cdot \frac{\partial E}{\partial w_1} \\
&= w_1 - \eta \cdot (y - t) \cdot w_5 \cdot x_1 \\
&= 0.5 - 0.1 \cdot 1.05 \cdot 1 \cdot 1 \\
&= 0.395
\end{aligned}
$$

$$
\begin{aligned}
w_2^+ &= w_2 - \eta \cdot \frac{\partial E}{\partial w_2} \\
&= w_2 - \eta \cdot (y - t) \cdot w_5 \cdot x_2 \\
&= 1.5 - 0.1 \cdot 1.05 \cdot 1 \cdot 0.5 \\
&= 1.4475
\end{aligned}
$$

$$
\begin{aligned}
w_3^+ &= w_3 - \eta \cdot \frac{\partial E}{\partial w_3} \\
&= w_3 - \eta \cdot (y - t) \cdot w_6 \cdot x_1 \\
&= 2.3 - 0.1 \cdot 1.05 \cdot 1 \cdot 1 \\
&= 2.195
\end{aligned}
$$

$$
\begin{aligned}
w_4^+ &= w_4 - \eta \cdot \frac{\partial E}{\partial w_4} \\
&= w_2 - \eta \cdot (y - t) \cdot w_5 \cdot x_2 \\
&= 3 - 0.1 \cdot 1.05 \cdot 1 \cdot 0.5 \\
&= 2.9475
\end{aligned}
$$

Great！现在我们已经更新了所有的梯度，完成了一次梯度下降法。我们用得到的新的 w^+ 再来预测一次网络输出值，根据Feed Forward Pass得到 $y^+ = 3.1768$ ，那么新的误差是 $E^+ = 0.3388$ ，相比于之前的 $E = 0.55125$ 确实是下降了呢，说明我们的模型预测稍微准了一点。只要重复这个步骤，不断更新网络参数我们就能学习到更准确的模型啦。

每经过一个 layer，Weight will be added，end with input

每一个 node 都是一个线性组合的结果

prior layer

Bias-Variance

prediction error

three parts:

- Bias Error
- Variance Error
- Irreducible Error

Bias Error

are the simplifying assumptions made by a model to make the target function easier to learn.

Low Bias: Suggests less assumptions about the form of the target function

High-Bias: Suggests more assumptions about the form of the target function

Examples of **low-bias** machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines

Examples of **high-bias** machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression

Variance Error

Variance is the amount that the estimate of the target function will change if different training data was used.

— picking out the hidden underlying mapping

Low Variance: Suggests small changes to the estimate of the target function with changes to the training dataset. **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Bias-Variance Trade-Off

Linear machine learning algorithms often have a high bias but a low variance.

Nonlinear machine learning algorithms often have a low bias but a high variance

bias-variance trade-off

k-nearest neighbors — low bias and high variance,

increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model

support vector machine

increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

- Increasing the bias will decrease the variance.
- Increasing the variance will decrease the bias.

Information theory

Information theory quantifies the amount of information present.

the amount of information is characterized as:

Probability → chaos →
randomness → disorder

- Predictability:
  - Guaranteed events have zero information.
  - Likely events have little information. (Biased dice have little information.)
  - Random events process more information. (Random dice have more information.)
- Independent events add information. Rolling a dice twice with heads have twice the information of rolling the dice once with a head.

> *In information theory, chaos processes more information.*

Information of an event

$$I(x) = -\log(P(x))$$

In information theory, information and random-ness are positively correlated. High entropy equals high randomness and requires more bits to encode it.-

If has a base of 2, it measure the number of bits to encode the information.

weighed information for all events

Entropy

Entropy can be roughly thought of as how much variance the data has.

- A dataset of only blues ●●●● would have very **low** (in fact, zero) entropy.
- A dataset of mixed blues, greens, and reds ●●●●●● would have relatively **high** entropy.

Information Entropy for a dataset with C classes:

$$H(x) = E_{x\sim P}[I(x)]$$

$$H(x) = -\mathrm{E}_{x\sim P}[logP(x)]$$

$$H(x) = -\sum_{x} P(x)\log P(x)$$

$$E = -\sum_{i}^{C} p_i \log_2 p_i$$

where p_i is the probability of randomly picking an element of class i (i.e. the proportion of the dataset made up of class i).

The easiest way to understand this is with an example. Consider a dataset with 1 blue, 2 greens, and 3 reds: ●●●●●●. Then

$$E = -(p_b \log_2 p_b + p_g \log_2 p_g + p_r \log_2 p_r)$$

We know $p_b = \frac{1}{6}$ because $\frac{1}{6}$ of the dataset is blue. Similarly, $p_g = \frac{2}{6}$ (greens) and $p_r = \frac{3}{6}$ (reds). Thus,

$$\begin{aligned} E &= -(\frac{1}{6}\log_2(\frac{1}{6}) + \frac{2}{6}\log_2(\frac{2}{6}) + \frac{3}{6}\log_2(\frac{3}{6})) \\ &= \boxed{1.46} \end{aligned}$$

Consider 3 blues as an example: ●●●.

$$E = -(1 \log_2 1) = \boxed{0}$$

entropy of a coin

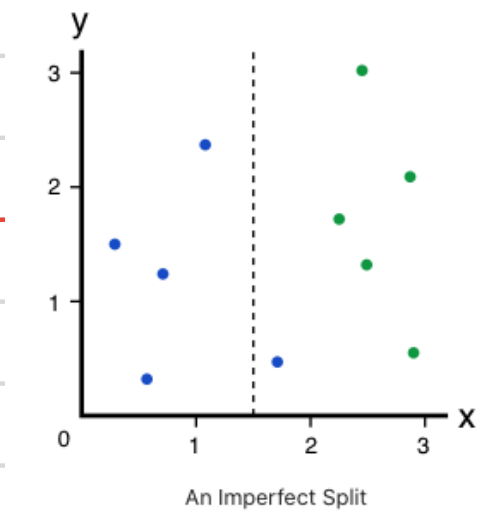
$$H(X) = -p(head) \cdot \log_2(p(head)) - p(tail) \cdot log_2(p(tail)) = -\log_2 \frac{1}{2} = 1$$

bit to represent head (0 = head 1 bit to represent tail (1 = tail).

fair die

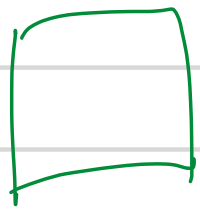
For a fair die, $H(X) = \log_2 6 \approx 2.59$. A fair die has more entropy than a fair coin because it is less predictable.

Information Gain



An Imperfect Split

Before split



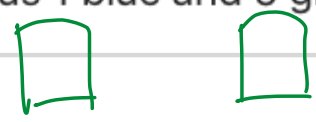
Before the split, we had 5 blues and 5 greens,

$$E_{before} = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = \boxed{1}$$

After the split,

Left Branch has 4 blues, so $E_{left} = \boxed{0}$ because it's a dataset of all one color.

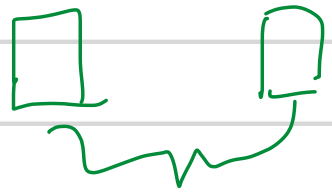
Right Branch has 1 blue and 5 greens, so



$$E_{right} = -(\frac{1}{6} \log_2(\frac{1}{6}) + \frac{5}{6} \log_2(\frac{5}{6})) = \boxed{0.65}$$

Now that we have the entropies for both branches, we can determine the quality of the split by weighting the entropy of each branch by how many elements it has.

After split



$$E_{split} = 0.4 * 0 + 0.6 * 0.65 = \boxed{0.39}$$

We started with $E_{before} = 1$ entropy before the split and now are down to 0.39!

Information Gain = how much Entropy we removed, so

$$\text{Gain} = 1 - 0.39 = \boxed{0.61}$$

This makes sense: higher Information Gain = more Entropy removed, which is what we want. In the perfect case, each branch would contain only one color after the split, which would be zero entropy!

Cross entropy

cross entropy

measures the minimum of bits to encode x with distribution P using the wrong optimized encoding scheme from Q

$$H(P, Q) = -\sum_{x} P(x)\log Q(x)$$

> *In deep learning, P is the distribution of the true labels, and Q is the probability distribution of the predictions from the deep network.*

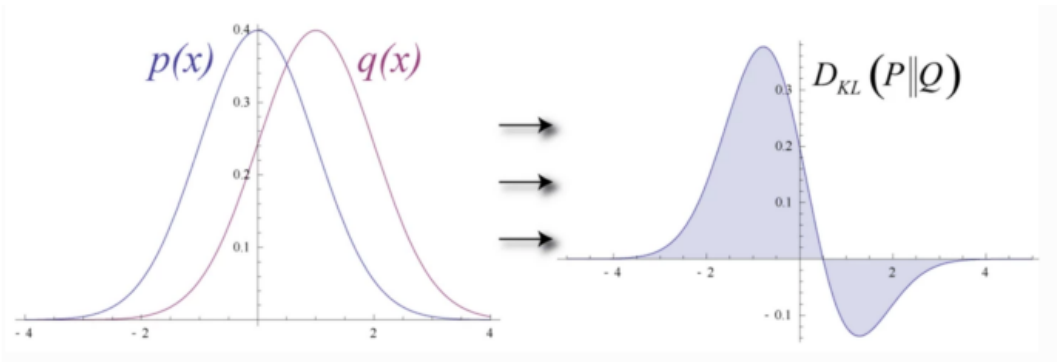
KL Divergence

In deep learning, we want a model predicting data distribution *Q* resemble the distribution *P* from the data. Such difference between 2 probability distributions can be measured by KL Divergence which is defined as:

$$D_{KL}(P||Q) = \mathrm{E}_x \log \frac{P(x)}{Q(x)}$$

$$D_{KL}(P||Q) = \sum_{x=1}^{N} P(x)\log\frac{P(x)}{Q(x)}$$

$$= \sum_{x=1}^{N} P(x)[\log P(x) - \log Q(x)]$$



KL-divergence is not commutative: $D_{KL}(P||Q) \neq D_{KL}(Q||P)$.

cross entropy

$$H(P, Q) = -\sum P\log Q$$

$$= -\sum P\log P + \sum P\log P - \sum P\log Q$$

$$= H(P) + \sum P\log\frac{P}{Q}$$

$$H(P, Q) = H(P) + D_{KL}(P||Q)$$

$$H(P) = -\sum P\log P,$$

$$H(P, Q) = -\sum P\log Q, \quad \text{and}$$

$$D_{KL}(P||Q) = \sum P\log\frac{P}{Q}.$$

So cross entropy is the sum of entropy and KL-divergence.

Cross entropy $H(P, Q)$ is larger than $H(P)$ since we require extra amount of information (bits) to encode data with less optimized scheme from Q if $P \neq Q$. Hence, KL-divergence is always positive for $P \neq Q$ or zero otherwise.

Many cost functions used in deep learning, including the MSE, can be derived from the MLE.

$H(P)$ only depends on P: the probability distribution of the data. Since it is un-related with the model θ we build, we can treat $H(P)$ as a constant. Therefore, **minimizing the cross entropy is equivalent to minimize the KL-divergence**.

$$H(P, Q_\theta) = H(P) + D_{KL}(P||Q_\theta)$$
$$\nabla_\theta H(P, Q_\theta) = \nabla_\theta(H(P) + D_{KL}(P||Q_\theta))$$
$$= \nabla_\theta D_{KL}(P||Q_\theta)$$

Maximum Likelihood Estimation

We want to build a model with $\hat{\theta}$ that maximizes the probability of the observed data (a model that fits the data the best: **Maximum Likelihood Estimation MLE**):

$$\hat{\theta} = \arg\max_\theta \prod_{i=1}^{N} p(x_i|\theta)$$

$$\hat{\theta} = \arg\min_\theta -\sum_{i=1}^{N} \log p(x_i|\theta)$$

NLL and minimizing cross entropy is equivalent

$$\hat{\theta} = \arg\min_\theta -\sum_{i=1}^{N} \log q(x_i|\theta)$$
$$= \arg\min_\theta -\sum_{x\in X} p(x)\log q(x|\theta)$$
$$= \arg\min_\theta H(p, q)$$

Mean square error (MSE

Maximum A Posteriori (MAP)

MLE maximizes $p(y|x;\theta)$. $\theta^* = \arg\max_\theta (P(y|x;\theta)) = \arg\max_w \prod_{i=1}^{n} P(y|x;\theta)$

Alternative, we can find the most likely θ given *y*:

$$\theta^*_{MAP} = \arg\max_\theta p(\theta|y) = \arg\max_\theta \log p(y|\theta) + \log p(\theta)$$

logistic function

$$p(y_i|x_i, w) = \frac{1}{1 + e^{-y_i w^T x_i}}$$

$$J(w) = -\sum_{i=1}^{N} \log p(y_i|x_i, w) - \sum_{j=1}^{d} \log p(w_j) - C$$
$$= \sum_{i=1}^{N} \log(1 + e^{-y_i w^T x_i}) + \frac{\lambda}{2}\|w\|^2 + constant$$

Mean square error

$$\hat{y} = f(x;\theta) y \sim N(y;\mu = \hat{y}, \sigma^2) p(y|x;\theta) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y-\hat{y})^2}{2\sigma^2}\right)$$

with σ pre-defined by users

$$J = \sum_{i=1}^{m} \log p(y|x;\theta)$$

$$= \sum_{i=1}^{m} \log \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}\right)$$

$$= \sum_{i=1}^{m} -\log(\sigma\sqrt{2\pi}) - \log\exp\left(\frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}\right)$$

$$= \sum_{i=1}^{m} -\log(\sigma) - \frac{1}{2}\log(2\pi) - \frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}$$

$$= -m\log(\sigma) - \frac{m}{2}\log(2\pi) - \sum_{i=1}^{m} \frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}$$

$$= -m\log(\sigma) - \frac{m}{2}\log(2\pi) - \sum_{i=1}^{m} \frac{\|y^{(i)} - \hat{y}^{(i)}\|^2}{2\sigma^2}$$

$$\nabla_\theta J = -\nabla_\theta \sum_{i=1}^{m} \frac{\|y^{(i)} - \hat{y}^{(i)}\|^2}{2\sigma^2}$$

Maximum A Posteriori (MAP)

$$\theta^* = \arg\max_{\theta} (P(y|x;\theta)) = \arg\max_{w} \prod_{i=1}^{n} P(y|x;\theta)$$

$$\theta^*_{MAP} = \arg\max_{\theta} p(\theta|y) = \arg\max_{\theta} \log p(y|\theta) + \log p(\theta)$$

$$\theta_{MAP} = \arg\max_{\theta} p(\theta|y)$$

$$= \arg\max_{\theta} \log p(y|\theta) + \log p(\theta) - \log p(y)$$

$$= \arg\max_{\theta} \log p(y|\theta) + \log p(\theta)$$

To demonstrate the idea, we use a Gaussian distribution of $\mu = 0$, $\sigma 2 = 1\lambda$ as the our prior:

$$p(\theta) = \frac{1}{\sqrt{2\pi\frac{1}{\lambda}}} e^{-\frac{(\theta-0)^2}{2\frac{1}{\lambda}}}$$

$$\log p(\theta) = -\log\sqrt{2\pi\frac{1}{\lambda}} + \log e^{-\frac{\lambda}{2}\theta^2}$$

$$= C' - \frac{\lambda}{2}\theta^2$$

$$-\sum_{j=1}^{N} \log p(\theta) = C + \frac{\lambda}{2}\|\theta\|^2 \quad \text{L-2 regularization}$$

Assume the likelihood is also gaussian distributed

$$p(y^{(i)}|\theta) \propto e^{-\frac{(\hat{y}^{(i)} - y^{(i)})^2}{2\sigma^2}}$$

$$-\sum_{i=1}^{N} \log p(y^{(i)}|\theta) \propto \frac{1}{2\sigma^2}\|\hat{y}^{(i)} - y^{(i)}\|^2$$

$$J(\theta) = \sum_{i=1}^{N} -\log p(y^{(i)}|\theta) - \log p(\theta)$$

$$= -\sum_{i=1}^{N} \log p(y^{(i)}|\theta) - \sum_{j=1}^{d} \log p(\theta)$$

$$= \frac{1}{2\sigma^2}\|\hat{y}^{(i)} - y^{(i)}\|^2 + \frac{\lambda}{2}\|\theta\|^2 + constant$$

MSE with L2-regularization.

Preprocessing data

Dummy variables

scikit-learn: OneHotEncoder()

pandas: get_dummies()

Origin
US
Europe
Asia

→

origin_Asia	origin_Europe	origin_US
0	0	1
0	1	0
1	0	0

Pd

```
import pandas as pd
df = pd.read_csv('auto.csv')
df_origin = pd.get_dummies(df)
print(df_origin.head())
```

```
df_origin = df_origin.drop('origin_Asia', axis=1)
print(df_origin.head())
```

```
   mpg  displ   hp  weight  accel  size  origin_Europe  origin_US
0 18.0  250.0   88    3139   14.5  15.0              0          1
1  9.0  304.0  193    4732   18.5  20.0              0          1
```

missing data

Pd

```
df.insulin.replace(0, np.nan, inplace=True)
df.triceps.replace(0, np.nan, inplace=True)
df.bmi.replace(0, np.nan, inplace=True)
```

```
df = df.dropna()
```

Sk

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0
imp.fit(X)
X = imp.transform(X)
```

normalize

```
from sklearn.preprocessing import scale
X_scaled = scale(X)
```

pipeline

Missing

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
logreg = LogisticRegression()
steps = [('imputation', imp),
         ('logistic_regression', logreg)]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
```

```
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
pipeline.score(X_test, y_test)
```

Scaling

```
from sklearn.preprocessing import StandardScaler
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=21)
knn_scaled = pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
accuracy_score(y_test, y_pred)
```

CV

```
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
parameters = {knn__n_neighbors: np.arange(1, 50)}
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=21)
cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(X_train, y_train)
y_pred = cv.predict(X_test)
```

```
print(cv.best_params_)
```

```
{'knn__n_neighbors': 41}
```

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

```
0.956
```

```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.97	0.90	0.93	39
1	0.95	0.99	0.97	75
avg / total	0.96	0.96	0.96	114