CPE 486/586: Deep Learning for Engineering Applications

02 Crash Course on Machine Learning

Spring 2026

Rahul Bhadani

Outline

PyTorch Basic Operations

https://github.com/rahulbhadani/CPE486586_FA25/blob/main/Code/CPE486586_Ch03_Basic_Linear_Algebra_PyTorch.ipynb

Supervised Learning: Linear Regression

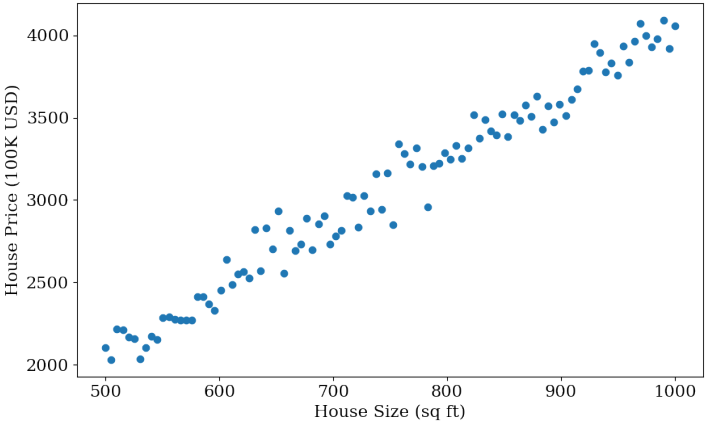
Supervised Learning from Examples

Input $\mathbf{x}$		**Output** y	**Application**
Email	$\rightarrow$	Spam (? (0/1))	Spam Filtering
Audio	$\rightarrow$	Text Transcripts	Speech Recognition
English	$\rightarrow$	Bengali	Machine Translation
Ad, User Infor	$\rightarrow$	Clicked (0/1)	Online Advertising
Image, Radar Info	$\rightarrow$	Position of other Cards	Self-driving Cars
Image of Phone	$\rightarrow$	Defect (0/1)	Automated Quality Inspection

The Supervised Learning Setting

- Consider **Input**: $\mathcal{D} := \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ where $\mathbf{x}_i \in X$ is the feature vector (called **explanatory/predictor variable**), and $y_i \in Y$ is the known true label (called **response variable**).
- Supervised learning algorithms learn from **"right answers"**

Setting up the Linear Regression Problem



Can we guess the price of house for a size of 650 sq ft?

Setting up the Linear Regression Problem

Definition

Linear regression is a kind of supervised machine learning algorithm where a linear discriminant model $g(\mathbf{x})$ (a straight line) is used to fit on the given dataset.

We define a linear model as $g(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + w_0$.

Data Model and Assumptions for SLR

As input data is $\mathcal{D} := \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, we imagine that the actual equation that models the physical process may be

$$y_i = (w_0 + w_1 x_i) + \epsilon_i \tag{1}$$

which in essence says that $Y = \textit{signal} + \textit{noise}$. ϵ_i is the random error or noise term.

For simplicity, we assume that the noise is zero-mean, constant variance noise.

- $\mathbb{E}[\epsilon_i] = 0 \; \forall \; i$.
- $\sigma^2[\epsilon_i] = \sigma^2$ (a constant) $\forall \; i$.
- $\sigma[\epsilon_i, \epsilon_j] = 0 \; \forall \; i \neq j$.

Regression and Least Square

1. The problem of regression becomes finding w_0 and w_1.
2. We use **Least Square** method to estimate w_0, and w_1. We will denote estimated $\hat{w}_0$, and $\hat{w}_1$.
3. The fitted value would be $\hat{y}_i = \hat{w}_0 + \hat{w}_1 x_i$.

We want to minimize

$$Q = \sum (y_i - \hat{y}_i)^2.$$

4. Residuals are $e_i = y_i - \hat{y}_i$. We want residuals to be as low as possible.

The Least-Square Solution

The solution comes out to be the closed form:

$$\hat{w}_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$

$$\hat{w}_0 = \bar{y} - \hat{w}_1 \bar{x}$$

Goodness of Fit using Coefficient of Determination R^2

Goodness of Fit, i.e. how well this linear regression model fits into the given dataset is given by

$$R^2 = 1 - \frac{\text{SSE}}{\text{SSTO}}$$

where $\text{SSTO} = \sum(y_i - \bar{y})^2$.
R^2 is the percentage of the total variation of the y_i explained by the regression model.

Properties of R^2

Some interesting properties of R^2:

1. $R^2 = 1$ when every point lies on the regression straight line.
2. $R^2 = 0$ when data are cloudy.

Some limitations of R^2:

1. $R^2 \to 1$ indicates a strong linear relationship but it might be a poor fit.
2. $R^2 \to 0$ indicates a weak linear relationship but it may be a good **nonlinear fit**.

First-order Model with Two Predictor Variables

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \epsilon_i \tag{2}$$

In this case, our general assumption model, as seen in the single predictor variable will hold true, such as $\mathbb{E}[y_i] = w_0 + w_1 x_{i1} + w_2 x_{i2}$.

In addition, the following value holds, as you see by taking the partial derivatives of the model:

$$\frac{\partial \mathbb{E}[Y]}{\partial X_1} = w_1 \qquad \frac{\partial \mathbb{E}[Y]}{\partial X_2} = w_2 \tag{3}$$

Matrix form of the Linear Regression Model I

$$\mathbf{y}_{n\times 1} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{x}_{n\times p} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1(p-1)} \\ 1 & x_{21} & x_{22} & \cdots & x_{2(p-1)} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & x_{n1} & x_{n2} & \cdots & x_{n(p-1)} \end{bmatrix} \tag{4}$$

$$\mathbf{w}_{p\times 1} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{p-1} \end{bmatrix} \qquad \boldsymbol{\epsilon}_{n\times 1} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{5}$$

Linear Regression Model in Matrix Form I

In matrix form, we can write it as

$$\mathbf{y}_{n\times 1} = \mathbf{x}_{n\times p}\mathbf{w}_{p\times 1} + \boldsymbol{\epsilon}_{n\times 1} \tag{6}$$

Expectation and Variance-Covariance Matrix I

Here, $\mathbb{E}[\epsilon] = 0$, and variance-covariance matrix is

$$\sigma^2[\boldsymbol{\epsilon}] = \begin{bmatrix} \sigma^2 & 0 & 0 & \cdots & 0 \\ 0 & \sigma^2 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \\ 0 & 0 & \cdots & 0 & \sigma^2 \end{bmatrix} = \sigma^2 \mathbf{I} \tag{7}$$

In this case, instead of random variable Y, and X, we have random vector $\mathbf{Y}$, and $\mathbf{X}$, and $\mathbb{E}[\mathbf{Y}] = \mathbf{X}\mathbf{w}$.

Least Square Estimator/Cost Function for MLR

Cost Function

$$J = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \sum_{k=0}^{p-1} w_k x_{ik})^2$$
$$= \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 = \frac{1}{2n}(\mathbf{y} - \mathbf{xw})^\top(\mathbf{y} - \mathbf{xw}) \quad (8)$$

Normal Equation for MLR

Normal Equation

$$\mathbf{x}^\top \mathbf{x}\hat{\mathbf{w}} = \mathbf{x}^\top \mathbf{y} \tag{9}$$

How to solve this for $\mathbf{w}$?

Deriving the Solution

Normal Equation

$$\frac{\partial J}{\partial \mathbf{w}} = -\frac{1}{2n}\mathbf{x}^\top(\mathbf{y} - \mathbf{x}\mathbf{w}) = 0 \Rightarrow -\frac{1}{2n}(\mathbf{x}^\top\mathbf{y} - \mathbf{x}^\top\mathbf{x}\mathbf{w}) = 0 \tag{10}$$

Final Solution and Inverse Condition

Normal Equation

$$\hat{\mathbf{w}} = \frac{1}{n}(\mathbf{x}^\top \mathbf{x})^{-1}\mathbf{x}^\top \mathbf{y}$$

provided $(\mathbf{x}^\top \mathbf{x})^{-1}$ exists.

What's the condition for an inverse to exist?

Supervised Learning: Logistics Regression

Logistics Regression

Logistics Regression is not a *Regression* but a **classification** method.

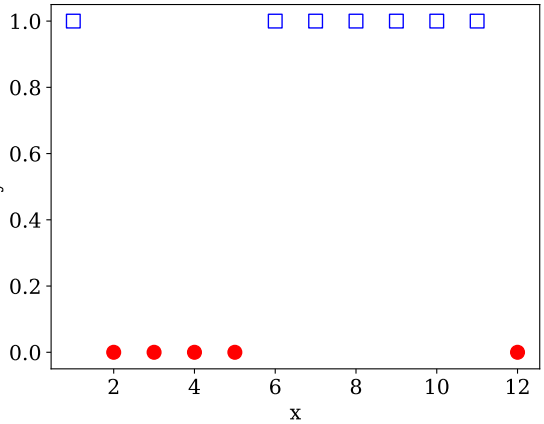
- Linear regression provides us a measure of how close a sample is to the decision boundary from the function $g(\mathbf{x})$. That is the larger $|g(\mathbf{x})|$ the farther a sample is from the plane.
- Unfortunately, this score does not directly correspond to the probability that a sample belongs to a class.
  - We could try to think of a heuristic to connect $g(\mathbf{x})$ to a probability; however, let us try to derive a relationship.
  - We need to revisit Bayes to get the conversation of logistic regression started.
- Linear regression ended up having a very convenient solution (i.e., it was a convex optimization task) and we want to try to obtain a similar results – *if possible*.

A Dataset with Two Classes

Consider the dataset given in the table:

x	1	2	3	4	5	6	7	8	9	10	11
y	1	0	0	0	0	1	1	1	1	1	1

A scatter plot for this dataset

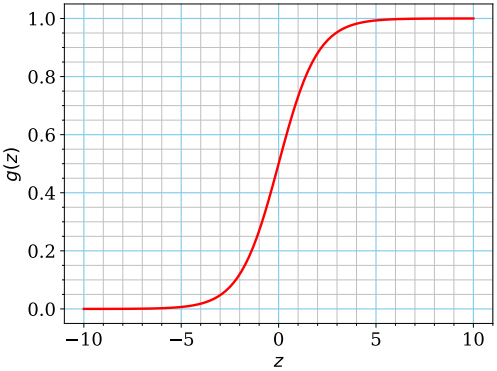


Clearly, distinguishing red and blue points won't work if we use the Linear Regression method. Also, linear regression predicts a continuous value, hence it won't be able to predict either 0 or 1.

Sigmoid Function

By looking at the data, we think we can classify or correctly label the data points if we have a function that is low (0) for some low values of x, and high (1) for some values of x. One such function is **Sigmoid Function** or **Logistics Function** ($g(z)$ in the graph).

$$g(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid Function

We see that

$$\lim_{z \to -\infty} g(z) = 0 \tag{11}$$

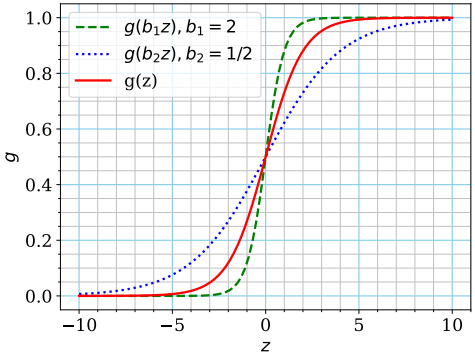
and

$$\lim_{z \to \infty} g(z) = 1 \tag{12}$$

It means, that for a very high positive value of x, the function approaches 1, and for a very high negative value of x, the function approaches 0.

Generalized Sigmoid Function

Now, what if we want the slope to change faster or slower? How should we modify the Sigmoid function?



$$g(z) = \frac{1}{1 + e^{-z \cdot b}}$$

Defining Sigmoid Function for Logistics Regression

In the above example b_1, b_2 and $\pm a$ are learnable parameters that we are going to learn from data the same way we learned w_1 and w_0 in the linear regression. Thus, we consider a parameter vector θ that consists of a vector of parameters to be learned using data $\mathbf{x}$ that we can use to write a generalized version of sigmoid that is data-dependent by setting $z = \theta^\top \mathbf{x}$

$$h_\theta(\mathbf{x}) = g(\theta^\top \mathbf{x}) = \frac{1}{1 + e^{-\theta^\top \mathbf{x}}}$$

Problem Statement in Logistics Regression

The overall goal in Logistics Regression becomes finding $P(Y|X)$, i.e, given information about the random variable X that gives the distribution of the predictor variable, find the probability that the random variable Y takes a certain value.

Assumptions in Logistics Regresions

- **Linearity:** The model assumes that the relationship between the natural log of the probabilities (of each outcome and your predictor variable is linear.
- **No Outliers:** Logistics regression is very sensitive to outliers.
- **Independence:** Each observation point should be independent of each other.

Logistic Regression

From the earlier formulation, h_θ estimates the probability that the data points belong $Y = 1$, and hence we can write it as

$$P(Y = 1|X = x) = h_\theta(\mathbf{x}) \tag{13}$$

$$P(Y = 0|X = x) = 1 - h_\theta(\mathbf{x}) \tag{14}$$

Cost Function

Just like linear regression, we need a cost function based on which we want to estimate θ.

Here, we have two scenarios:

1. $y_i = 1$:
   - if $h_\theta(\mathbf{x}) \approx 1$, cost ≈ 0 (no penalty)
   - if $h_\theta(\mathbf{x}) \approx 0$, cost $\approx$ very high
2. $y_i = 0$:
   - if $h_\theta(\mathbf{x}) \approx 1$, cost $\approx$ very high
   - if $h_\theta(\mathbf{x}) \approx 0$, cost ≈ 0 (no penalty)

Then as the outcome is 0 or 1, it follows the Bernoulli Distribution.

Bernoulli Distribution Revisited

The Bernoulli distribution is a special case of the binomial distribution where a single trial is conducted (so n would be 1 for such a binomial distribution).

The formula for pmf associated with a Bernoulli random variable over possible outcomes x is given as follows:

$$f(x; p) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases} \tag{15}$$

We can also express this formula as,

$$f(x, p) = p^x (1 - p)^{1-x}, \quad x \in \{0, 1\} \tag{16}$$

Likelihood Function

Hence, probability of assigning a label y based on the data $\mathbf{x}$ is given by

$$P(Y = y|X = \mathbf{x}) = h_{\boldsymbol{\theta}}(\mathbf{x})^{y}\Big[1 - h_{\boldsymbol{\theta}}(\mathbf{x})\Big]^{(1-y)} \tag{17}$$

Now, we write taking all data into account (all data points are independent):

$$L(\theta) = \Pi_{i=1}^{n} P(Y = y_i|X = \mathbf{x}_i) = \Pi_{i=1}^{n} h_{\boldsymbol{\theta}}(\mathbf{x}_i)^{y_i}\Big[1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i)\Big]^{(1-y_i)} \tag{18}$$

This is called **likelihood function**.

Log-Likelihood Function

Now, take the log of the likelihood function, which is called the log-likelihood function:

$$LL(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \log(h_{\boldsymbol{\theta}}(\mathbf{x}_i)) + (1 - y_i) \log \left[1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i) \right] \tag{19}$$

This is also called **cross-entropy loss**.

Optimization Function

In the logistics regression, our goal becomes maximizing the log-likelihood by choosing the appropriate θ. For this, we take the partial derivative of $LL(\theta)$ with respect to θ.

Optimization Function

$$\frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{n} \left[y_i - h_{\boldsymbol{\theta}}(\mathbf{x}_i) \right] x_{ij} \tag{20}$$

The derivation comes from the fact that the logistics function has a special property:

$$\frac{\partial g(z)}{\partial z} = g(z)[1 - g(z)] \tag{21}$$

Logistics Regression with SGD

Equating

$$\frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_j} = 0$$

will give optimal θ_j^*. However, the expression doesn't have any closed form, so we resort to using numerical methods for solving the optimization, and hence, in this case, we use **Gradient Ascent Optimization** which is equivalent to minimizing the negative of the log-likelihood function, for which we can use already discussed **Gradient Descent Optimization**.

Updated Rule

$$\theta_j \leftarrow \theta_j - \eta \frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_j}$$

$$\theta_j \leftarrow \theta_j - \eta \sum_{i=1}^{n} \Big[y_i - h_{\boldsymbol{\theta}}(\mathbf{x}_i) \Big] x_{ij}$$

Pseudo Code

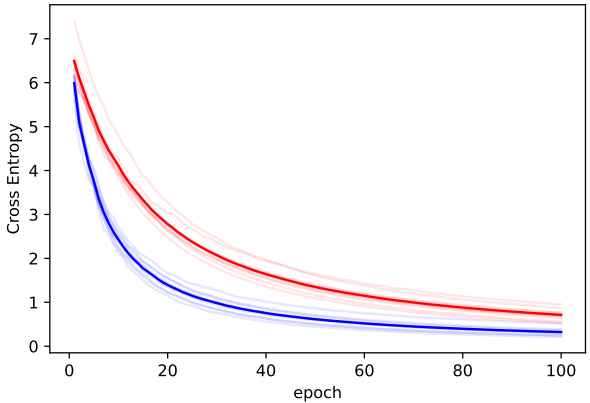
Logistic regression is quite easy to code up using stochastic gradient descent. You'll need an approach to determine when to stop training.

- **Input**: $\mathcal{D} := \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, η, T
- **Initialize**: $\boldsymbol{\theta} = 0$ and $\theta_0 = 0$ % or initialize from $\mathcal{N}(0, \sigma)$
- **for** $t = 1, \ldots, T$
  - **for** $j = 1, \ldots, n$
    - $i = \texttt{np.random.randint}(n)$
    - $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta(y_i - g(\mathbf{x}_i))\mathbf{x}_i$
    - $\theta_0 = \theta_0 + \eta(y_i - g(\mathbf{x}_i))$
  - **if** CONVERGED, STOP

Stopping Criteria

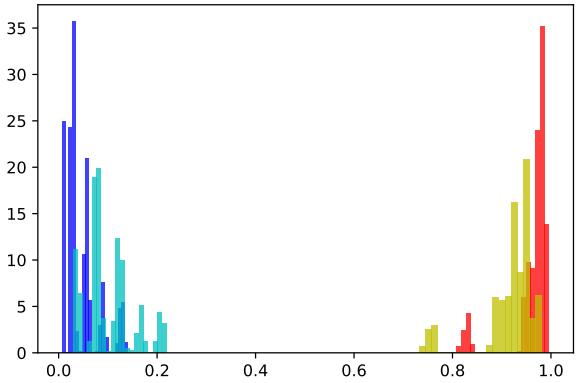
$$\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \leq \epsilon, \text{ or } \texttt{CrossEntropy}(t) - \texttt{CrossEntropy}(t+1) \leq \epsilon \tag{22}$$

Logistic Regression Example



Cross entropy loss on the Iris dataset for two logistic regression models. The blue is trained with a learning rate of $\eta = 0.0025$ and the red is trained with a learning rate of $\eta = 0.001$

Logistic Regression Example



Probabilities, $g(\mathbf{x})$, on predicted data on a validation set. The Cyan/Yellow probabilities are from the model with $\eta = 0.001$ and the Blue/Red probabilities are from the model with $\eta = 0.0025$.

Logistic Regression (in code)

```
class LR:
    def __init__(self, lr=0.0025, epochs=50, split=.1):
        self.lr = lr
        self.epochs = epochs
        self.w = None
        self.b = None
        self.cross_ent = np.zeros(epochs)
        self.split = split

    def score(self, X):
        return 1.0/(1 + np.exp(-(np.dot(self.w, X) + self.b)))
```

Logistic Regression (in code)

```
def crossent(self, X, y):
    ce = np.log(self.score(X[y==1])).sum() + \
             np.log(1.0 - self.score(X[y==0])).sum()
    return -ce
```

Logistic Regression (in code)

```
def fit(self, X, y):
    i = np.random.permutation(len(y))
    X, y = X[i], y[i]
    self.w, self.b = np.zeros(X.shape[1]), 0
    M = np.floor((1-self.split)*len(y)).astype(int)
    Xtr, ytr, Xva, yva = X[:M], y[:M], X[M:], y[M:]
    for t in range(self.epochs):
        # run stochastic gradient descent
        for i in np.random.permutation(len(ytr)):
            self.w += self.lr*(ytr[i] - self.score(Xtr[i]))*Xtr[i]
            self.b += self.lr*(ytr[i] - self.score(Xtr[i]))
        self.cross_ent[t] = self.crossent(Xva, yva)
```

Logistic Regression (in code)

```
def predict(self, X):
    return 1.0*(self.score(X[i]) >= 0.5)

def predict_proba(self, X):
    return self.score(X[i])

def predict_log_proba(self, X):
    return np.log(self.predict_proba(X))
```

Logistics Regression Using Sklearn

```
import numpy as np
from sklearn.linear_model import LogisticRegression

# Create an instance of a Logistic Regression Classifier and fit the data.
logreg = LogisticRegression()
logreg.fit(x, y.ravel())

# Predict
y_hat = logreg.predict(x)

print("Predictions:", y_hat)
```

Logistic Regression Performance Evaluation

Accuracy

Accuracy is a metric used to evaluate the performance of classification models. Informally, accuracy is the fraction of predictions our model got right. It measures the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances.

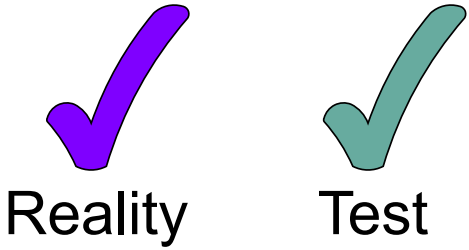
$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Confusion Matrix

Confusion Matrix displays how many of the data points were classified correctly and incorrectly.

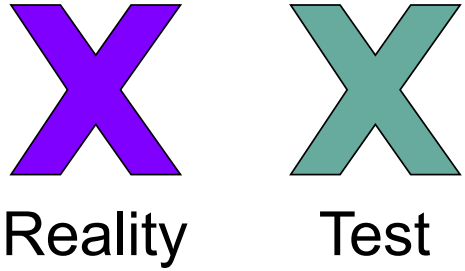
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y, y_hat)
print(confusion_matrix)
```

True Positive (tp)



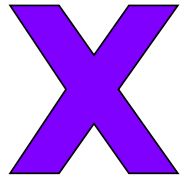
True Positive (tp): Arpita has cancer. She takes a medical test for cancer, and the test result is positive. This is a true positive because the test correctly identified that Arpita has cancer.

True Negative (tn)



True Negative (tn): Mumtaj does not have cancer. She takes the same medical test, and the test result is negative. This is a true negative because the test correctly identified that Mumtaj does not have cancer.

False Positive (fp)

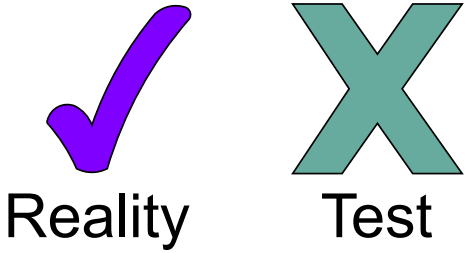


Reality

Test

False Positive (fp): Niu does not have cancer. However, when she takes the medical test, the test result is positive. This is a false positive because the test incorrectly identified that Niu has cancer.

False Negative (fn)



False Negative (fn): Diana has cancer. However, when she takes the medical test, the test result is negative. This is a false negative because the test incorrectly identified that Diana does not have cancer.

Type I error
(false positive)

Type II error
(false negative)

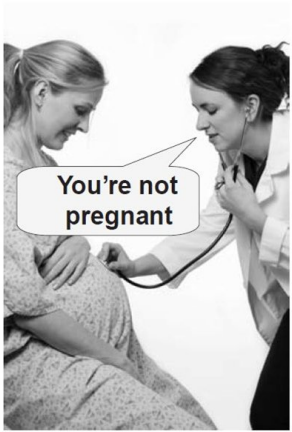


Figure 3.1 Type I and Type II errors

Precision

Precision is the ratio

$$\text{Precision} = \frac{tp}{tp + fp}$$

This means the classifier cannot label a sample as positive if it is negative.

Recall

Recall is

$$\text{Recall} = \frac{tp}{tp + fn}$$

It is the ability of the classifier to find all the positive samples.

F-Score

F-score is the weighted harmonic mean of the precision and recall.

$$F_{score} = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Support

Support is the number of occurrences of each class in the test set.

In Sklearn: ...

```
from sklearn.metrics import classification_report
print(classification_report(y, y_hat))
```

Principal Component Analysis

Why?

- Helpful for reducing number of features
- For 2D/3D visualization
- Helpful in unsupervised algorithms
- Data storage cost can be lowered

Intuition Behind PCA

x_1: length of the car, x_4: height of the car

Class Work

Problem Setting

We are interested in finding projections $\tilde{\mathbf{x}}_n$ of data points $\mathbf{x}_n$ that are are similar to the original data points as possible, but which have significant lower intrinsic dimensionality.

Question

Given a dataset $X = \{\mathbf{x}_1, \cdots \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathbb{R}^D$, is there a subspace $\mathbb{R}^M$, $M << D$ in which X approximately lies?

% Mathematics behind PCA

Feature Preprocessing

- Normalize the dataset so that $\mu = 0$, $\sigma = 1$.

$$\begin{aligned} \boldsymbol{\mu} &= \frac{1}{N}\sum_{i=1}^{n}\mathbf{x}_i \\ \mathbf{x}_i &= \mathbf{x}_i - \boldsymbol{\mu} \\ \sigma_j^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \mu_j)^2 \\ x_{ij} &= \frac{x_{ij}}{\sigma_j} \end{aligned} \tag{23}$$

Rescaling helps in finding duplicate features such as km/h, mph

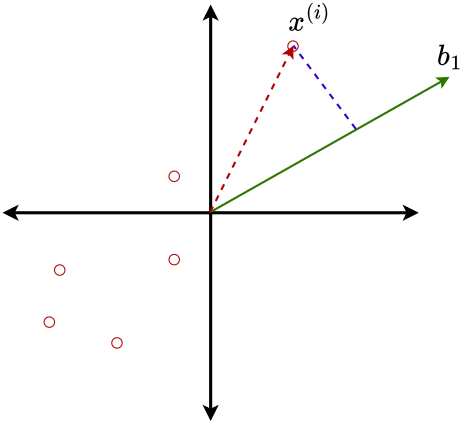
Feature Processing

More concretely, we transform dataset, so that we have iid dataset $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ with 0 mean and convariance matrix Σ:

$$\Sigma = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top \tag{24}$$

Here $\mathbf{x}_i$ is a D-dimensional vector.

Find the Direction of Maximum Variance



To project $\mathbf{x}_i$ onto a new axis $\mathbf{B}$:

$$\mathbf{z}_i = \text{proj}_{\mathbf{B}}(\mathbf{x}_i) = \mathbf{B}^\top \mathbf{x}_i$$

We define the projection matrix as $\mathbf{B} := [\mathbf{b}_1, \cdots, \mathbf{b}_M] \in \mathbb{R}^{D \times M}$.
We assume that the columns of $\mathbf{B}$ are orthonormal such that

$$||b_i|| = 1$$

and $\mathbf{b}_i^\top \mathbf{b}_j = 0$.

Find the Direction of Maximal Variance

We maximize the variance of the low-dimensional coe using a sequential approach. Start with a single vector $\mathbf{b}_1 \in \mathbb{R}^D$ that maximizes the variance of the projected data.

How do we choose unit vector $\mathbf{b}_1$ so that we maximize

$$V = \frac{1}{N} \sum (\mathbf{b}_1^\top \mathbf{x}_i)^2$$

Here, expressing with sum of squares accounts for negative projections in other quadrants.

The Direction with Maximal Variance

Expanding

$$V = \frac{1}{N}\sum(\mathbf{b}_1^\top \mathbf{x}_i)^2 = \frac{1}{N}\sum_{i=1}^{N} \mathbf{b}_1^\top \mathbf{x}_i \mathbf{x}_i \mathbf{b}_1$$
$$= \mathbf{b}_1^\top \left(\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i{}^\top\right)\mathbf{b}_1 = \mathbf{b}_1^\top \Sigma \mathbf{b}_1 \tag{25}$$

The dot product is symmetric, i.e. $\mathbf{b}_1^\top \mathbf{x}_i = \mathbf{x}_i^\top \mathbf{b}_1$.

Lagrange Multiplier to Solve the Optimization Problem

Our goal is to maximize $\mathbf{b}_1^\top \Sigma \mathbf{b}_1$ such that

$$\mathbf{b}_1^\top \mathbf{b}_1 = 1 \Rightarrow \mathbf{b}_1^\top \mathbf{b}_1 - 1 = 0$$

We use the method of Lagrange Multiplier

$$\mathcal{L}(\mathbf{b}_1, \lambda) = \mathbf{b}_1^\top \Sigma \mathbf{b}_1 - \lambda(\mathbf{b}_1^\top \mathbf{b}_1 - 1)$$

Lagrange Multiplier to Solve the Optimization Problem

$$\mathcal{L}(\mathbf{b}_1, \lambda) = \mathbf{b}_1^\top \Sigma \mathbf{b}_1 - \lambda(\mathbf{b}_1^\top \mathbf{b}_1 - 1) \tag{26}$$

Take the gradient with respect to $\mathbf{b}_1$

$$\begin{aligned} &\nabla_{\mathbf{b}_1} \mathcal{L}(\mathbf{b}_1, \lambda) = \Sigma \mathbf{b}_1 - \lambda \mathbf{b}_1 = 0 \\ &\Rightarrow \Sigma \mathbf{b}_1 = \lambda \mathbf{b}_1 \end{aligned} \tag{27}$$

We see that $\mathbf{b}_1$ is an eignevector of the covariance matrix Σ and the lagrange multiplier λ is the corresponding eigenvalue.

Eigenvalue Problem

From the eigenvector property:

$$V = \mathbf{b}_1^\top \Sigma \mathbf{b}_1 = \lambda \mathbf{b}_1^\top \mathbf{b}_1 = \lambda \tag{28}$$

Hence, the variance of the data projected onto a 1-D subspace equal the eigenvalue associated the basis vector $\mathbf{b}_1$ that spans the subspace.

To maximimze the variance of the low-dimensional code, we choose the basis vector associated with the largest eigenvalue of the convariance matrix. This eigenvector is called the **first principal component**.

Prinicpal Components

If we consider all basis vectors, there will be D solution, i.e. D eigenvalues.
If you want to project into, say, $\mathbb{R}^2$ from $\mathbb{R}^6$, you select the first basis b_1 and b_2, sorted by the magnitude of respective eigenvalues.

Note

$b_1, b_2, \cdots$ are new basis vectors for the reduced data.

Reduced Dimensions

The reduced dimension, z_i can be written as

$$\mathbf{z}_i = \begin{bmatrix} \mathbf{b}_1^\top \mathbf{x}_i & \leftarrow \text{projection onto } \mathbf{b}_1 \\ \mathbf{b}_2^\top \mathbf{x}_i & \leftarrow \text{projection onto } \mathbf{b}_2 \\ \mathbf{b}_3^\top \mathbf{x}_i & \leftarrow \text{projection onto } \mathbf{b}_3 \\ \vdots & \vdots \end{bmatrix} \tag{29}$$

Here, $\mathbf{z}_i \in \mathbb{R}^M$.

Reconstruction

We can reconstruct $\mathbf{x}_i$ as folllows, although with loss of information as :

$$\mathbf{x}_i^{\text{recon}} = \mathbf{B}\mathbf{z}_i = \sum_{j=1}^{M} z_{ij}\mathbf{b}_j \tag{30}$$

Note

The quality of the reconstruction depends on the number of principal components M used. If $M = D$, the reconstruction will be perfect, but if $M < D$, some information will be lost.

Reconstruction Error

$$\text{Error} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{x}_i^{\text{recon}}\|^2 \tag{31}$$

This error measures how well the reduced-dimensional data approximates the original data. We should be looking at PCA minimizing this error while reducing the dimensionality of the data.

Note

The reconstruction error is directly related to the eigenvalues of the covariance matrix. The smaller the eigenvalues of the discarded components, the smaller the reconstruction error.

Choosing the Number of Principal Components

To choose the number of principal components M, we can use the following criteria:

- **Variance Explained:** Select enough components to explain a desired percentage of the total variance (e.g., 95%).
- **Scree Plot:** Plot the eigenvalues in descending order and look for an "elbow" point where the eigenvalues start to level off.
- **Cumulative Variance:** Calculate the cumulative sum of the eigenvalues and stop when the cumulative sum reaches a certain threshold.

The number of principal components M is a trade-off between dimensionality reduction and the amount of information retained.

Using Singular Value Decomposition for PCA

After standardizing the dataset, the covariance matrix is $\Sigma = \frac{1}{N}\mathbf{X}\mathbf{X}^\top$ in matrix representation.

Assuming that we denote the matrix of eigenvectors, sorted according to the magnitude of eigenvalue by $\mathbf{U}$, the the PCA transformation of the data is $\mathbf{Z} = \mathbf{U}^\top\mathbf{X}$. By selecting only the first M rows of $\mathbf{Z}$, we have projected the data from D down to M dimensions.

When using the data matrix $\mathbf{X}$ notation, we follow a convention where the rows denote the features from 1 to D and the columns are the samples from 1 to N.

Using Singular Value Decomposition for PCA

We decompose $\mathbf{X}$ using SVD:

$$\mathbf{X} = \mathbf{Q}\mathbf{\Gamma}\mathbf{V}^\top$$

and find that we can write the covariance matrix as

$$\Sigma = \frac{1}{N}\mathbf{X}\mathbf{X}^\top = \frac{1}{N}\mathbf{Q}\mathbf{\Gamma}^2\mathbf{Q}^\top$$

Using Singular Value Decomposition for PCA

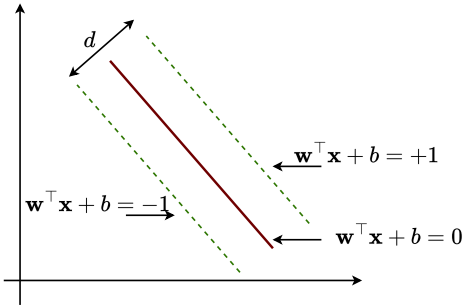
The transformed data thus is

$$\mathbf{Z} = \mathbf{U}^\top \mathbf{X} = \mathbf{U}^\top \mathbf{Q} \mathbf{\Gamma} \mathbf{V}^\top$$

where $\mathbf{U}^\top \mathbf{Q}$ is a simple $D \times N$ matrix which is one on the diagonal and zero everywhere else. Thus, we can write the transformed data in terms of the SVD decomposition of $\mathbf{X}$.

Support Vector Machine

Motivating SVM

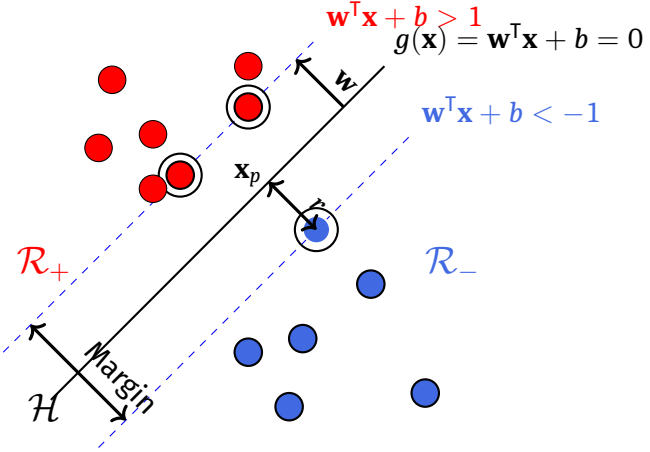


A linear support vector machine (SVM) aims to find a decision plane $\mathbf{w}^\top \mathbf{x} + b = 0$ that maximizes the margin of separation.

Assume that all data points satisfy the constraints:

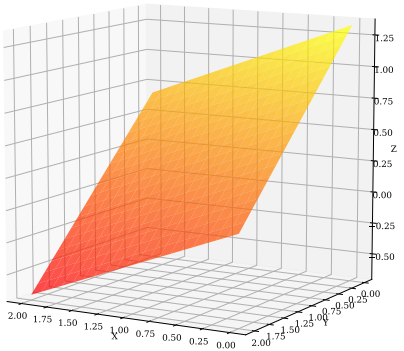
- $\mathbf{w}^\top \mathbf{x} + b \geq +1$ where $y_i = +1$
- $\mathbf{w}^\top \mathbf{x} + b \leq -1$ where $y_i = -1$

The SVM and Linear Classification



Equation for a Hyperplane

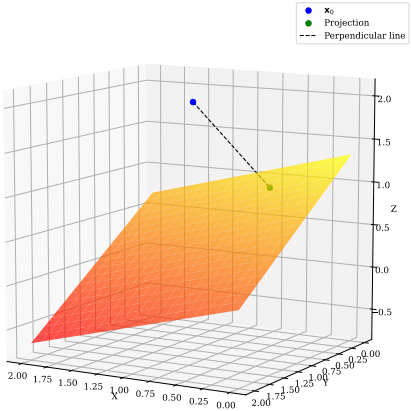
$$\mathbf{w}^\top \mathbf{x} = \beta$$



Euclidean Projection of a Point on a Plane

We project a point $\mathbf{x}_0$ onto the hyperplane $\mathbf{w}^\top \mathbf{x} = \beta$.
The projection point $\mathbf{x}^*$ on the plane is give by

$$\mathbf{x}^* = \mathbf{x}_0 + (\beta - \mathbf{w}^\top \mathbf{x}_0) \frac{\mathbf{w}}{||\mathbf{w}||^2} \qquad (32)$$



Euclidean Projection as an Optimization Problem

One way to come up with the above equation is to solve

$$\min \frac{1}{2}||\mathbf{x} - \mathbf{x}_0||^2$$

subject to

$$\mathbf{w}^\top \mathbf{x} = \beta$$

Distance between two Hyperplanes

Consider two Hyperplanes :
$\mathcal{H}_1 : \mathbf{w}^\top \mathbf{x} = b_1$
$\mathcal{H}_2 : \mathbf{w}^\top \mathbf{x} = b_2$
The distance between two hyperplanes is computed by projecting any point $\mathbf{x}_1$ that is already on one plane $\mathcal{H}_1$ onto $\mathcal{H}_2$.

$$\mathbf{x}_2 = \mathbf{x}_1 + (b_2 - \mathbf{w}^\top \mathbf{x}_0)\frac{\mathbf{w}}{||\mathbf{w}||^2}$$
$$\mathbf{x}_2 = \mathbf{x}_1 + (b_2 - b_1)\frac{\mathbf{w}}{||\mathbf{w}||^2} \tag{33}$$

Distance between two Hyperplanes

Hence, the distance between two hyperplanes is given by

$$d = ||\mathbf{x}_2 - \mathbf{x}_1|| = \left|\left|(b_2 - b_1)\frac{\mathbf{w}}{||\mathbf{w}||^2}\right|\right|$$
$$d = ||\mathbf{x}_2 - \mathbf{x}_1|| = \frac{|b_2 - b_1|}{||\mathbf{w}||} \tag{34}$$

Formulating SVM Problem: Hyperplane Distance

In SVM problem, if we have

- $\mathbf{w}^\top \mathbf{x} + b = +1$
- $\mathbf{w}^\top \mathbf{x} + b = -1$

Let $b_1 = 1 - b$ and $b_2 = -1 - b$.

Then,

$$d = ||\mathbf{x}_2 - \mathbf{x}_1|| = \frac{|b_2 - b_1|}{||\mathbf{w}||}$$
$$= \frac{|-1 - b - 1 + b|}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||} \quad (35)$$

Formulating SVM Problem: Hyperplane Distance

Hence, the distance between two hyperplanes is given by
We want to maximize d, which is equivalent to minimizing $\frac{||\mathbf{w}||}{2}$.

However $||\mathbf{w}||$ is not convex, so instead we minimize $\frac{||\mathbf{w}||^2}{2}$ which is convex.

Final Optimization Problem for SVM

The conversation on constrained optimization was all about minimization tasks and our SVM formulation is a maximization problem. We can convert our maximization task to a minimization one by

$$\mathbf{w}^* = \arg\min \frac{1}{2}\|\mathbf{w}\|_2^2$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \% \; 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0$$

Unsupervised Learning: K-Means

What is Unsupervised Learning?

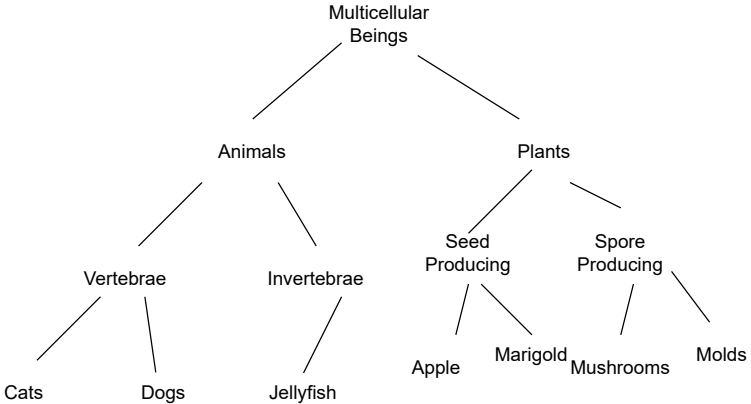
- In contrary to supervised learning, in unsupervised learning, there is no labeled data.
- That is to say, there is no human-supervision involved in labeling a feature vector.

When to use Unsupervised Learning?

1. When we are not sure what we are looking at, unsupervised learning is a useful technique.
2. When we want to just group together several things but we don't care about what these groups belong to.
3. To find a hidden pattern.

When Unsupervised Learning is Useful?

1. The task is to find a pattern or a relationship between variables in order to provide a company with useful information so that they can create marketing strategies, decide on which clients they should focus on to maximize the profits or which customer segment they should put more effort to expand in the market.
2. In grouping together various images. Example: Creating two groups of images from a dataset of images containing the images of cats and dogs.
3. Image Segmentation: Identifying different segments of an image with some semantics.
4. In biological taxonomy:


Multicellular Beings
Animals
Plants
Vertebrae
Invertebrae
Seed Producing
Spore Producing
Cats
Dogs
Jellyfish
Apple
Marigold
Mushrooms
Molds

Clustering

Clustering

- Clustering is a type of unsupervised learning aimed at dividing unlabeled data into various groups.
- In clustering, similar data points based on their features are grouped together while dissimilar data points are separated out.
- Clustering can be used in various applications such as market segmentation, outlier detection, network analysis, and other applications discussed above.

Types of Clustering

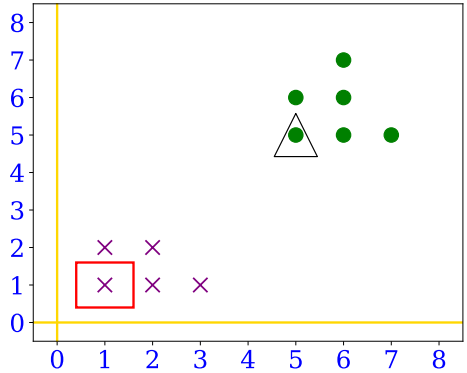
1. Centroid-based or Partition-based clustering
   - Example: K-means
2. Connectivity-based clustering
   - Example: Hierarchical clustering
3. Density-based clustering
   - Example: DBSCAN
4. Graph-based clustering
   - Example: Affinity Propagation
5. Distribution-based Clustering
   - Example: Gaussian Mixture-Models

K-Means

- K-means is the most common type of clustering method that works in a heuristic fashion.
- It starts with assuming how many clusters can be there.
- If there are k clusters, we start by initializing k centroids.
- Consider the following data points in two clusters:
  - Cluster 1: (1,1), (2,1), (1,2), (2,2), (3,1)
  - Cluster 2: (5,5), (5,6), (6,5), (6,6), (7,5), (6,7)

K-Means Clustering Example

- **Step 1:** Initialize with two random centroids (1,1) and (5,5).



K-Means Clustering Example (Continued)

- **Step 2:** Calculate Euclidean distance of each point to the centroid and assign each data point to the nearest centroid.
- In this case, it is clear to see that data points closer to (1,1) are (1,1), (2,1), (3,1), (1,2), (2,2) and data points closer to (5,5) are (5,5), (5,6), (6,5), (6,6), (7,5), and (6,7).
- Now, we recalculate two centroids (x_1, y_1) and (x_2, y_2).

$$
\begin{aligned}
x_1 &= \frac{1+2+3+1+2}{5} = \frac{9}{5} = 1.8 \\
y_1 &= \frac{1+1+1+2+2}{5} = \frac{7}{5} = 1.4 \\
x_2 &= \frac{5+5+6+6+7+6}{6} = \frac{35}{6} = 5.83 \\
y_2 &= \frac{5+6+5+6+5+7}{6} = \frac{34}{6} = 5.66
\end{aligned}
\tag{36}
$$

K-Means Iteration

- Hence, the new centroids are (1.8, 1.4) and (5.83, 5.66).
- Now, we again want to assign points belonging to one of the two centroids calculated by calculating the distance of each point to the centroids.
- So basically, if a point is (x_i, y_i) and two centroids are (x_{c1}, y_{c1}) and (x_{c2}, y_{c2}), then:
  - If $\sqrt{(x_i - x_{c1})^2 + (y_i - y_{c1})^2} < \sqrt{(x_i - x_{c2})^2 + (y_i - y_{c2})^2}$, then (x_i, y_i) belongs to cluster 1.
  - Otherwise, it belongs to cluster 2.

K-Means Convergence

- We will keep repeating until no improvement is possible.
- For more than two clusters, we will have $\arg\min_j \left(\sqrt{(x_i - x_{cj})^2 + (y_i - y_{cj})^2} \right)$ be the cluster index belonging to point (x_i, y_i).

The End