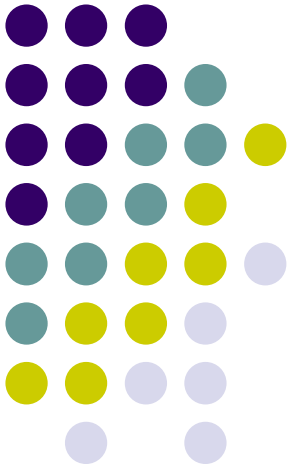
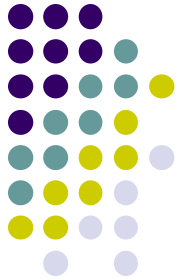
Machine Learning

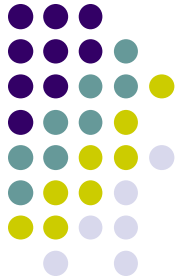
Supervised Learning Basics



Lecture Overview

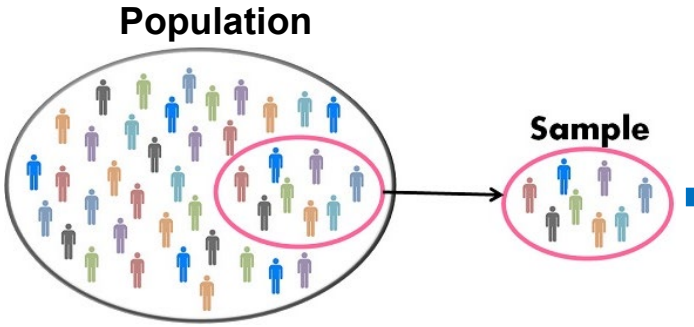


- Review of relevant mathematics (as needed basis)
- Data preparation and preprocessing
- Supervised learning for regression
  - Linear regression modeling
- Supervised learning for classification
  - Bayesian classification
  - Logistic regression
- Model validation, evaluation and selection
- Generative modeling



Data Preparation and Preprocessing

How to Collect a Data set



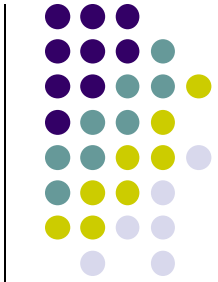
Data set

State	Price	Bedrooms	TotalSqft	LotSize
Nev	260000	2	2042	10173
Nev	66900	3	1392	13069
Vir	127900	2	1792	7065
Nev	181900	3	2645	8484
Nev	262100	2	2613	8355
Nev	147500	2	1935	7056
Nev	167200	2	1278	6116
Nev	395700	2	1455	14422

Excellent health statistics - smokers are less likely to die of age related illnesses.'

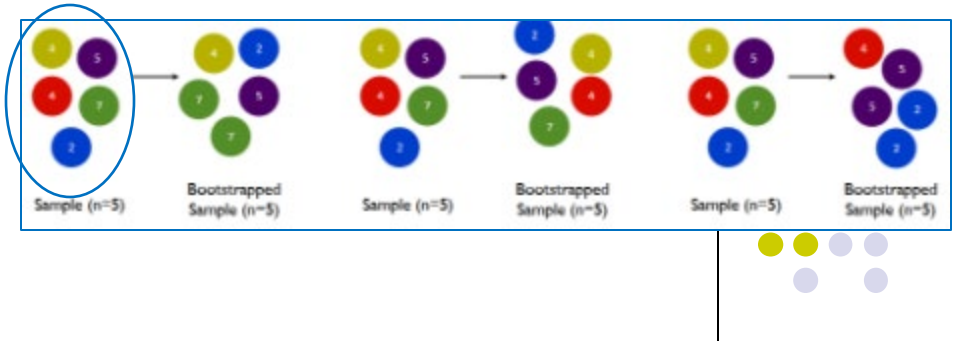
- **A population and sample**
  - A population is the complete collection of all elements to be studied
  - A sample is a sub-collection of members selected from a population.
- **Sampling**
  - With replacement, a member of the population can be chosen more than once.
  - Without replacement, a member of the population can be chosen only once.
- **Sampling error**
  - The discrepancy between a sample statistic and its population parameter.
  - **A sample data** must be collected in an **appropriate way** (random and large enough).
- **Misuse of sample**
  - "There are three kinds of lies: lies, damned lies, and statistics." (B. Disraeli)
  - Numbers don't lie but people can.

Creating Training Data Set and Testing Data Set



- **Why two data sets?**
  - Training data set for modeling and testing data set for testing the model
- **Holdout**
  - A data set is partitioned to training and testing (e.g., 50:50 or 2/3:1/3).
- **Random subsampling**
  - Repeating Holdout several times and computing the average accuracy.
- **Bootstrap sampling**
  - Training data are sampled with replacement.
  - The data not included in the bootstrap samples are test data.
- **Cross-validation** (CV)
  - All the examples in a data set is used for both training and testing, utilizing as much data as possible.
    - CV is computationally expensive but works well for most algorithms.
    - CV is one of the most popular methods to compare learning algorithms and tune algorithm parameters.

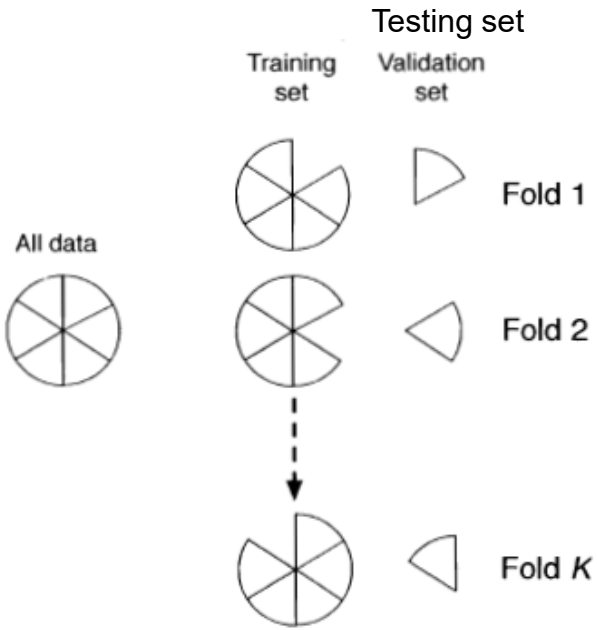
Bootstrap Sampling



- Given a sample data D containing N examples, create a set of D_i, by **drawing n examples** at **random** with replacement from D.
  - Some samples may be repeated in each D_i.
    - E.g., D={1,2,3}, bootstrapped samples D_1={1,1,2}, D_2={1,2,3}, D_3={1,3,3}
  - It is a type of resampling method.
    - An empirical bootstrap sample is drawn from observations.
    - A parametric bootstrap sample is drawn from a parameterized distribution (e.g., a normal distribution).
- **Why resampling?**
  - Loosely based on the law of large numbers, resampled data will approximate the true population.
  - Ca be good approximation and cheaper.

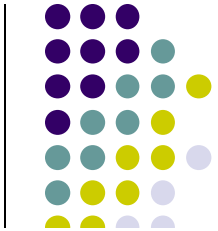
Cross-Validation (CV)

- **CV** is a **resampling method** that uses different portions of the data for training and testing.



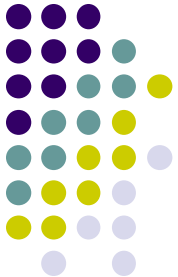
- **Steps of K-fold cross-validation**
  - Shuffle the data set randomly and split it into k groups.
  - For each group in k groups
    1) Take the group as test data set and the remaining groups for training data (k-1 groups).
    2) Run the learning algorithm and evaluate the model on the test data set.
    3) Retain the evaluation score and discard the model.
  - Summarize the results using model evaluation scores (e.g., total or mean score).
- **Configuration of k**
  - The k value is chosen such that each training/testing group of data is large enough to be statistically representative of the data set.
  - K ≈ 5-10, k=10 known to be good through experimentation (generally low bias a modest variance).
  - K=N, known as “Leave One Out CV” (LOOCV).
  - Other variations: stratified CV, repeated CV, nested CV.

Data Preprocessing
(Common data engineering tasks)



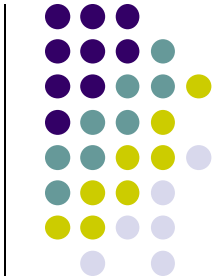
- **The goal** is to prepare the right form of data for the ML algorithm.
- **Common tasks**
  - Aggregation
    - Combining data from multiple data sources (e.g., files, databases, data warehouse)
  - Sampling
    - Selecting a subset of the data when getting a population data is difficult and expensive
  - Cleaning
    - Handling invalid values such as NaN, zeros, missing values. Features with too many invalid values (> λ) can be removed.
  - Data transformation
    - Discretization of continuous to discrete values or transformation of symbolic values to numerical values
  - Scaling and normalization
    - If features are not on a similar scale, gradient descent search can take longer or be lost.
    - The values of dependent variables are not generally changed except a few cases e.g., outputs of perceptron learning
  - Feature selection
    - Identifying the significant variables or removing redundant or irrelevant variables for dimensionality reduction

Data Preprocessing



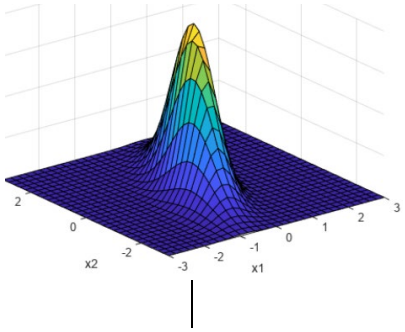
- **Scaling** (converting feature values on a similar scale)
  - Min-max scaling $z = \frac{x - \min(x)}{\max(x) - \min(x)}$
  - Converting values to [0,1] or [-1,1] (by dividing them by the largest/smallest value)
- **Normalization** (changing the shape or distribution of the data)
  - Normalization (computing z score): $z = \frac{x - \bar{x}}{s}$ where s is standard deviation.
  - mean normalization: $z = \frac{x - \text{mean}(x)}{\max(x) - \min(x)}$
- **Transformation**
  - Discretization of continuous values or symbolic data to numerical values
  - Log transformation log(x) to address asymmetrical data to reduce variability and make it less skewed (for easier to interpret)
  - Unit vector transformation (to normalize the vector)
  - Box Cox transformation (to eliminate heteroskedasticity)

Dimensionality Reduction

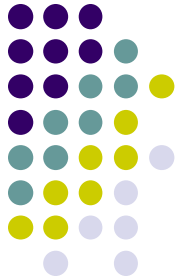


- **Curse of dimensionality** with high-dimensional data (>100 features)
  - With a fixed number of training examples, the predictive power first increases as number of dimensions is increased but then decreases (called "peaking" or "hughes" phenomenon).
  - Causes critical issues in sampling, distance functions, anomaly detection, optimization, and analysis.
- **Dimensionality reduction**
  - Modeling the reduced dimension can be done more efficiently and accurately, which can also result in a simpler model.
  - **Some approaches**
    - Data preprocessing
    - Backward feature elimination, forward feature construction
    - LASSO
    - Principle Component Analysis (PCA)
    - Discriminant analysis: Linear or Generalized Discriminant Analysis
    - Clustering

Feature Selection



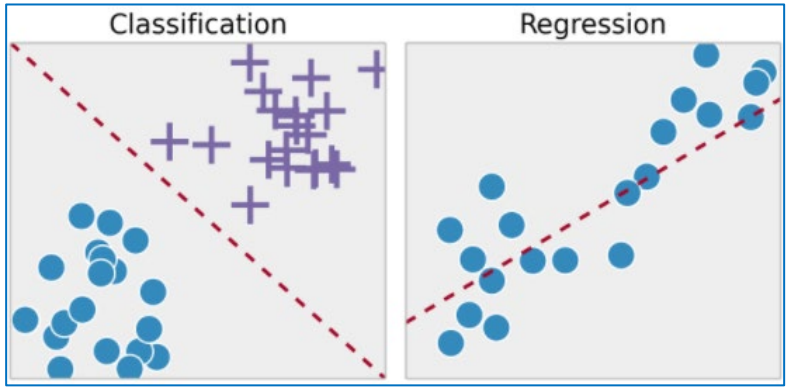
- Key assumptions for multiple regression:
  - **Linearity** between independent and dependent variables
  - **Little or no autocorrelation** (serial correlation in time-series data).
    - Correlation of the same variable in successive time intervals. Measure Durbin-Watson test.
  - **No multicollinearity**
    - Independent variables are highly linearly related (multicollinearity), the estimated model may change erratically by a small change in the data or model or can be overfitted.
    - Measure Variance Inflation Factor (VIF) = $\frac{1}{1-R^2}$ or correlation matrix: VIF=1 (not correlated), 1-5 (moderately correlated), >5 (highly correlated). A rule of thumb (VIF < 3).
  - **Multivariate normality** (normality of residuals)
    - Any linear combination of the variables is normally distributed. Or it may indicate the curvilinearity. Measure q-q plot (straight line) or Jarque Bera test.
  - **Homoscedasticity** (constant variance of residuals)
    - The variance of the residual are similar across the values of the independent variables. Otherwise, it may result in overestimating the goodness of fit.
- Multivariate outliers
  - Outliers may skew the model. Values measured by Mahalanobis's distance function exceeding the critical value of chi-square value with critical alpha = 0.001.
- Dimensionality reduction
  - Remove redundant or irrelevant features, keep only the significant features.



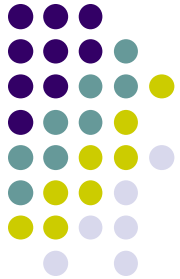
Supervised Learning for Regression

Supervised Learning

- Given a training data set $\{(X_1, X_2, \ldots, X_k, Y_j) \mid j=1 \ldots N\}$
  - Learn a function or classifier $f: X \rightarrow Y$
    - The process of learning **f** is called training.
  - Predict outcomes for a given X (**new data**):
    - If the learned function f is used to predict a continuous value, $\hat{y}$, it's called regression.
    - If the learned classifier f is used to determine a discrete value, $\hat{y}$, it's called classification.
- **Classification vs. regression**
  - Classification is to find the *decision boundary* that **separates** the different groups of data as clearly as possible.
  - Regression is to find the *function* that **fits** the data the best.



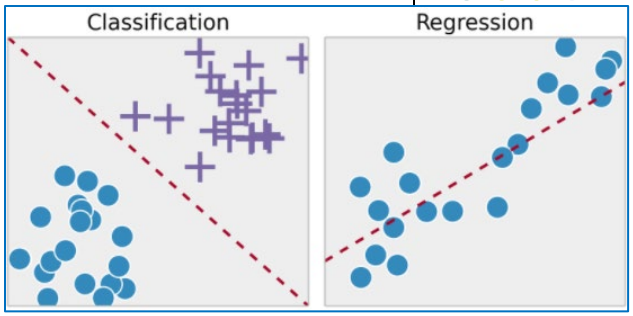
Key Components in Designing Machine Learning Algorithms



- **Goal** (determines learning tasks)
  - To acquire knowledge by finding patterns, relationships (models)
- **Input:** training data set
  - Examples with label/class/target called "**training data**"
- **Output:** Knowledge in different forms
  - Patterns (e.g., repeated data, forms, or shapes)
  - (Concepts) classes (e.g., Yes/No, high/middle/low)
  - Rules (e.g., If credit score > 800, the loan application is approved)
  - Functions (e.g., $f(x) = x^2 + 2x + 3$)

Year	*Sales*	*GDP*
2013	100	0.50%
2014	250	2.00%
2015	275	2.25%
2016	200	1.50%
2017	300	2.50%

Supervised Learning Methods

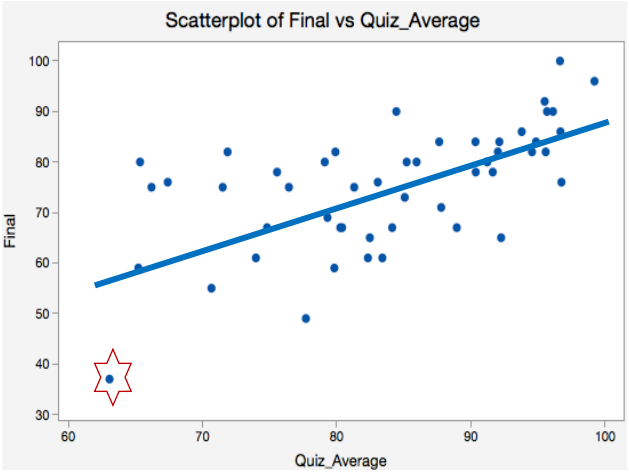


- **Regression**
  - The least-square method
  - Gradient descent method
  - Artificial Neural Network (ANN)
  - K-Nearest Neighbors (KNN), Decision tree, Random forests, Support Vector Machines (SVM), Logistic regression, etc.
- **Classification**
  - Logistic regression
  - Bayesian, Naïve Bayes
  - Artificial Neural Network (ANN), K-Nearest Neighbors (KNN), Decision tree, Random forests, Support Vector Machines (SVM), etc.
- **Both** regression and classification
  - The most supervised learning methods can be used for both regression and classification via small changes.

Supervised Learning for Regression

- Given a training data set $\{(X_1, X_2, \ldots, X_k, Y_j) \mid j=1 \ldots N\}$
  - Learn a function or classifier $f: X \rightarrow Y$
    - The process of learning **f** is called training.
  - For a given X (new data):
    - If the learned function f is used to predict a continuous value, $\hat{y}$, it's called regression.

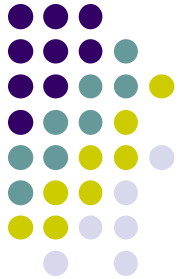
QuizAvg	Final
64	38
66	59
67	80
68	77
71	58
...	...



f(x) = 0.75x + 51

Regression

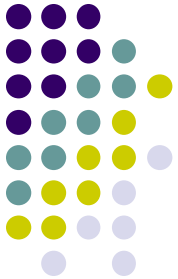
Regression Modeling



- **Regression modeling approaches**
  - *Linear regression*: Models represented by a constant or a parameter multiplied by an independent variable, called *linear combination*
  - *Non-linear regression*: Models represented in all possible forms (many possible forms)

- **Linear regression**
  - Simple regression is to learn a simple function with two variables.
    - $y = w_0 + w_1 x$ (two variables x (independent), y (dependent) where w_0 is intercept, w_1 is slope.)
  - Multiple regression is to learn a polynomial function with multiple variables.
    - $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_k x_k$ (multiple variables)
    - Simple regression is the simplest form of multiple regression.
  - **Outcomes of linear regression modeling**
    - A linear function: $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_k x_k$
    - A non-linear function: $y = w_0 + w_1 x_1 + w_2 x_1^2 + w_3 x_1 x_2 + w_4 x_3^3 \ldots$
    - Note: still a linear combination, but the form of a function can be a **linear** or **non-linear** polynomial.
    - In most cases, we don't need the non-linear regression.

The Linear Regression Model



- **The general linear model** can be written as an equation:
  - $Y = w_0 + w_1 X_1 + \cdots + w_k X_k + \varepsilon$
    - X are independent (exploratory) variables/features, representing the data.
    - Y is a dependent (response) variable.
    - w are coefficients/weights; k is number of features.
    - ε is a random variable (noise by measurements or others), assumed to have a $N(0, \sigma^2)$ distribution; Y is a random variable with mean, $\mu = w_0 + w_1 X_1 + \cdots + w_k X_k$ and variance σ^2.
- Example: **Housing price data with four features**:

Price (K$) (Y)	*Size (X_1)*	*Bedroom (X_2)*	*Bathroom (X_3)*	*Built year (X_4)*
375 (y_1)	1024 (x_{11})	3 (x_{12})	2 (x_{13})	1978 (x_{14})
425 (y_2)	1329 (x_{21})	3 (x_{22})	5 (x_{23})	1992 (x_{24})
…	…	…	…	…
465 (y_N)	1893 (x_{N1})	4 (x_{N2})	4 (x_{N3})	1980 (x_{N4})

Uppercase X_i is a column data.

Lowercase x_i is a row data.

x_{ij} is a value.

- **Housing data in equation form**

$$
\begin{aligned}
y_1 &= w_0 + w_1 x_{11} + w_2 x_{12} + w_3 x_{13} + w_4 x_{14} \\
y_2 &= w_0 + w_1 x_{21} + w_2 x_{22} + w_3 x_{23} + w_4 x_{24} \\
&\cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \\
y_N &= w_0 + w_1 x_{N1} + w_2 x_{N2} + w_3 x_{N3} + w_4 x_{N4}
\end{aligned}
$$

Regression Model in Vector/Matrix Form

- **Stacking all the equations**
  - k number of variables (features)
  - N number of equations (rows)

$$
\begin{array}{lllllll}
y_1 & = & w_0 & + & w_1 x_{11} & + \ \ldots \ + & w_k x_{1k} \\
y_2 & = & w_0 & + & w_1 x_{21} & + \ \ldots \ + & w_k x_{2k} \\
\ldots & & \ldots & & \ldots & \ldots & \ldots \\
y_N & = & w_0 & + & w_1 x_{N1} & + \ \ldots \ + & w_k x_{Nk}
\end{array}
$$

- **The equations in vector form**
  - $\begin{bmatrix} y_1 \\ \ldots \\ y_N \end{bmatrix} = w_0 \begin{bmatrix} 1 \\ \ldots \\ 1 \end{bmatrix} + w_1 \begin{bmatrix} x_{11} \\ \ldots \\ x_{N1} \end{bmatrix} + \cdots + w_k \begin{bmatrix} x_{1k} \\ \ldots \\ x_{Nk} \end{bmatrix}$ or $Y = w_0 1 + w_1 X_1 + \cdots w_k X_k$

X_i is a feature (column) vector.
x_i is a row vector (ith training example).

- **The equations in matrix form**
  - $\begin{bmatrix} y_1 \\ \ldots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \ldots & x_{1k} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & x_{N1} & \ldots & x_{Nk} \end{bmatrix} \begin{bmatrix} w_0 \\ \ldots \\ w_k \end{bmatrix}$ or **Y = Xw**

Y X w

The Goal of Regression Modeling

- **Housing price data set**

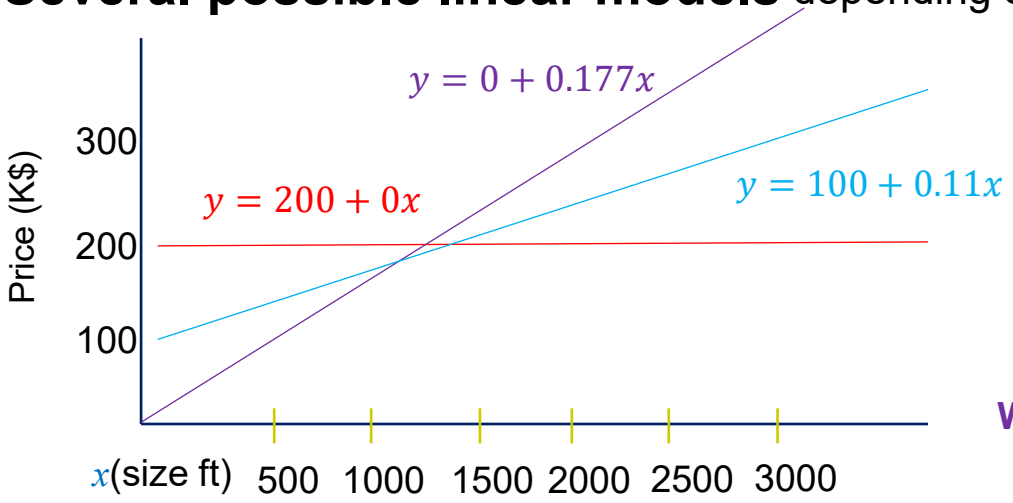
Price (K$) (Y)	*Size (X_1)*	*Bedroom (X_2)*	*Bathroom (X_3)*	*Built year (X_4)*
375 (y_1)	1024 (x_{11})	3 (x_{12})	2 (x_{13})	1978 (x_{14})
425 (y_2)	1329 (x_{21})	3 (x_{22})	5 (x_{23})	1992 (x_{24})
…	…	…	…	…
465 (y_N)	1893 (x_{N1})	4 (x_{N2})	4 (x_{N3})	1980 (x_{N4})

- **Housing price data set in matrix form**
  - $\begin{bmatrix} 375 \\ 425 \\ \dots \\ 465 \end{bmatrix} = \begin{bmatrix} 1 & 1024 & 3 & 2 & 1978 \\ 1 & 1329 & 3 & 5 & 1992 \\ \dots & \dots & \dots & \dots & \dots \\ 1 & 1893 & 4 & 4 & 1980 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_4 \end{bmatrix}$, X and Y are from training data (known); w are unknown.
- **The goal is to find or estimate** w of the function *f(X)*
  - So that Y = f(X) = Xw
  - With the estimated $\hat{w}$, for a new observation, x_{new}, we can ***predict*** *y* *by* $\hat{y} = \hat{w}^T x_{new}$.
  - We want the predicted value $\hat{y} \approx$ actual value y.

Learning a Simple Linear Model

X(Size feet²)	Y(Price $k)
856	399.5
1512	449
865	350
1044	345
…	…

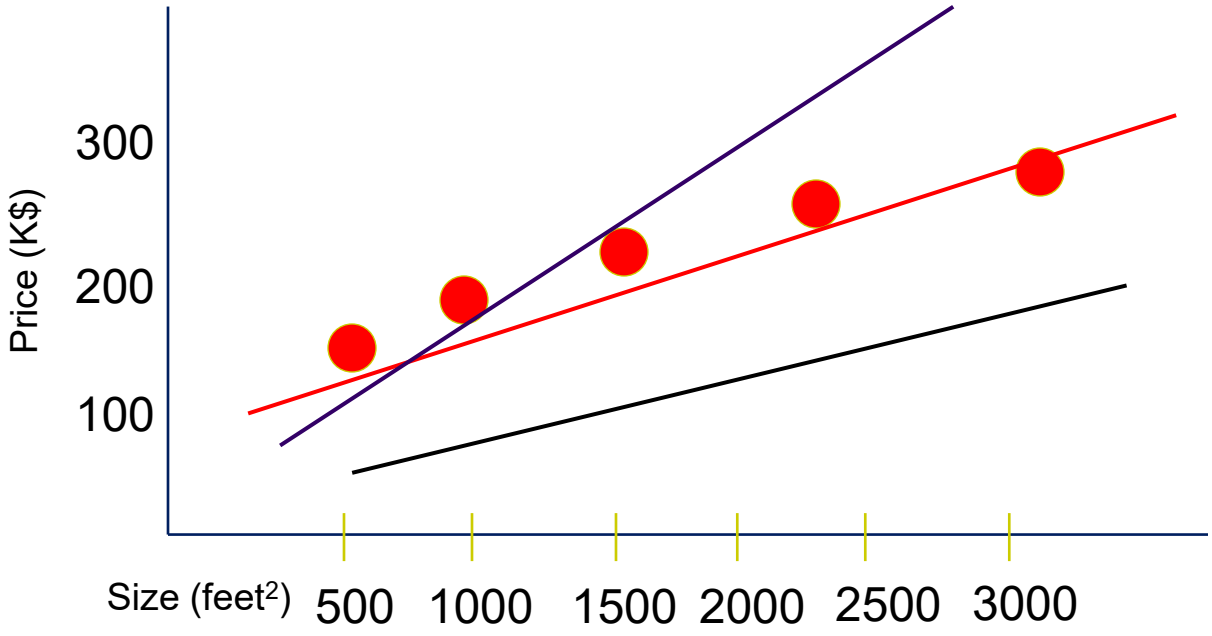
- **Given X** and **Y** value pairs in the training data, **learn/estimate** parameters **w** of function **f(X)** that maps X to Y: **f(X)** = Xw
  - Prediction example: If w_0=50, $w_1 = 300$, what's the price of a house of 895 square feet?
- **Several possible linear models** depending on the **parameters** (w_0, w_1)



Which model do you want to choose?

How do we Find the Best Model?

- We want the most accurate model with the least error in **prediction**.



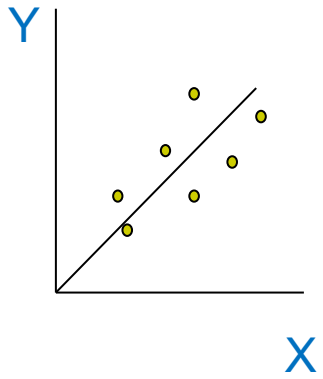
Red dots are training examples.

- **One simple idea**:
  - Choose the parameters $W = (w_0, w_1)$ so that $f(x)$ is closest to the actual values of **Y** in the training examples (x, y).

The Least Square Approach

- The **basic idea** of the (ordinary) least square approach
  - **Find w that minimizes the squared error**:

$$argmin_w \sum_i (y_i - \hat{w}^T x_i)^2$$



y_i is the actual value.

$\hat{w}$ is the estimated parameter.

$\hat{w}^T x_i$ is the predicted value of y, $\hat{y}$ for a given data x_i.

  - Finding w with minimum squared distances between data in a training set and predicted line.
    - **Note:** The regression problem is **restated** as **an optimization problem** of an **error function.**

- **Why least square?**
  - This problem can be solved **analytically** (convincing probabilistic interpretation). How?

Solving Linear Regression by the Least Square Approach

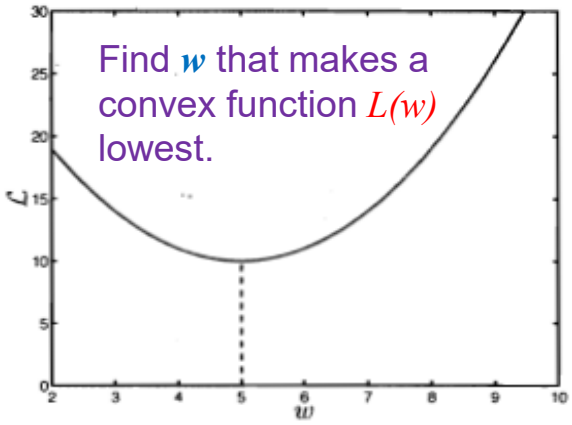
- **Find parameters** W that minimizes the loss function $L(w)$ for <x, y> value pairs of all the training examples (Gauss and Legendre, 1809).

$$\textbf{Minimize } L(w) = \frac{1}{N}\sum_i \left(y_i - f_w(x_i)\right)^2$$

Squaring residuals is important!

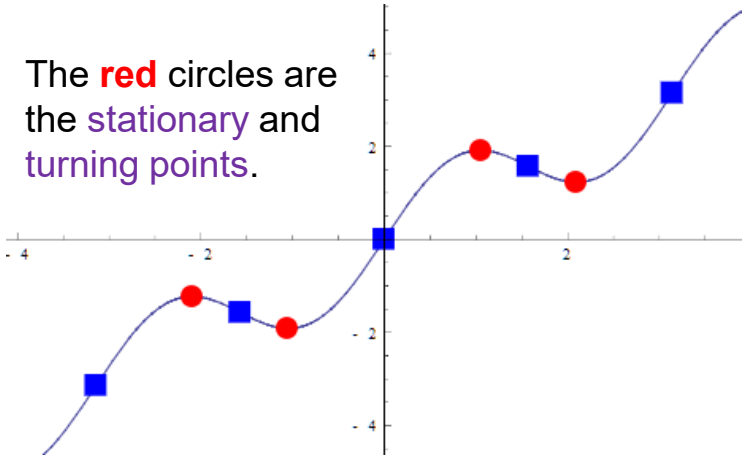
N is the number of samples in a training data set.

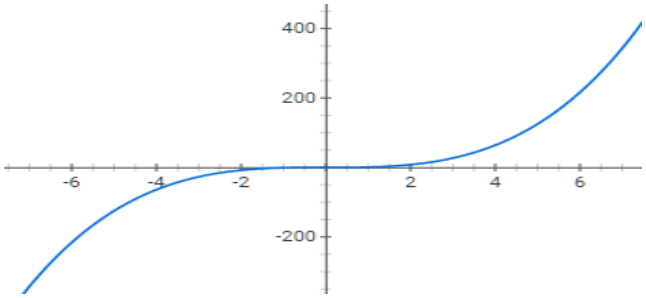
  - $f_w(x_i) = w^T x_i$
  - Residual Sum of Squares (**RSS**) = $\sum_i \left(y_i - f_w(x_i)\right)^2$
  - **Loss** or Mean Squared Error (**MSE**), $L(w) = \frac{1}{N} RSS$



- **How can we find the lowest point of *L(w)*?**

Stationary Points and Turning Points

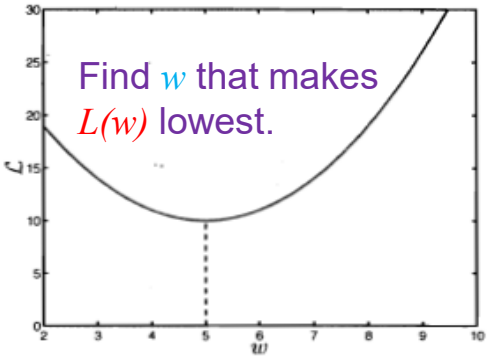


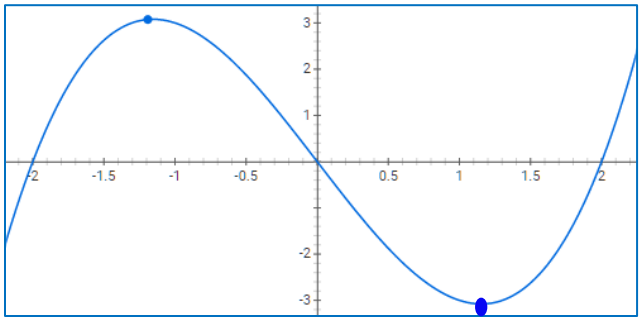


The function **f(x) = x³** has a stationary point, but not a turning point.

- A stationary point of a *differentiable function* of one variable is a point on the graph where the function's *partial derivative* is zero (The gradient is zero where the tangent is horizontal to x-axis).
- A turning point is a point at which the *derivative changes* **sign**.
- Stationary point vs. Turning point
  - A **turning point** is a **stationary point** if the function is differentiable.
  - **Not all stationary points** are turning points. If the function is twice differentiable, the stationary points are horizontal inflection points but not turning points.

Derivatives to Test Turning Points

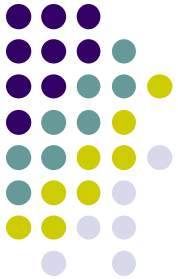




- The **first derivative** test:
  - A local minimum is where $f'(x)$ changes from **–** to **+**.
  - A local maximum is where $f'(x)$ changes from **+** to **–**.
  - **Solving $f'(x) = 0$** returns the x-coordinates of all turning points (solutions).

- The **second derivative** test:
  - **If** $f''(x) > 0$, the turning point at x is concave up (convex); f(x) has a local minimum at x.
  - **Else if** $f''(x) < 0$, the turning point at x is concave down (concave); f(x) has a local maximum at x.
  - **Else**: Saddle point (neither local minima nor maxima)

Deriving a Multiple Regression Model



$$Argmin_w\ L(w) = \frac{1}{N}\sum_i (y_i - w^T x_i)^2$$

from $f_w(x_i) = w^T x_i$

- **To find w that minimizes $L(w)$**
  - Re-arranging $L(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - w^T x_i)^2$ => $\frac{1}{N}(y - wx)^T(y - wx)$

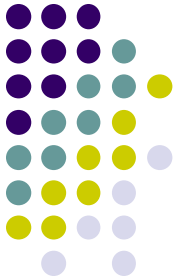
    => ... $= \frac{1}{N}(w^T X^T X w - 2w^T X^T y + y^T y)$
  - Taking a partial derivative of $L(w)$ w.r.t w and solving it ($\frac{\partial L}{\partial w} = 0$)

$$\frac{\partial L}{\partial w} = \frac{2}{N}X^T X w - \frac{2}{N}X^T y = 0 \Rightarrow X^T X w = X^T y$$

(Refer to Rogers at el. book for the detailed derivation)

  - To get the estimated w, multiply $(X^T X)^{-1}$ both sides: $Iw = (X^T X)^{-1} X^T y$
  - Finally, we derived a linear model: $\widehat{\boldsymbol{w}} = (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T y}$ that satisfies the least square error.
  - The math is beautiful!

Predicting a New Value



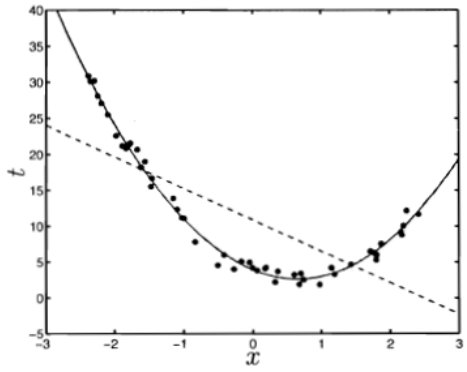
- Given a **new example** x_{new}, the **predicted value**, $\hat{y}_{new}$ from the **model** $\hat{w}^T x_{new}$ obtained by the least square is computed by:

$$\hat{y}_{new} = \hat{w}^T x_{new} \text{ where } \hat{w} = (X^T X)^{-1} X^T y$$

  - A value of y_i is computed by **dot product** of w^T and x_i, that is, $y_i = w^T x_i$

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_k x_{ik}$$

Learning Non-Linear Functions



- **How can we learn non-linear functions?**
  - Use either linear regression or non-linear regression.
- **Using the linear regression** to learn non-linear function
  - Make its terms in a linear model **non-linear**, e.g., $h_1 = X_1^2$, $h_2 = X_2^3$.
    - These new terms can be any functions of independent variables.
  - **A form of non-linear function f(X)**:

$$f(X) = w_0 + w_1 X_1 + w_1 h_1 + w_2 X_2 + w_2 h_2 + w_3 X_3 + w_3 h_3 + \ldots + w_k X_k$$

    - The values of h can be **pre-computed**, replaced in the training data, then fed to $\hat{w} = (X^T X)^{-1} X^T y$ (if using the least square approach).
    - **Note**: The problem of learning non-linear functions remains as a multiple linear regression since it is still a linear combination of terms with w, X and h, but the function learned from the process can be non-linear.

A Modeling Example and Error Measures

- Residual Sum of Squares (**RSS**)

  RSS = $\sum_i (y_i - \hat{y})^2$
- Mean Squared Error (**MSE**)

  MSE = $\frac{1}{N}\sum_i (y_i - \hat{y})^2$
- Root MSE (**RMSE**)

  RMSE = $\sqrt{MSE}$

  (better measure as it is the same unit as the data)
- **Coefficient of determination**

  R^2 = $1 - ({}^{RSS}/_{TSS})$

  where total sum of squares (**TSS**) = $\sum_i (y_i - \bar{y})^2$

  $\bar{y}$ is mean of $\hat{y}$.

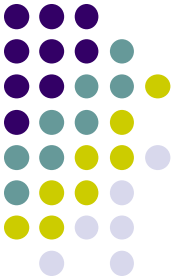
  A higher **R^2** indicates more variability is explained by the model.
- Confidence interval
  - Standard Error (SE) = ${}^{\sigma}/_{\sqrt{N}}$,
    - Upper 95% limit = $\bar{x}$ + (SE*1.96)
    - Lower 95% limit = $\bar{x}$ + (SE*1.96)
  - Standard error of regression = $\sqrt{MSE}$

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

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

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

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

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

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

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % mse(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test,
diabetes_y_pred))

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

Popular data sets: iris, diabetes, digits, wine, breast_cancer, etc.

Optimizing Loss Function Iteratively

(to deal with a huge matrix X from a large data set and computing inverse matrix)



- For **simple regression**, **minimize $L(w_0, w_1)$** w.r.t (w_0, w_1)

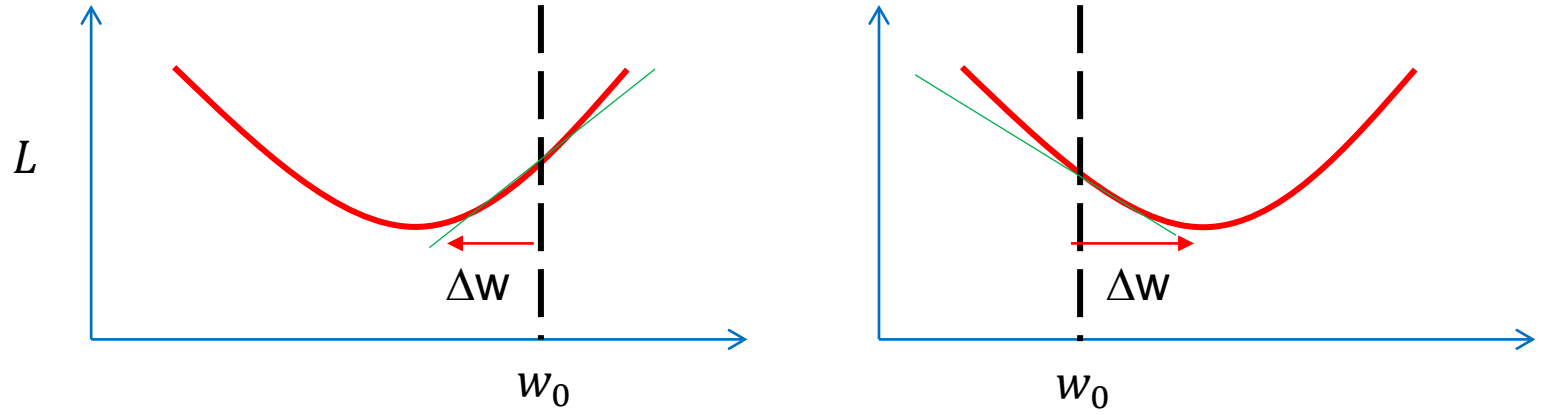
$$L(w_0, w_1) = \frac{1}{N}\sum_i \left(y_i - f_w(x_i)\right)^2 \textbf{ or } \frac{1}{N}\sum_i (y_i - (w_0 + x_i * w_1))^2$$

- **Iterative method** for a **computational solution**
  i. Start with some initial parameters e.g., $(w_0 = 0, w_1 = 0)$
  ii. Keep changing (w_0, w_1) to reduce $L(w_0, w_1)$
  iii. Stop when desirable conditions are satisfied.
- **Questions**:
  1) What can be **initial values** of parameters $(\boldsymbol{w_0, w_1})$?
  2) How to **change** the parameter values $(\boldsymbol{w_0, w_1})$, by **how much**?
  3) How do we know if the new parameters w will **reduce** $\boldsymbol{L(w_0, w_1)}$?
  4) What are the conditions to stop the iteration?

Gradient Descent Method

- To **minimize** $L(w_0)$, we can use the following iterative method
  i. Start with an initial $(w_0 = random)$
  ii. Keep changing w_0 to reduce $L(w_0)$: $w_{next} = w_0 \pm \Delta w$
  iii. Stop when certain conditions are satisfied.

How to tell whether the **gradient** at a **point** w_0 is negative or positive?

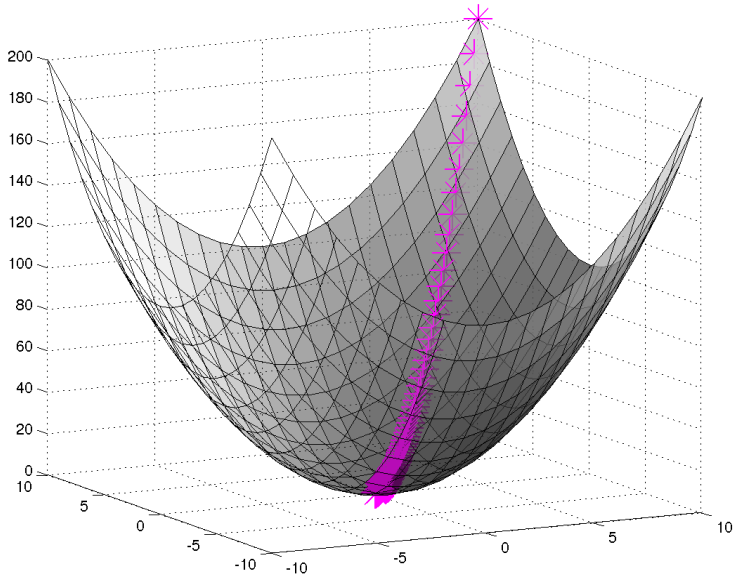


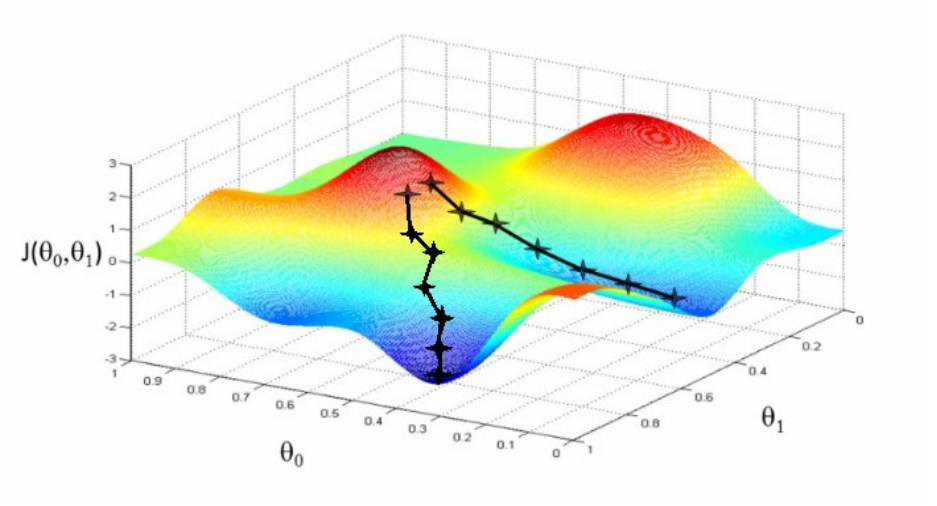
To reduce $L(w_0)$, left case: Δw should be positive, but right case: Δw should be negative.

How do we determine the **sign** of Δw (e.g., how do we change w_0, subtract or add)?

Linear Regression as a *Convex* Optimization Problem

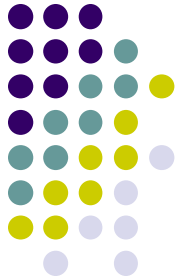
Gradient descent (also called *steepest descent*) is an **iterative optimization approach** to find the **minimum** or **maximum** of a function by taking **1st order derivative**.





Gradient descent will reach a ***global* optimum** if a function has only one optimum. Otherwise, it will reach a **local optimum**.

Finding w by Gradient Descent Method



To **minimize** $L(w_0, w_1)$ for simple regression, **minimize** $L(w_0)$ and $L(w_1)$.

To find w_0

- Initialize (w_0) with a random value
- **Repeat** updating w_0 **until** convergence

$$w_0 = w_0 - \alpha \frac{\partial L(w_0)}{\partial w_0}$$

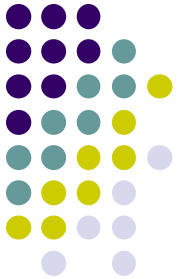
where hyper parameter α is a tuning parameter (or learning rate) to control the speed and accuracy of convergence.

To find w_1

- Initialize (w_1) with a random value
- **Repeat** updating w_1 **until** convergence

$$w_1 = w_1 - \alpha \frac{\partial L(w_1)}{\partial w_1}$$

Derivatives of $L(w_0, w_1)$ to determine $\Delta \mathbf{w}$



- **From Loss Function**

$$L(w_0, w_1) = \frac{1}{2N}\sum_i (y_i - w^T x_i)^2$$

$$= \frac{1}{2N}\sum_i (w_0 + w_1 x_i - y_i)^2$$

Why $\frac{1}{2N}$ instead of $\frac{1}{N}$?

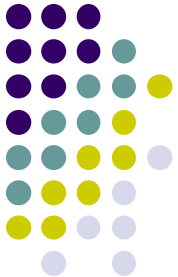
Or instead of MSE, we can use RSS = $\sum_i (w_0 + w_1 x_i - y_i)^2$

- The **partial derivatives** of $L(w_0, w_1)$ w.r.t. $\boldsymbol{w_0}$ and $\boldsymbol{w_1}$:

$$\frac{\partial}{\partial w_0} L(w_0, w_1) = \frac{1}{N}\sum_i (w_0 + w_1 x_i - y_i)$$

$$\frac{\partial}{\partial w_1} L(w_0, w_1) = \frac{1}{N}\sum_i (w_0 + w_1 x_i - y_i) x_i$$

Simple Regression by Gradient Descent Method



Initialize (w_0, w_1) with random values

Repeat until convergence {

$$w_0 = w_0 - \alpha \frac{1}{N} \sum_i (\underbrace{w_0 + w_1 x_i}_{\text{Predicted value}} - \underbrace{y_i}_{\text{Actual value}})$$

$$w_1 = w_1 - \underbrace{\alpha \frac{1}{N} \sum_i (w_0 + w_1 x_i - y_i) x_{i1}}_{\Delta \mathbf{w}}$$

}

Each (w_0, w_1) is calculated by the residual sum of all the N examples in a training data set.

- At each iteration, **update** w_0 and w_1 **simultaneously** (independently).

- A tuning parameter α is set empirically (e.g., [0.0, 1.0] as it can inflate or deflate **Δw**, impacting convergence speed and possibly accuracy.
  - Too small, slow convergence
  - Too large, fail to converge or diverge

Multiple Regression by Gradient Descent Method

$$Y = w_0 + w_1X_1 + w_2X_2 + w_3X_3 + \ldots + w_kX_k \ (X_0 = 1)$$

Initialize (w_j) with k+1 random values (k number of features)

Repeat until convergence {

for each w_j (j=0, …, k) in parallel

$$w_j = w_j - \alpha \frac{1}{N} \sum_i (w^T x_i - y_i) x_{ij}$$

(update each $\boldsymbol{w_j}$ **simultaneously** (independently)

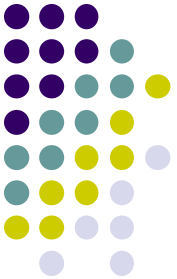
}

Each w_j is calculated by the residual sum of all the **N** examples in a training data set.

For example, we can compute w_0, w_1, w_2:

- $w_0 = w_0 - \alpha \frac{1}{N} \sum_i (w^T x_i - y_i) x_{i0} \ (x_{i0} = 1)$
- $w_1 = w_1 - \alpha \frac{1}{N} \sum_i (w^T x_i - y_i) x_{i1}$
- $w_2 = w_2 - \alpha \frac{1}{N} \sum_i (w^T x_i - y_i) x_{i2}$

A Variant of Gradient Descent Method



- **Batch Gradient Descent** (we just discussed this method.)
  - At each step, use **ALL the training samples**.

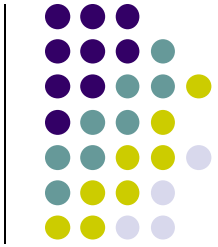
$$w_j = w_j - \alpha \frac{1}{N} \sum_i (w^T x_i - y_i) x_{ij}$$

- **Stochastic Gradient Descent**
  - Use a portion of random training samples at each iteration.
  - This is effective with a big data for an online/adaptive learning.

How to improve convergence

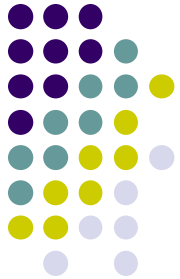
- Use more training examples
- Different learning rates α (smaller or bigger)
- Closed form solution (a normal equation using a finite number of standard operations)

Training Methods



- **Pretraining**
  - Train with the initial weights learned from some examples and fine-tune the model.
- **Sequential training**
  - The weight update for each input example. Slow convergence but may avoid local minima, potentially reaching better solutions.
- **Batch training**
  - All the training examples are fed to the network. The weight update for each epoch for the average error. Faster convergence to a local minimum since the weights are moved in the direction that most of the inputs want them to move.
- **Stochastic Gradient Descent and Minibatches**
  - Stochastic Gradient Descent (SGD)
    - The weights are updated for each one randomly selected input. Repeat the process until convergence. Often used when the training set is very large.
  - Minibatches Gradient Descent (MGD, combination of batch and SGD)
    - The training set are split into fixed-size or random batches, then update the weights for each batch.
    - The training set are then randomly shuffled into new batches and the process is repeated.
    - If the batches are small, then the global minimum may be found although at the cost of heading in the wrong direction.

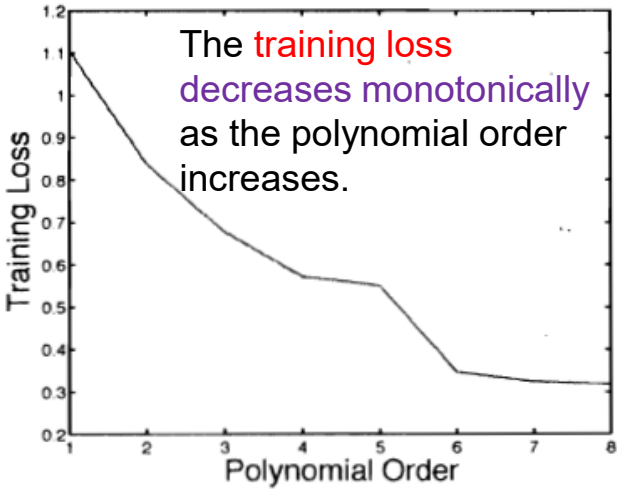
The Least Square vs. Gradient Descent

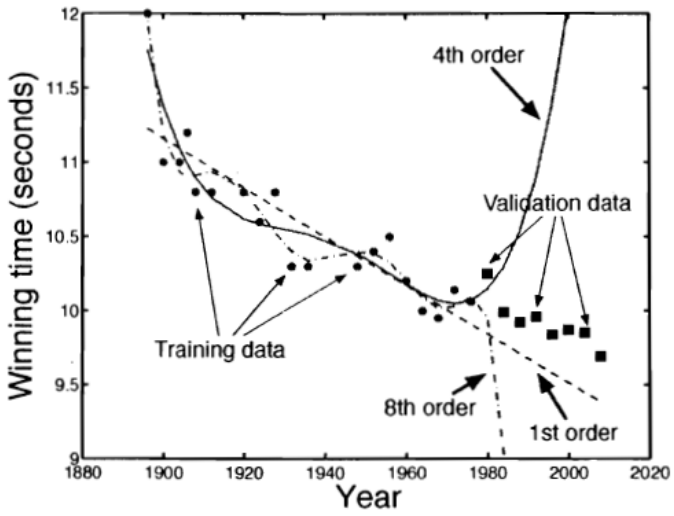


- **The least square method** (analytical solution for minimum error)
  - Very simple formula and simple implementation
  - but for big data, a large matrix requires large memory
  - but computing matrix inverse can be expensive for a large matrix
- **Gradient descent method** (iterative solution by approximation)
  - Fast even with big data (because computing is dot product of vectors)
  - *Stochastic gradient descent* is very *memory efficient*.
  - The method is easily extensible to other problems.
  - but it requires parameter tweaking:
    - How to decide convergence?
    - What is the best learning rate?
- **Most mathematical approaches for machine learning problems**
  - solved based on analytical solution (very few), estimation, approximation, and (min or max) optimization.

Validating a Regression Model

- **Training error and testing error**
  - Split a data set into training data, testing data, (optionally validation data) and measure the errors.
- **What is the best model?**
  - The model generalized ***beyond*** the training data and makes the accurate predictions on **unseen data** (validation, testing, and new data).
- **Measuring the accuracy** of a regression model
  - Training error, testing error, (validation error) using RMSE, R^2 or other measures.





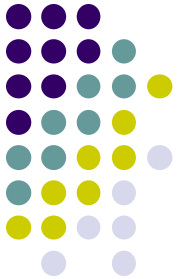
Some higher order polynomials are off from the validation data.

Why?

Over-fitting and Under-fitting

- Each model is a result of generalization process from the **training data**.
- **Over-fitting** and **Under-fitting**
  - Over-fitting: A model fits too closely to the training data (complex model), but the quality of the predictions can deteriorate rapidly.
    - **High variance**: **Low** training error and **high** generalization error, causing an algorithm to model the random noise or memorize the training data.
    - Occam's Razor: Given two models with the same generalization errors, the simpler model is preferred over the more complex model.
  - Under-fitting: A model cannot adequately capture the underlying structure of the data.
    - **High bias**: **High** training error and **high** systemic error, causing an algorithm to miss the relevant relations between the independent and dependent variables.
- **Bias-variance tradeoff**
  - **Tradeoff** is to determine the optimal model complexity such that it is able to **generalize well** without over-fitting.
  - This is a challenging task!

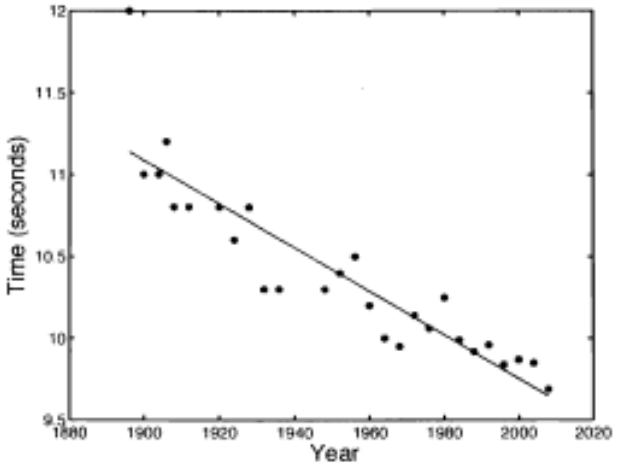
Solving Underfitting and Overfitting Problems



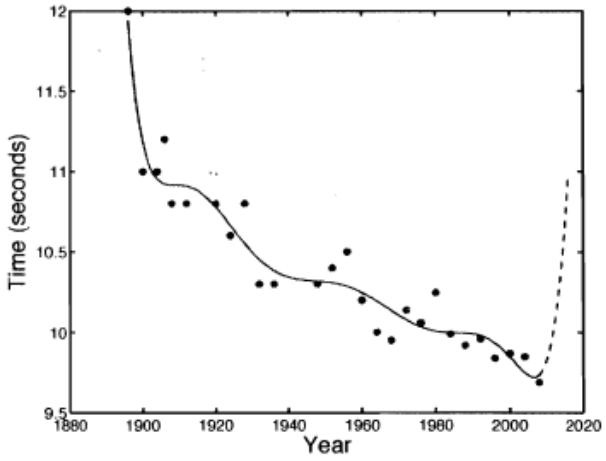
- **Common causes: recommended solutions**
  - Data quality and random noise: preprocess the data
  - Lack of representative examples: use more training data
    - How much data are appropriate? A rule of thumb for statistically reliable result: At least 5 training examples for each dimension and >30 examples
    - Generate more training data using semi-supervised learning or generative method
  - Too many variables: reduce the dimension and select the core features
  - Algorithms with poor generalization capability: use a better algorithm
- **Common causes: recommended solutions of underfitting**
  - Too simple models: make the models complex
- **Common causes: recommended solutions of overfitting**
  - Too complex models: simplify the models (regularization), early stopping, dropout, pruning, Bayesian priors, etc. and evaluate its performance on unseen data (testing/validation data).

Complex Models

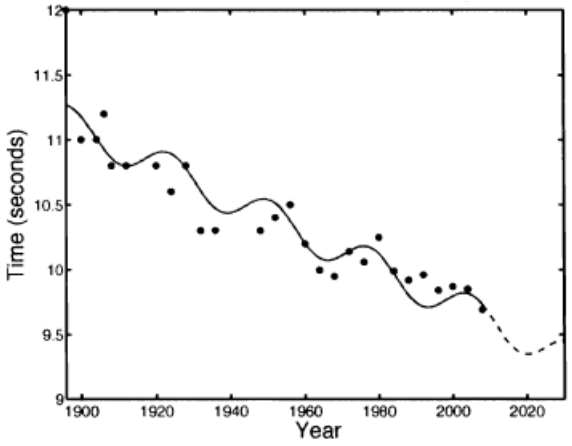
(a) A linear model with L = 1.358



(b) An 8th order polynomial model with L = 0.459

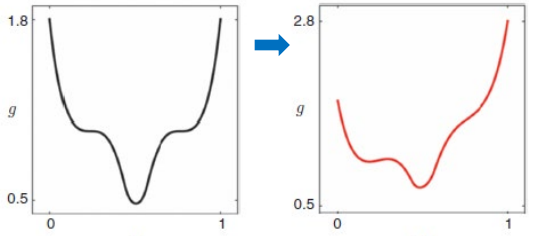


(c) A customized model with L = 1.1037



- **What's going on with higher order models?**
  - The models with increasing complexity (higher order model) fit better the data and result in lower RSS values (*low error in training*) and the W values of some variables get larger as it gets closer to the training data (*possibly learning even the noise data*), but *some prediction may be out of the range* even small deviations in the new examples (not properly generalized).
  - **To correct this problem**, add a penalty term in the error function (regularization).

Regularizing Models



Effect of convex regularizing

- ***Regularizing*** a model to **prevent over-fitting**:

  M(w) + λR(w) where λR(w) is a regularized term (penalty term or shrinkage factor);

  λ can be any real value >0; w are shrinking coefficients.

  - Why penalize the magnitude of coefficients w?

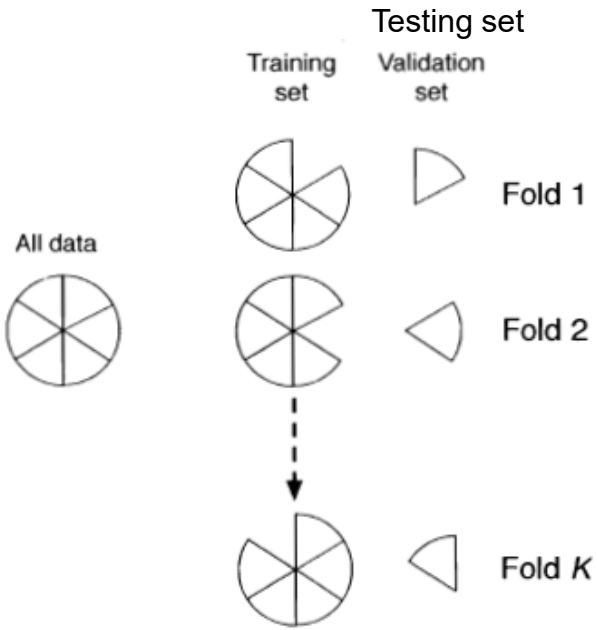
- **Regularizing** $L(w) = \frac{1}{N}\sum_i \left(y_i - f_w(x_i)\right)^2$ or $L(w) = \text{RSS(w)}$
  - LASSO (Least Absolute Shrinkage and Selection Operator):

    $L'(w) = L(w) + \lambda \sum |w_i|$ ($\lambda||w_i||_1$, L1 regularization)
  - Ridge: $L'(w) = L(w) + \lambda \sum {w_i}^2$ ($\lambda||w_i||_2$, L2 regularization)
  - Elastic net: LASSO + Ridge

- **The role of the tuning parameter λ**
  - If λ = 0, linear regression (without regularization).
  - The larger λ, more aggressively penalization is, the coefficient is close to zero, resulting in a simpler model.
  - λ should be chosen wisely (using cross-validation) as **it controls** the tradeoff between penalizing not fitting the data and penalizing overly complex models.

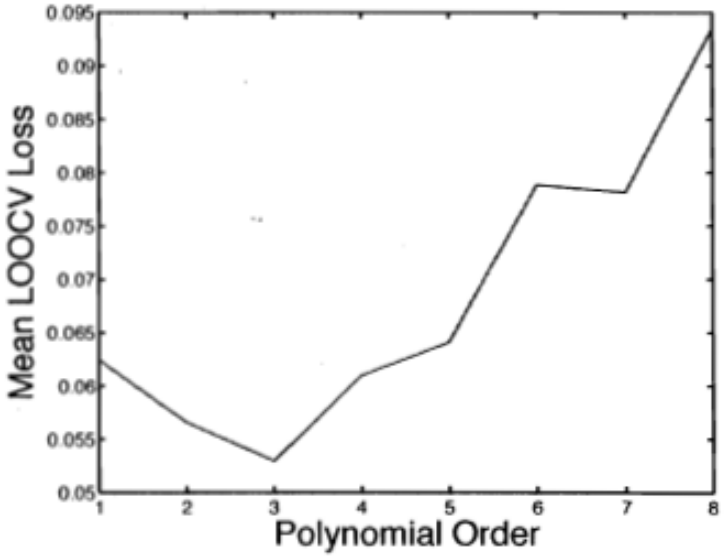
Cross-Validation (CV) for Evaluation

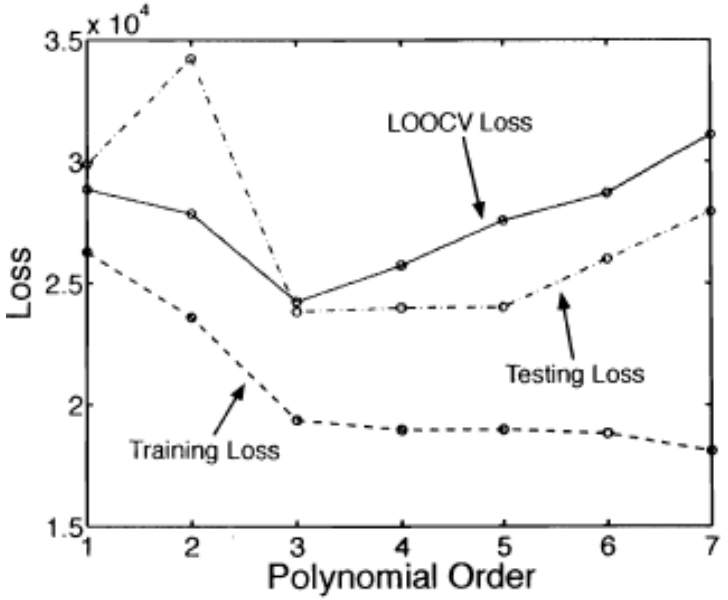
- **CV** is a **resampling method** that uses different portions of the data for training and testing.



- **Steps of K-fold cross-validation**
  - Shuffle the data set randomly and split it into k groups.
  - For each group in k groups
    1) Take the group as test data set and the remaining groups for training data (k-1 groups).
    2) Run the learning algorithm and evaluate the model on the test data set.
    3) Retain the evaluation score and discard the model.
  - Summarize the results using model evaluation scores (e.g., total or mean score).
- **Configuration of k**
  - The k value is chosen such that each training/testing group of data is large enough to be statistically representative of the data set.
  - K ≈ 5-10, k=10 known to be good through experimentation (generally low bias a modest variance).
  - K=N, known as “Leave One Out CV” (LOOCV).
  - Other variations: stratified CV, repeated CV, nested CV.

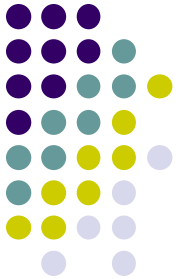
Example: Model Evaluation using LOOCV



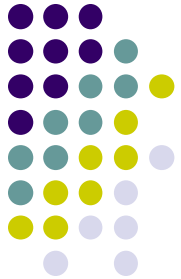


- Left graph: **Mean LOOCV loss** as polynomials of increasing order are fitted to the data.
- Right graph: The **training loss** keeps **decreasing** as the **order increases**.
- The LOOCV loss and the test loss decreases as the **order increased to 3** and then increase as the order is increased further.

Model Evaluation and Selection

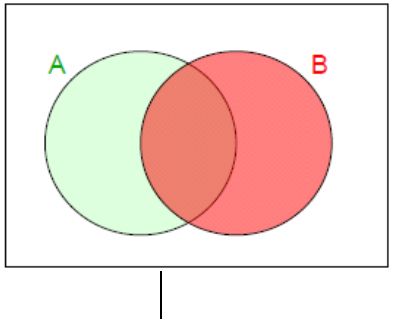


- Choose the **best model** through the statistical analysis and model comparison.
- Some **criteria to compare models**
  - Accuracy, Number of parameters, Goodness-of-fit, Estimation of a confidence interval => best predictive power
- **How to evaluate the accuracy**
  - % of correct/incorrect predictions (error)
  - The value of loss
    - Mean Squared Error (MSE), RMSE (Root MSE), Mean Percentage Error
  - Mallow's C_p = MSE + $2\sigma^2 P/N$
    - σ is the error variance; P is the number of predictors (features); N is the number of examples.
    - $\mathbf{C_p}$ is used when the data is too small using the entire data set.
  - Generalization error incorporating the model complexity



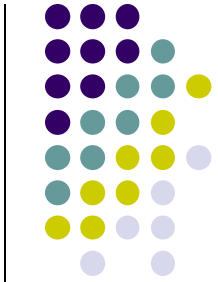
Review of Relevant Mathematics

Fundamentals of Probability Theory



- **Why probability in machine learning?**
  - The world is a very uncertain place!
  - One of the best way to estimate a future event and **quite successful**.
- **Random variables** and **probability axioms**
  - A random variable X represents uncertain outcomes under a domain.
  - $P(X = x)$, $P(x)$, or $Pr(x)$ is the probability that the value or event x is observed.
  - Probability mass function $\boldsymbol{p(x)}$ for **discrete** variables: $0 \leq p(x) \leq 1$, $\Sigma p(x) = 1$
  - Probability density function $\boldsymbol{f(x)}$ for **continuous** variables: $\int f(x)dx = 1$, $\int_b^a f(x)dx \geq 0$
- **Fundamental rules**
  - Addition: $P(A \cup B) = P(A)+P(B) - P(A \cap B)$
    - If events A and B are **independent** (disjoint), $P(A \cup B) = P(A)+P(B)$
  - Conditional probability: $P(A|B) = \frac{\boldsymbol{P(A,B)}}{\boldsymbol{P(B)}}$
    - If events A and B are **independent**, $P(B|A) = P(B)$ or $P(A|B) = P(A)$
  - Joint probability: $P(A,B) = P(A)P(B|A) = P(B)P(A|B)$
    - If events A and B are **independent**, $P(A,B) = P(A)P(B)$

Probability, Likelihood, and Odds



- **Probability:** $P(X = x)$
  - A measure of an outcome x of X (data) with the known model parameter θ, computing the ratio of occurrence of an outcome.
    - E.g., assuming θ = 0.5, what is the probability of getting 3 heads in 10 tosses?
    - The probability axioms hold, $0 \leq p(x) \leq 1$, $\Sigma p(x) = 1$
- **Likelihood** (first coined by **Fisher**): $L(\theta|X)$
  - A measure of how well a model parameter θ explains the observed data or the probability of observing the data under the assumption (θ is unknown) that the model parameter θ is true, $f(X|\theta)$ or $P(X|\theta)$.
    - If a coin landed head 3 times in 10 tosses, what is the likelihood that the coin is fair (θ=0.5)? Likelihood is **quantitative measure**. The probability axioms do not hold.
  - **Why is the likelihood thinking useful?**
    - We can ask a question in terms of the maximum likelihood estimate like "For **which model parameter** does the observed data have the biggest probability?"
- **Odds**
  - The ratio between the probability in **favor** (p) and the probability **against** it (1-p)

    odds = $\frac{p}{1-p}$ or p : (1-p)
    - A **quantitative measure** in [0, ∞], therefore, the probability axioms NOT hold.

Probability Distribution for Discrete Variables

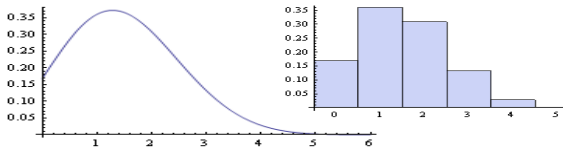
$P(T)$

T	P
warm	0.5
cold	0.5

$P(W)$

W	P
sun	0.6
rain	0.1
fog	0.3
meteor	0.0

- **Probability distribution** for random variables *T, W*
  - Probabilities in a table, graph, or in a function
  - $P(W = rain) = 0.1, T(cold) = 0.5$
- **Joint distributions** for variables *T, W*
  - $P(t_1, .., t_n)$, e.g., P(hot, sun) = 0.4



Probability distribution in graphs for continuous and discrete variables

- **Marginal distributions** *P(T)* and *P(W)* from *P(T,W)*
  - Marginalizing (summing out) rows, eliminating some variables

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

$P(t) = \sum_w P(t, w)$

$P(T)$

T	P
hot	0.5
cold	0.5

$P(w) = \sum_t P(t, w)$

$P(W)$

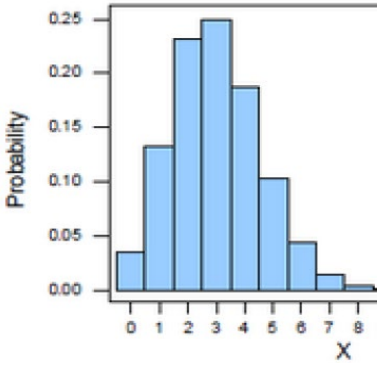
W	P
sun	0.6
rain	0.4

$P(T, W)$

T	W	P
hot	sun	0.4
hot	rain	0.1
cold	sun	0.2
cold	rain	0.3

- Mean, Variance, and Standard deviation
  - Mean: $\mu = \Sigma x p(x)$, variance: $\sigma^2 = \Sigma (x-\mu)^2 p(x)$, standard deviation: $\sqrt{\sigma^2}$

Binomial and Bernoulli Distributions (for discrete variables)



- **Binomial distribution**
  - A random variable X can take on one of **two possible values {0,1}**.
    - Multinomial, if more than two values.
  - The Probability Mass Function (**PMF**) computes the probability of getting k successes in n **independent trials** (k=0,1,…,n) and the probability of success, p:

$$\mathrm{P(X = k)} = \binom{n}{k} p^k (1-p)^{n-k} \text{ where } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

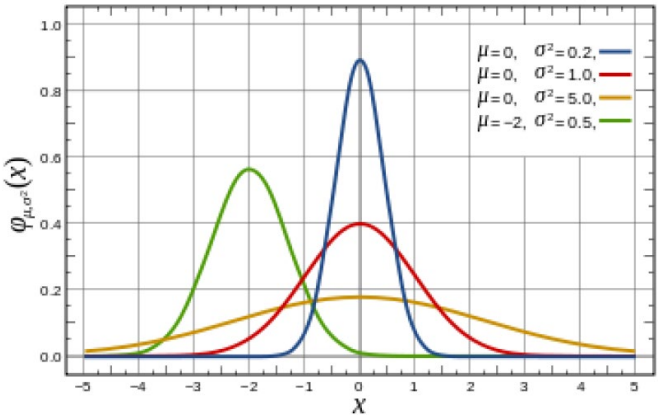
    - With the PMF, you can get the probability directly from the function.
- **Bernoulli distribution**
  - A special case of Binomial distribution for only single trial (**n=1, k={0,1}**).
  - PMF computes the probability of success (k=1):

$$P(X = k) = p^k (1-p)^{1-k}$$

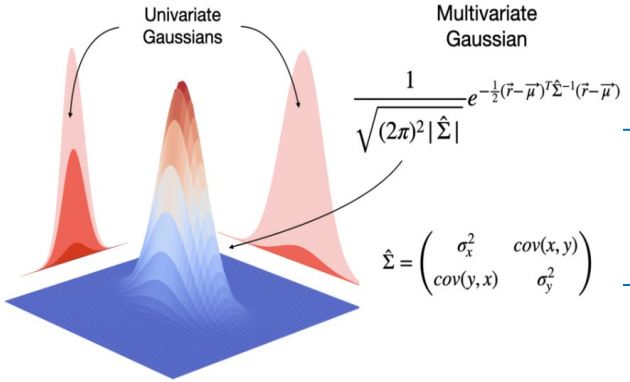
- **Multinomial distribution** (generalization of binomial)
  - PMF: $P(X = x) = \frac{N!}{\prod_i x_i!} \prod_i q_i^{x_i}$ where q_i are the parameters; $\sum q_i = 1; x = [x_1, \ldots, x_N]^T$
    - X is a random variable representing a document; x can be considered as a vector of word count x_i for a document where each document contain N words $\sum x_i = N$.

Gaussian (Normal) Distribution (for continuous variables)

Single variable



Multiple variables



The law of large numbers

The central limit theorem

- The **probability density function** (**PDF**):

$N(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, standard normal distribution: μ = 0 and σ^2=1

  - PDF for multivariate Gaussian is shown above figure for multiple variables.
  - PDF provides the likelihood that the value of a random variable will fall between a certain range of values.

- **Calculating a probability**
  - The probability of the outcome is the area of the range ($\int_b^a f(x)dx$).

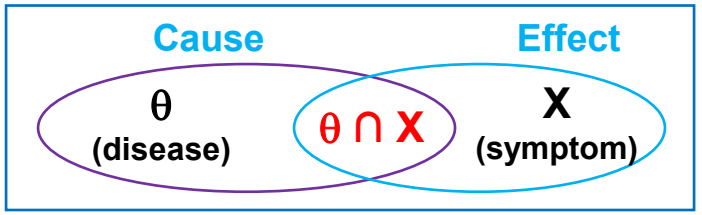
Bayes' Rule and Bayesian Approach

Bayes, 1763

- **Bayes' rule**
  - From $P(X|\theta) = \frac{P(X,\theta)}{P(\theta)}$, $P(X,\theta) = P(X|\theta)P(\theta)$

$$P(\theta|X) = \frac{P(\theta,X)}{P(X)} = \frac{P(X,\theta)}{P(X)} = \frac{P(X|\theta)P(\theta)}{P(X)}$$

  - The Bayes rule: $P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$
    - θ is a hypothesis or parameter to find.
    - **X** is the evidences or data.
- **Why is Bayes' rule useful?**
  - Often one conditional is tricky to compute but the other conditional can be simple.
  - Bayes' rule lets us build one conditional from its reverse.
    - Simple idea but amazingly powerful!
    - Foundation of many **practical** systems



Example

Given symptoms $\boldsymbol{X}$, what is the probability of having the disease θ, $P(\theta|X)$?

In many cases, given symptoms (effect) X, finding the disease (cause) θ, that is, computing $P(\theta|X)$ is difficult.

Often, it is easier to compute the probability of disease, $P(\theta)$ and the probability of observing the symptoms X given the disease θ, $P(X|\theta)$.

Elements of Bayes' Rule

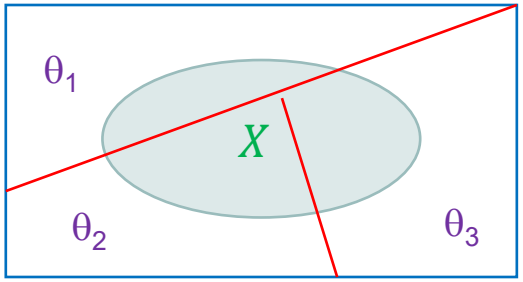
$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

$$Posterior = \frac{Likelihood * Prior}{Marginal}$$

- **Posterior probability**: P(hypothesis | evidence) or **P(θ|X)**
  - given the evidence, The probability of the hypothesis, P(flu | "fever")
- **Likelihood**: P(evidence | hypothesis) or **P(X|θ)**
  - Given that the hypothesis is true, the likelihood of the evidence.
    - If the evidence is consistent with hypothesis, it will increase the posterior probability, P("fever" | flu).
- **Prior probability**: P(hypothesis) or **P(θ)**
  - The probability of the hypothesis prior to any evidence, based on **prior knowledge**, P(flu).
- **Marginal likelihood**: P(evidence) or **P(X)**
  - The sum of all probabilities related to the evidence fever from all the other hypotheses.
    - New evidence will update the posterior probability.

Marginal Likelihood in Bayes' Rule

- **Marginal likelihood**
  - Assume that the entire sample space and event X within it, are **partitioned** by **the set of disjoint hypotheses** θ ($\theta = \cup\theta_i$).



  - $P(X) = P(\cup_i P(\theta_i, X)) = P(\theta_1, X) + P(\theta_2, X) + \cdots + P(\theta_i, X)$
    - since **hypotheses** are **disjoints** and partition X.
  - By the *total probability theorem* $P(X) = \sum_i P(X|\theta_i)P(\theta_i)$
    - $P(\theta, X) = P(\theta)P(X|\theta)$ from $P(X|\theta) = \frac{P(\theta,X)}{P(\theta)}$

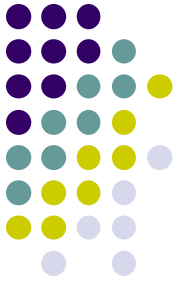
- **Bayes' rule**

$$P(\theta_i|X) = \frac{P(X|\theta_i)P(\theta_i)}{P(X)} = \frac{P(X|\theta_i)P(\theta_i)}{\sum_j P(X|\theta_j)P(\theta_j)}$$ where ***i*** is one of j

Maximum Likelihood Estimation (MLE)

- **Likelihood**: $P(X|\theta)$
  - How well the model parameters θ explain the observed data X, giving the probability of the observed data X given θ.
- **Likelihood function**: $L(\theta|X)$
  - A function of the model parameters θ given the observed data X in terms of the likelihood $P(X|\theta)$, a quantitative measure (instead of probability) of how well the parameter values explain the observed data.
- **Maximum Likelihood Estimation (MLE)**: $\hat{\theta}_{MLE} = argmax_{\theta} L(\theta|X)$
  - Seeks to find θ that maximize the likelihood function $L(\theta|X)$.
  - **Log-Likelihood**: $LL(\theta|X) = \log L(\theta|X)$
  - $\hat{\theta}_{MLE} = argmax_{\theta} LL(\theta|X)$
  - **Why log?**
    - Multiplying numbers [0,1] many times → 0 as those numbers go to infinity (underflow)
    - The logarithm function monotonically increases.
    - **Maximizing** a **likelihood** ≡ **maximizing** the **log** likelihood.
    - Taking derivative of log is easier!

MLE Example



- **A coin tossing game**
  - There were **4** heads (h) and **6** tails (t) out of 10 flips.
  - We don't know if the coin is fair. Estimate the probability of the coin θ using MLE.

- **Computing $\widehat{\theta}_{MLE}$**

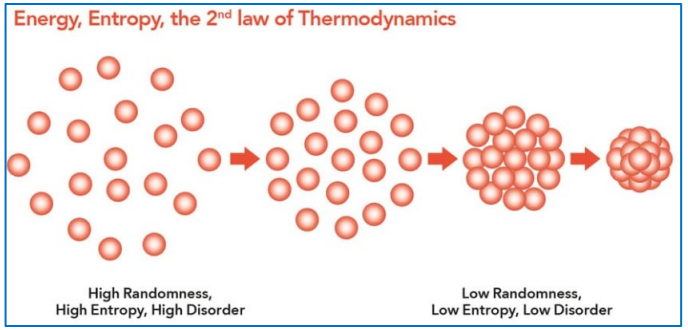
  Using the Binomial distribution as the likelihood $P(X|\theta)$ objective function:

  $$L(\theta|X) = \prod P(x_i|\theta) = \theta^h(1-\theta)^t \; where \; \theta = P(h);\; (1-\theta) = P(t)$$

  Solving the log likelihood MLE:

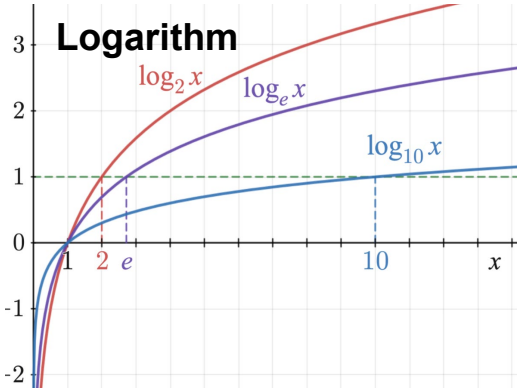
  $$\theta_{MLE} = argmax_\theta \prod_i P(x_i|\theta) = argmax_\theta log P(x_i|\theta)$$
  $$= argmax_\theta log[\theta^h(1-\theta)^t]$$
  $$\frac{\partial}{\partial\theta}[log\,\theta^h(1-\theta)^t] = \frac{\partial}{\partial\theta}[hlog\theta + tlog(1-\theta)] = \frac{h}{\theta} - \frac{t}{1-\theta} = 0$$
  $$\hat{\theta}_{MLE} = \frac{h}{h+t} = \frac{4}{4+6} = 0.4$$ (that proves your intuition is correct)

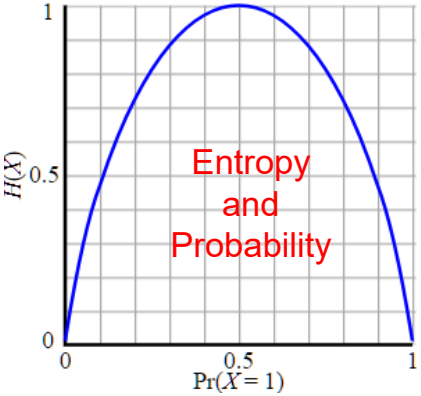
Uncertainty and Entropy



- **Entropy in physics**
  - If the particles inside a system have many possible positions to move around, the system has high entropy.
    - e.g., water in solid state (low entropy), water in gas state (high entropy)
- **Entropy in information theory** (Claude Shannon, 1916-2001)
  - Entropy is a measure of the uncertainty associated with a random variable.
    - It quantifies the average amount of information (or surprise) one should expect when observing a value drawn from a probability distribution
  - How much information is acquired from the observation of a random event?
    - That depends on how much information the event carries.
    - The more information it carries, the higher entropy.
  - Higher entropy indicates greater uncertainty or randomness.
  - Lower entropy indicates less uncertainty, more predictability.
- **Entropy and knowledge**
  - The more we know, the less uncertain the event (variable) is.

Shannon's Entropy





- **Entropy H(X)**
  - Entropy of a discrete random variable X with probability mass function P(X) is given:

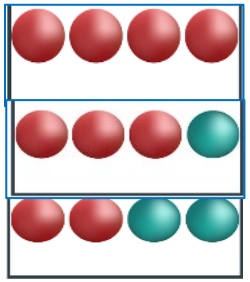
    $\boldsymbol{H(X) = \sum_{i=1}^{N} P(x_i) I(x_i)}$ where Information content $\boldsymbol{I(x_i) = -logP(x_i)}$

    $\boldsymbol{H(X) = -\sum_{i=1}^{N} P(x_i) logP(x_i)}$
    - Why negative log?
  - Interpretation:
    - The expected (average) number of bits needed to encode an outcome from the random variable X. The more unpredictable the outcome is, more bits needed.
    - **Shannon's entropy** is measured in **binary bits** (the log base 2).

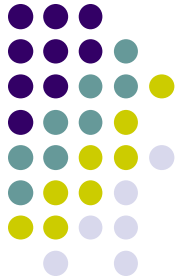
- **Properties of H(X):** $0 \leq H(X) \leq \log(1/N)$
  - Higher entropy when outcomes are smoothly spread.
    - The maximum entropy is **log(1/N)** when every outcome has equal chance, $p(x) = 1/N$.
  - Lower entropy when outcomes are concentrated.
    - The minimum entropy is **zero** when $p(x) = 1$, log(1)= 0 or $p(x) \approx 0$ (log(0) is undefined).

Examples: Entropy

- **Fair coin tossing**
  - For a fair coin tossing, what is the entropy for the coin tossing event X?
    - The possible values of X: h(head) or t(tail)
    - p(h) = 0.5. p(t) = 0.5. $\mathbf{H(X) = -\sum_{i=1}^{N} p(x_i) \log_2 p(x_i)}$
    - H(X) = $-p(h)\log_2 p(h) - p(t)\log_2 p(t) = -1/2\log_2(1/2) - 1/2\log_2(1/2)$ = **1 bit**.
    - It means that **one bit** is required to encode the message of the coin tossing.
- **Rigged coin tossing**
  - For a rigged coin tossing with 75% chance of seeing heads, what is the entropy for the coin tossing event X?
    - Now, we know more about the coin, p(h)=3/4 and p(t)=1/4.
    - H(X) = $-3/4\log_2(3/4) - 1/4\log_2(1/4)$ = **0.811 bits**.
- **Three boxes with colored balls**
  - What is the entropy of each box?



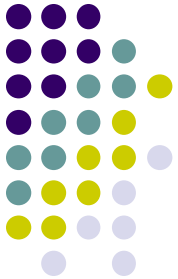
Example: Entropy of a Training Data Set



Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

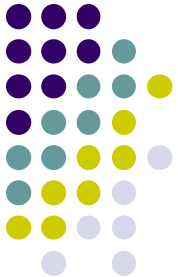
- What's the **entropy** of a decision whether "playing golf" or not?
  - What's the **entropy** of classification from this training data set?
    - P(Yes)=9/14=0.64, P(No)=5/14=0.36
    - H(PlayGolf) = –[0.64*log0.64+0.36*log0.36] ≈ 0.94

Measuring Impurity (non-Homogeneity)



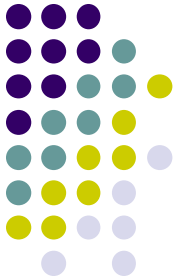
- Pure or homogeneous data set
  - If it contains only a single class. Otherwise, the data set is impure or heterogenous.
- **Several indices to measure the degree of impurity**
  - **Entropy**: $-\sum_{i=1}^{N} p(c_i) \log_2 p(c_i)$ (used in ID3 algorithm)
  - Gini index: $1 - \sum_{i=1}^{N} p(c_i)^2$ (used in CART algorithm)
  - Classification error index: 1 – max{$p(c_i)$}
  - Other measures:
    - Chi-squared statistic: a measure of independence between groups.
    - Reduction in variance between two groups.
  - Can be used in decision tree induction or data summarization
- **Examples**
  - For a data set with 3 classes and $p(C_1)=0.4$, $p(C_2)=0.3$, $p(C_3)=0.3$
  - Entropy: $-(0.4\log 0.4+0.3\log 0.3+0.3\log 0.3) = 1.571$
  - Gini index: $1 - (0.16^2+0.09^2+0.09^2) = 1 - 0.0418 = 0.9582$
  - Classification error index: 1 – max{0.4, 0.3, 0.3} = 1 – 0.4 = 0.60

Cross-entropy



- Cross-entropy: $H(P,Q)$
  - $\boldsymbol{H(P,Q) = -\sum_i P(x_i) log Q(x_i)}$
    - The difference (divergence) between two probability distributions P and Q for a **given random variable X**
    - A quantitative measure of the dissimilarity between two probability distributions
    - The average number of bits needed to encode data coming from a source with distribution P (true distribution) when use model Q (approximation of the true or predicted distribution).
    - If Q is close to P, the cross-entropy is small. If Q diverges from P, the cross-entropy is larger.
- The Kullback-Leibler divergence (or relative entropy): $D_{KL}(P||Q)$
  - $D_{KL}(P||Q) = \sum_i \boldsymbol{P(x_i)}\, log(\frac{\boldsymbol{P(x_i)}}{\boldsymbol{Q(x_i)}})$
    - The number of additional bits needed to represent an event using Q instead of P.
    - If P and Q are identical, $D_{KL}(P||Q) = 0$.
  - $H(P,Q) = H(P) + D_{KL}(P||Q)$
    - Minimizing H(P,Q) with respect to Q ≡ minimizing $D_{KL}(P||Q)$

Examples: Cross-entropy

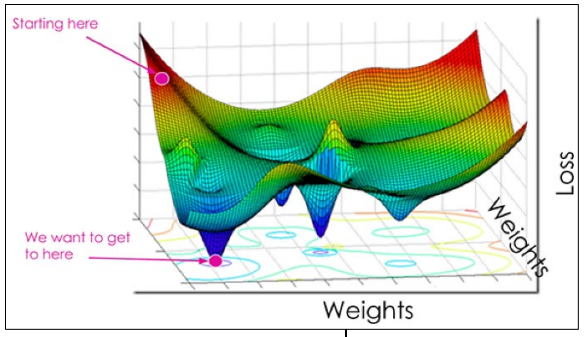


- Entropy and cross-entropy
  - Assume the following probability distributions:

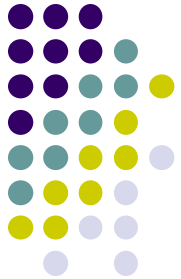
    P = [0.3, 0.5, 0.2]

    Q = [0.4, 0.4, 0.2]
  - **Entropy:** $H(P) = -\sum_i P(x_i) log P(x_i)$
    - H(P) = –[0.3log0.3 + 0.5log0.5 + 0.2log0.2] ≈ 1.4855, H(Q) ≈ ?
    - H(P) value represents the average number of bits needed to encode an event drawn from the distribution P.
  - **Cross-entropy:** $H(P, Q) = -\sum_i P(x_i) log Q(x_i)$
    - H(P,Q) = –[0.3log0.4 + 0.5log0.4 + 0.2log0.2] ≈ 1.3219 bits
    - This value represents the average number of bits needed to encode events from P using the optimal code for Q.
- The Kullback-Leibler divergence
  - $D_{KL}(P||Q) = \sum_i P(x_i)\, log(\frac{P(x_i)}{Q(x_i)})$
    - $D_{KL}(P||Q)$ = 0.3log($\frac{0.3}{0.4}$) + 0.5log($\frac{0.5}{0.4}$) + 0.2log($\frac{0.2}{0.4}$) ≈ -0.1245+0.1609-0.2 = -0.1635

Cross-entropy Loss



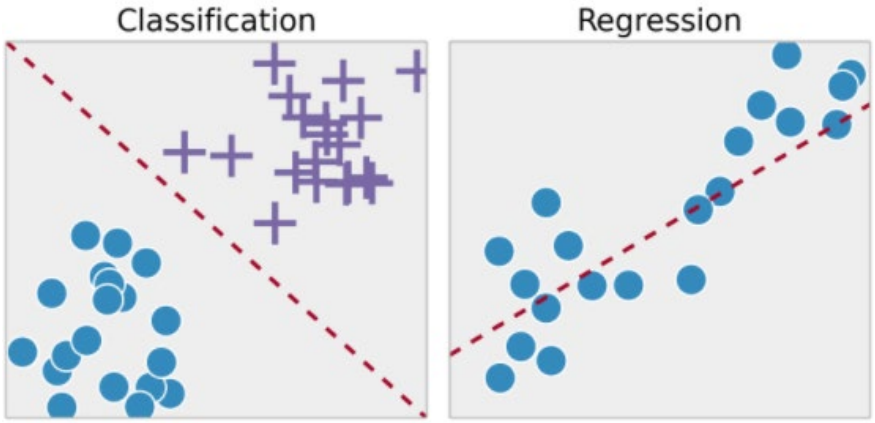
- **Various uses**
  - Measuring the difference between predicted and actual values in **classification**
  - How well one probability distribution predicts another in recommended systems
  - Information extraction and encoding-decoding in NLP
- **Minimizing cross-entropy $H(y, \hat{y})$**
  - Find θ that minimizes $H(y, \hat{y})$ to optimize classification models
    - $H(y, \hat{y})$ measures the difference between actual class y (probability distribution) and predicted class $\hat{y}$ (probability distribution) for a given example x.
    - $H(y, \hat{y})$ is minimized when both y and $\hat{y}$ are identical.
    - Better than the squared sum in classification as it leads to faster training as well as improved generalization
- **Cross-entropy loss functions**
  - For binary class
    - y is the true label (0 or 1) and $\hat{y}$ is the predicted label [0,1] calculated in probability
    - For y=1: $H(y = 1, \hat{y}) = -y log \hat{y}$
    - For y=0: $H(y = 0, \hat{y}) = -(1 - y) \log(1 - \hat{y})$
    - Combining both cases: $H(y, \hat{y}) = -[y log \hat{y} + (1 - y) \log(1 - \hat{y})]$
  - For multi-class: $H(y, \hat{y}) = -\sum_i y_i \log(\hat{y_i})$



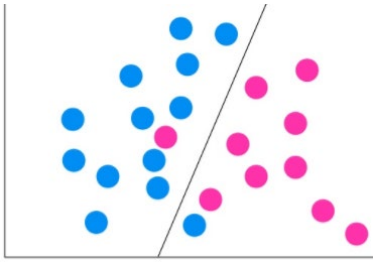
Logistic Regression (LR) for Classification

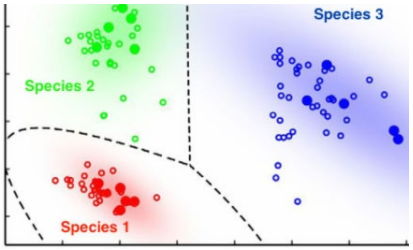
Supervised Learning for Classification

- Given a training data set $\{(X_1, X_2, \ldots, X_k, Y_j) | j = 1 \ldots N\}$
  - Learn a function or classifier $f: X \rightarrow Y$
    - The process of learning **f** is called training.
  - Predict outcomes for a given X (**new data**):
    - If the learned classifier f is used to determine a discrete value, $\hat{y}$, it's called classification.
- **Classification vs. regression**
  - Classification is to find the *decision boundary* that **separates** the different groups of data as clearly as possible.
  - Regression is to find the *function* that **fits** the data the best.



Types of Classification

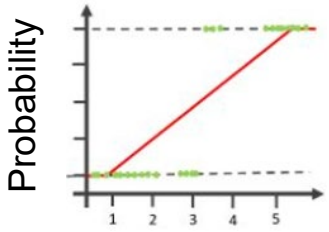




- **Types of classification**
  - Binary and multiclass (by number of classes)
  - Linear and nonlinear (by the type of boundary function)
  - Probabilistic vs non-probabilistic
  - Parametric vs non-parametric
    - Parametric needs the fixed parameters about the data, so makes assumptions (specific form) while non-parametric does not require the parameters.
  - Discriminative vs generative
    - Discriminative learns parameters that maximize $P(y|X)$ while generative learns parameters that maximize the joint distribution $P(X, y)$.
- **Numerous applications**
  - Pattern recognition, document classification, language modeling, etc.

Probability and Log(odds)

- **Scenario**: Given x number of years in service, will the computer breakdown?



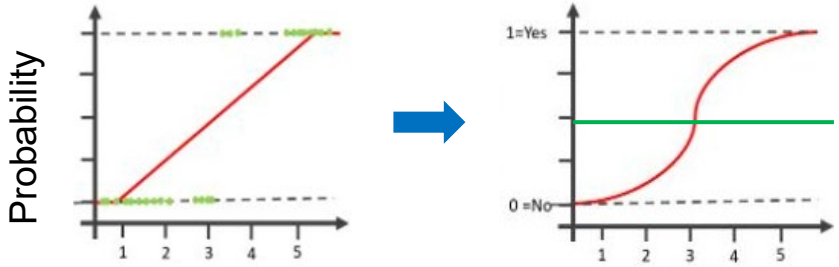
We can answer by a probability of breakdown for a given number of years of service: $y = w^T x \Rightarrow p(x)$.

Probability	Odds	Log Odds
0.100	0.111	-2.197
0.200	0.250	-1.386
0.300	0.428	-0.847
0.400	0.667	-0.405
0.500	1.000	0.000
0.600	1.500	0.405
0.700	2.333	0.847
0.800	4.000	1.386
0.900	9.000	2.197

- **Problems** of y-values in terms of **probability**
  - Expressing $w^T x$ on y=**[0,1]** is extremely difficult since $w^T x$ is unbounded.
  - How about taking log? **Logarithms** are unbounded, but in only one direction.
  - How about probability equivalent "**Odd**"=$\frac{p(x)}{1-p(x)}$? still only one direction $(0, \infty)$
  - **Log(odds)** = $ln(\frac{p(x)}{1-p(x)})$ defined on $(-\infty, \infty)$, finally unbounded! Now y= $ln(\frac{p(x)}{1-p(x)})$
    - **Example**: For 6 wins and 10 losses, the probability of win 6/16=0.375; Odd of win =6/10=0.6; Log(odd) of win, log(0.6) = -0.222
  - The general form of log(odds) is logit function: Logit(p) = $ln(\frac{p}{1-p})$
- **Advantages of using Log(odds)**
  - Symmetrical property, allowing fair comparison – useful property for classification
  - No restriction on data sampling – easy to update with new data

Logistic Function for Probability

- **Deriving the logistic function**
  - Let $y = ln\left(\frac{p(x)}{1-p(x)}\right)$, then $\left(\frac{p(x)}{1-p(x)}\right) = e^y$. Solve for $p(x) = e^y(1-p(x))$
  - $p(x) = e^y - e^y p(x)$, $p(x)(1+e^y) = e^y$, $p(x) = \frac{e^y}{1+e^y}$
  - By dividing both sides by e^y we get: $p(x) = \frac{1}{1+e^{-y}}$ and $1 - p(x) = \frac{e^{-y}}{1+e^{-y}}$.
- **The idea of using logistic function for classification**
  - Replacing y by $w^T x$, $\boldsymbol{p(x) = \frac{1}{1+e^{-w^T x}}}$, $\boldsymbol{1 - p(x) = \frac{e^{-w^T x}}{1+e^{-w^T x}}}$, $\boldsymbol{p(x)}$ = [0,1].
    - The name "logistic regression" is from the logistic function.
  - **Classification** $\boldsymbol{P(y=1|w,x) = \frac{1}{1+e^{-w^T x}}}$ when $\boldsymbol{w^T x}$>=0 otherwise $\boldsymbol{y = 0}$



Logistic and Sigmoidal Function

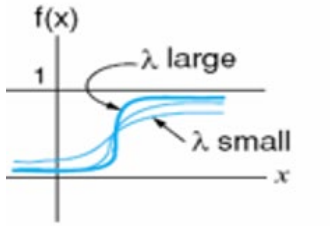
Logistic function is "S" shape, non-linear, and continuous function:

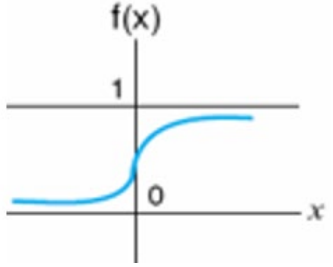
$$f(x) = \frac{L}{1+e^{-\lambda(x-x_0)}}$$

L is the curve's maximum value; e is the natural logarithm; λ is the steepness of the curve; x_0 is the x-value of the midpoint. When **L=1**, it becomes a sigmoidal function.

Sigmoidal function:

$$f(x) = \frac{1}{1+e^{-\lambda x}}$$

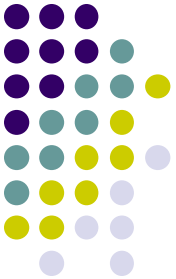




Usefulness of logistic or sigmoidal function:

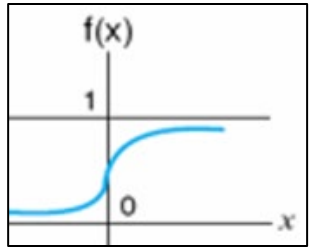
- The value of f(x) is normalized [0,1] – can be used for a probability value.
- A continuous function is differentiable at each point.
- The value of derivative is greatest where the sigmoidal function is changing most rapidly.

Recipe for Supervised Learning

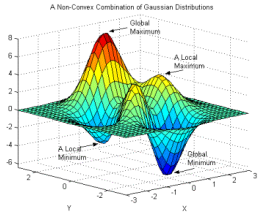


1. Given a training data set: $\{(x_i, y_i), \dots, (x_N, y_N)\}$
2. Define an objective or decision function: $\hat{y}_i = f_\theta(x_i)$
   - Is there any analytical solution? If yes, use it, If no, use iterative approach.
3. Define the goal (min or max: min is common) using the objective function and train it with (stochastic) gradient descent (or other available methods):
   - Define a loss function: $l(\hat{y}, y) \in \mathbb{R}$ (Log loss is commonly used)
   - Find $\theta^* = argmin_\theta \sum_{i=1}^{N} l(\hat{y}_i, y_i)$
4. Learn the model θ^* using:
   - Gradient: $\nabla l(\hat{y}, y)$ (by taking the derivative of the objective function)
   - Update rule: $\theta^{t+1} = \theta^t - \alpha \nabla l(f_\theta(x_i), y_i)$
5. Predict $\widehat{y_{new}}$ for x_{new} using $f_{\theta^*}(x_{new})$

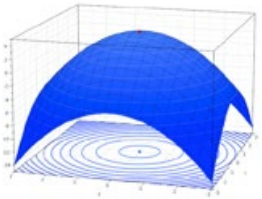
Objective Function

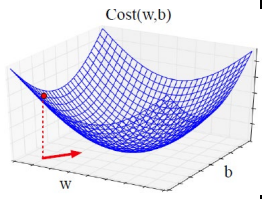


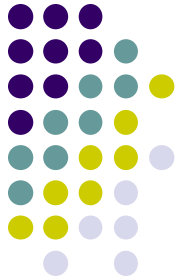
- Defining an objective function for ***binary classification***
  - Bernoulli mass function, $P(Y = y) = p^y(1-p)^{1-y}$ where y is 1 or 0
    - Note: $P(Y = y)$ is maximum when $Y = 1$ or $Y = 0$.
  - We can estimate the probability p using the logistic function $\hat{y} = \frac{1}{1 + e^{-\hat{w}^T x}}$
  - Objective function: $P(Y = y|X) = \hat{y}^y(1-\hat{y})^{1-y}$
- The goal with the objective function
  - Find w for $\hat{y}$ that maximizes $P(Y = y|X)$ given X.
  - $\hat{w}_{MLE} = argmax_w P(Y = y|X)$ or $\hat{w}_{MLE} = argmax_w log P(Y = y|X)$
- Defining log-likelihood function: $\boldsymbol{LL(w|X)}$
  - $\hat{w}_{MLE} = argmax_w LL(w|X) = y log\hat{y} + (1-y)\log(1-\hat{y})$
  - Is there an analytical solution?
    - Bad news: No closed-form solution, transcendental equation, cannot compute W directly
    - Good news: $\boldsymbol{LL(w|X)}$ is a concave function – easy to optimize using **approximation methods** such as gradient descent (ascent), Newton-Rapson, or Metropolis-Hastings (Markov Chain Monte Carlo).



Cross-entropy Loss







- **Log loss function:** $-LL(w|X)$
  - $\widehat{\boldsymbol{w}}_{\boldsymbol{MLE}} = \boldsymbol{argmin}_{\boldsymbol{w}} - LL(w|X) = -[ylog\hat{y} + (1-y)\log(1-\hat{y})]$
    - Two loss functions: $log\hat{y}$ when $Y = 1$ and $\log(1-\hat{y})$ when $Y = 0$
  - Why loss function?
    - Minimization is preferred since most algorithms are implemented in terms of cost or error reduction for optimal solutions, e.g., the least square, gradient-descent, also gives a lower-bound for optimal.
- **Cross-entropy loss:** $H(y,\hat{y})$ or $l(y,\hat{y})$
  - $H(y,\hat{y}) = -LL(w|X) = -(ylog\hat{y} + (1-y)\log(1-\hat{y}))$
    - Many mathematical concepts and models are not independent (instead connected and continuous like natural phenomenon).
  - Recall the cross-entropy discussed earlier:
    - For binary class, $H(y,\hat{y}) = -[ylog\hat{y} + (1-y)\log(1-\hat{y})]$
    - For multi-class, $H(y,\hat{y}) = -\sum_i y_i \log(\hat{y_i})$
  - Minimizing the cross-entropy is equivalent to find w that minimizes the log loss function, making the actual y and predicted $\hat{y}$ as close as possible.

Training in Logistic Regression

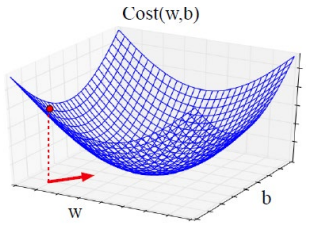
- **Deriving a gradient and update rule**
  - From the loss function $H(y, \hat{y}) = -LL(w|X) = -(ylog\hat{y} + (1-y)\log(1-\hat{y}))$ where $\hat{y} = \frac{1}{1+e^{-\hat{w}^T x}}$
  - The derived gradient (The detailed derivation steps are skipped.)

    $\frac{\delta H(y,\hat{y})}{\delta w_j} = [\hat{y} - y]x_j = (\hat{y} - y)x_j$ where x_j is the value of jth feature of x.

    $w_j = w_j - \alpha \frac{\delta H(y,\hat{y})}{\delta w_i}$, $w_j = w_j - \alpha(\hat{y} - y)x_j$ (We have seen this before.)

    $\frac{\delta H(y,\hat{y})}{\delta w_j} = \frac{1}{N}\sum_{i=1}^{N}[\hat{y_i} - y_i]x_{ij} = \frac{1}{N}(\hat{y} - y)^T X$ (using the entire data set)

  - Update rule: $w_j = w_j - \alpha \frac{1}{N}[\sum_{i=1}^{N}(\hat{y_i} - y_i)x_{ij}]$



- **Training**
  - Start with an initial w (randomly generated)
  - Compute $\hat{y}$
  - Adjust w using the update rule for $\hat{\boldsymbol{w}}_{\boldsymbol{MLE}} = \boldsymbol{argmin}_{\boldsymbol{w}} l(y, \hat{y})$

Example: Classification using LR

- **Sentiment classification**
  - Suppose we have the following text for a movie review. The review document x has 6 features with the sentiment class + or –.

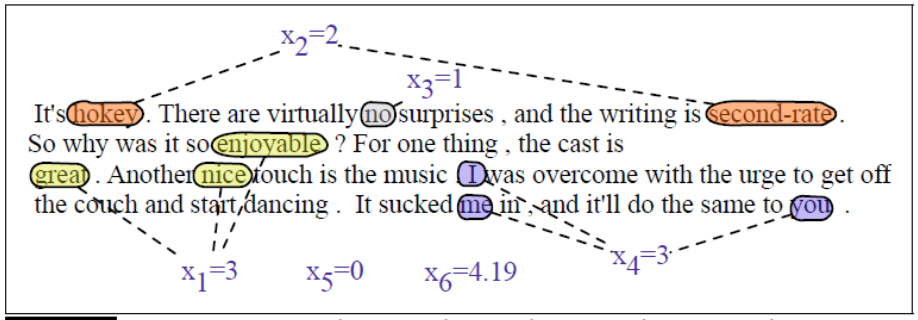
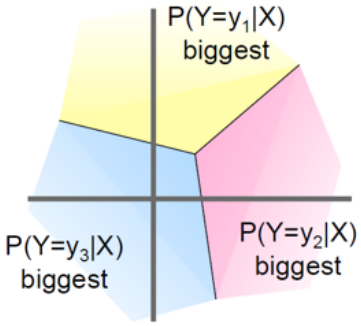


Figure 5.2 A sample mini test document showing the extracted features in the vector x.

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words ∈ doc)	3
x_2	count(negative lexicon words ∈ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns ∈ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

- **Assume** the estimated weights $\widehat{w}^T$ = [0.1, 2.5, -5.0, -1.2, 0.5, 2.0, 0.7].
- **For the input review** x, $p(+|x)$ and $p(-|x)$ can be computed:
  - x = [1, 3, 2, 1, 3, 0, 4.19]^T (1 is added for w_0)
  - $p(+|x) = P(y = 1|x) = \sigma(w^T x) = \sigma(0.833) = 0.70$
  - $p(-|x) = P(y = 0|x) = 1 - \sigma(w^T x) = 1 - \sigma(0.833) = 1 - 0.70 = 0.30$
  - Is it a positive sentiment based on $\widehat{w}_{MLE}$?

Multinomial Logistic Regression

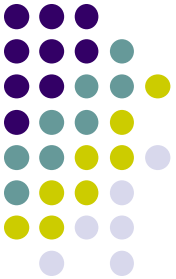


- **Logistic Regression for multi-class**
  - An extension of LR for binary classification to handle multiple classes
- **Multinomial Logistic Regression using Softmax**
  - Training data set $\{(X_1, X_2, \dots, X_k, Y_j) | j = 1 \dots N\}$ where Y_j has m ≥ 2 class values
    - Y_j can be one-hot encoding, e.g., for 3 class, [1,0,1], [0,1,0], [0,0,1]
  - M × (k+1) weights matrix, W
    - M is the total number of classes.
    - A weight vector W_m for each class m
  - Compute the linear combination for each class: $z_m = W_m^T x$
  - **Apply Softmax function** to get a class probability (not logistic function):

$$P(Y = m|x) = \hat{y} = softmax(z_m) = \frac{exp(z_m)}{\sum_{m=1}^{M} exp(z_m)}$$ (normalized e^z)

    - $softmax(z) = [\frac{exp(z_1)}{\sum_{m=1}^{M} \exp(z_m)}, \frac{exp(z_2)}{\sum_{m=1}^{M} exp(z_m)}, \dots, \frac{\exp(z_M)}{\sum_{m=1}^{M} exp(z_m)}]$ (probability distribution $\sum z = 1$)
    - We still choose only one class (with the highest probability) as the predicted class.
    - This is why it is also called Softmax regression.
  - Example: z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]
    - Softmax(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]

Multinomial Logistic Regression



- **Computing the class probability**
  - $P(Y = m|x) = \hat{y} = softmax(z_m) = \frac{exp(z_m)}{\sum_{m=1}^{M} \exp(z_m)}$
- **Gradient to update the weights**
  - The cross-entropy loss function is generalized from **2** to **M classes**, thus y_i for x_i is a vector.

    $H(\hat{y}, y) = P(Y = c|x) = -log \frac{\exp(w_c^T x)}{\sum_{m=1}^{M} \exp(w_m^T x)}$ where correct class, $y_c = 1, y_m = 0 \; \forall_m \neq c$

    For each class m, the gradient of w_j (jth element of input x):

    $\frac{\delta L(\hat{y},y)}{\delta w_{m,j}} = -(y_m - \hat{y}_m)x_j = -(y_m - P(y_m = 1|x))x_j = -(y_m - \frac{\exp(w_m^T x)}{\sum_{m=1}^{M} \exp(w_m^T x)})x_j$
- **Training**
  - Start with an initial w (randomly generated) for each class
  - Apply the Softmax function
  - Learn W_m that minimizes the cross-entropy loss using gradient descent
    - Note: All the weight vectors associated with each class are updated, not only the one with the highest predicted probability.
  - Repeat the process until converged

Binary LR vs Multinomial LR

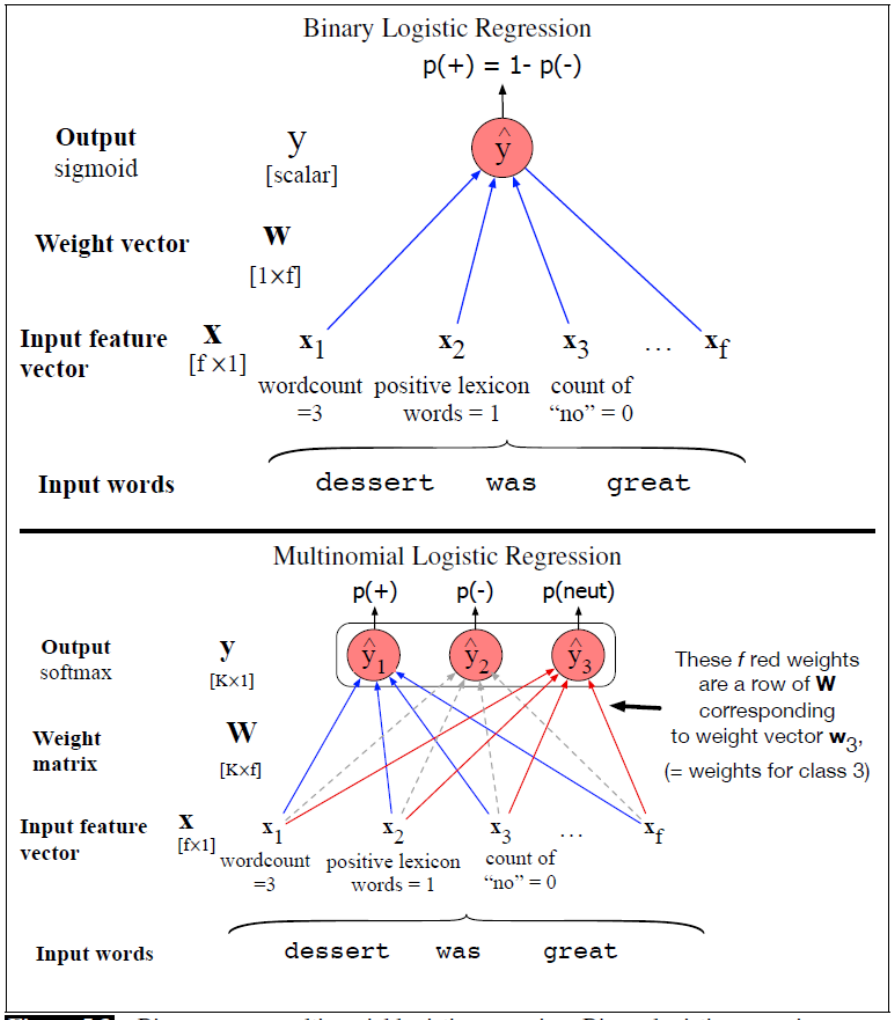
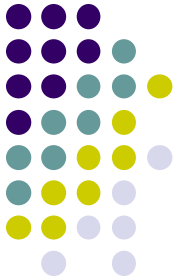


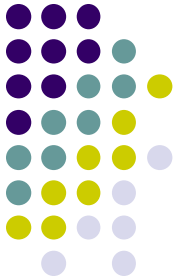
Figure 5.3 Binary versus multinomial logistic regression. Binary logistic regression uses a single weight vector **w**, and has a scalar output $\hat{y}$. In multinomial logistic regression we have K separate weight vectors corresponding to the K classes, all packed into a single weight matrix **W**, and a vector output $\hat{\mathbf{y}}$.

LR Regularization to Avoid Overfitting



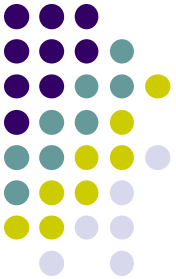
- Three major components of linear methods:
  - Loss function, regularization, and algorithm
- **Adding a regularization term to Log likelihood**
  - LL(θ, y, x) + λR(θ), R(θ) can be L1 or L2
- L1 (LASSO) regularization
  - $\hat{\theta} = argmax_\theta \sum_{i=1}^{N} logP(y_i|x_i) - \lambda||\theta||_1^1$
    - Note: The penalty term is **subtracted** because of argmax.
- L2 (Ridge) regularization
  - $\hat{\theta} = argmax_\theta \sum_{i=1}^{N} logP(y_i|x_i) - \lambda||\theta||_2^2$

Using Prior Knowledge



- **Strength of MLE** $\boldsymbol{\theta_{MLE}} = argmax_\theta P(X|\theta) = argmax_\theta \prod_i P(x_i|\theta)$
  - MLE gives the explanation of the data you observed.
  - That means if the model distribution is correct with enough data, MLE finds the true parameters.
- **Some problems of MLE**
  - MLE favors complex models that may overfit if the data size is small.
    - e.g., suppose you observe all five heads. What is $\hat{\theta}_{MLE}$?
  - MLE can be wrong if the model is not correct.
- What if I have **prior knowledge** $P(\theta)$ or **obtained more information later?**
  - Better idea: We can obtain a distribution over all possible values of θ rather than estimating a single θ.

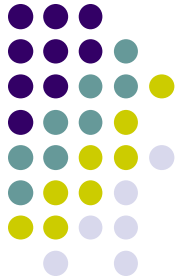
Maximum a Posteriori (MAP)



- **Estimating θ using the posterior probability $\boldsymbol{P(\theta|X)}$** in the Bayes rule.
- $\hat{\theta}^{MAP} = argmax_\theta P(\theta|X) = argmax_\theta \frac{P(X|\theta)P(\theta)}{\boldsymbol{P(X)}} \propto \boldsymbol{argmax_\theta P(X|\theta)P(\theta)}$
  - Seeks to find θ that maximize the $P(\theta|X)$
  - Maximizing $P(\theta|X) \propto \boldsymbol{P(X|\theta)P(\theta)}$
    - How to estimate the prior probability $\boldsymbol{P(\theta)}$?
    - For binary classification, the most common form of prior distribution $P(\theta)$ is a Beta distribution:

      $P(\theta) = Beta(\beta_0, \beta_1) = \frac{\theta^{\beta_1 - 1}(1-\theta)^{\beta_0 - 1}}{B(\beta_0, \beta_1)}$ where are β_0, β_1 are parameters.
  - This is generative thinking as estimation of $P(\theta|X)$ is based on the probability distribution $\boldsymbol{P(X|\theta)P(\theta)}$ = joint distribution $\boldsymbol{P(\theta, X)}$.

- **MLE vs MAP**
  - Both can be used for estimating model parameters.
  - **MAP** seeks to find the model parameters θ that maximize $P(\theta|X)$ the posterior probability given the observed data X using likelihood $P(X|\theta)$ and prior $P(\theta)$ while **MLE** seeks to find θ that maximizes $P(X|\theta)$ the likelihood of observing X.



Bayesian Classification

The Bayes Classifier

- **Bayes classifier**

$$P(Y = C_j|X) = \frac{P(X|Y = C_j)P(Y = C_j)}{P(X)} = \frac{P(X|Y = C_j)P(Y = C_j)}{\sum_{m=1}^{M} P(X|Y = C_m)P(Y = C_m)}$$

  - Two popular choices for **prior** class distribution
    - Uniform prior $P(Y = C_j) = 1/M$ where M is the number of classes (can be dropped)
    - Class size prior $P(Y = C_j) = C_{jN}/N$ where N is the # of examples in the training data and C_{jN} is the # of examples belonging to class C_j in the data set.
  - Marginal likelihood can be ignored.
  - Posterior probability can be estimated by Likelihood*Prior.
  - Therefore, classification is defined as a probability distribution over class C.
- **Training**:
  - For each class C_j, estimate $P(X|C_j)$ and $P(C_j)$:
    - If x is categorical, just count values.
    - If x is continuous, either discretize or estimate predetermined distribution such as Gaussian.
- **Prediction**:
  - For a new example x_{new}, choose the class C_j that **maximizes $P(x_{new}|C_j)P(C_j)$**.

Bayesian Classification

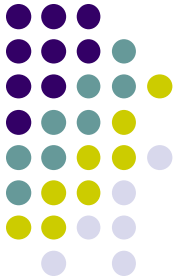
Name	Gender
Drew	Male
Claudia	Female
Drew	Female
Drew	Female
Alberto	Male
Karin	Female
Sergio	Male
Nina	Female

- **Bayesian classification**
  - Given a training data X with y as M possible classes C, the Bayes classifier assigns an example x in X a probability of a class C_j to using the Bayes rule:
  - $P(Y = C_j | x, X) = \frac{P(x, X|Y = C_j)P(Y=C_j)}{P(x,X)} = \frac{P(x, X|Y = C_j)P(Y=C_j)}{\sum_{m=1}^{M} P(x, X|Y = C_m)P(Y=C_m)}$
  - **MAP** classification: $\hat{c} = argmax_{c \in C} \frac{P(X|c)P(c)}{P(X)}$
    - The example x is classified into the class with the highest posterior probability.

- **Example:**
  - An officer arrested a person named "Drew" on a crime scene. Is the person "Male" or "Female"?
  - **Answer**: Let C_1=male, C_2=female, and x=drew in the data set.
    - $P(\text{male} \mid \text{drew}) = \frac{p(\text{drew}|\text{male})p(\text{male})}{p(\text{drew})} = \frac{1/3*3/8}{\frac{1}{3}*\frac{3}{8}+\frac{2}{5}*\frac{5}{8}} = \frac{1/8}{3/8}$
    - $P(\text{female} \mid \text{drew}) = \frac{p(\text{drew}|\text{female})p(\text{female})}{p(\text{drew})} = \frac{2/5*5/8}{\frac{1}{3}*\frac{3}{8}+\frac{2}{5}*\frac{5}{8}} = \frac{2/8}{3/8}$

By Bayesian classification, the **person** is more likely to be female.

Strength and Weakness of Bayesian Classification

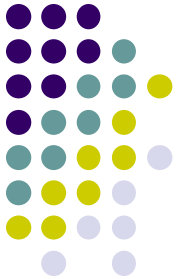


- **Strengths**
  - Can estimate the **posterior distribution** from likelihood and prior distributions for many cases (conjugate pairs).
    - The posterior density can explicitly **model** the uncertainty can be used to **make predictions**.
  - Can **combine** the **data observed** with **prior knowledge** in a principled way.
  - Diminishing effect of prior probability as more data are added.
  - The marginal likelihood shows the effect of the new data.
- **Weaknesses**
  - Require probability distribution that is unknown in many other cases
  - Probability calculation can be expensive as the size of data increases.
    - **Joint distribution** for model parameters with **K features**, for posterior, $argmax_Y P(Y|X_1, X_2, \dots, X_k)$, the likelihood, $P(X_1, X_2, \dots, X_k|Y)$

    $P(X_1, X_2, \dots, X_k|Y) = P(X_1, X_2, \dots, X_k, Y) = P(X_1|X_2, \dots, X_k, Y)P(X_2, \dots, X_k, Y)$

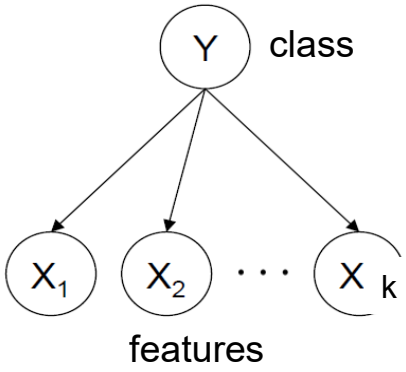
    $= P(X_1|X_2, \dots, X_k, Y)P(X_2|X_3, \dots, X_k, Y)P(X_3, \dots, X_k, Y) = \dots$

    $= P(X_1|X_2, \dots, X_k, Y)P(X_2|X_3, \dots, X_k, Y)P(X_3, \dots, X_k, Y) \dots P(X_{k-1}|X_k, Y)P(X_k|Y)P(Y)$



The Naïve Bayes Classifier

The Naïve Bayes Classifier



- Let's assume features are independent given a class.
  - P(X_1,X_2 |Y) = P(X_1|X_2,Y)P(X_2|Y) = P(X_1|Y)P(X_2|Y)
  - More generally $P(X_1, X_2, \dots, X_k|Y) = \prod_i P(X_i|Y)$
  - $\boldsymbol{P(C_j|X_1, X_2, \dots, X_k)} \propto P(C_j, X_1, X_2, \dots, X_k) = \propto P(C_j)P(X_1|C_j)P(X_2|C_j)$

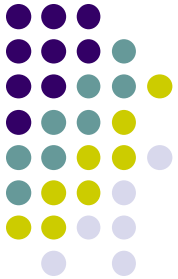
    $\dots \propto \boldsymbol{P(C_j)\prod_{i=1}^{K} P(X_i|C_j)}$

- **The Naïve Bayes classifier**
  - Given prior $P(C_j)$, K conditionally independent features $X_1, X_2, \dots, X_k$, the posterior probability by **MAP estimation**:

    $\hat{c} = \boldsymbol{argmax_{C_j} P(C_j)\prod_{i=1}^{K} P(X_i|C_j)}$ (Ignoring the marginal likelihood)

  - Are the Naïve Bayes assumptions realistic?
    - Yes, in many cases, e.g., for features of Apple(shape, color, size), each of these features contributes independently to the probability that the fruit is an apple.

How to Calculate the Likelihood



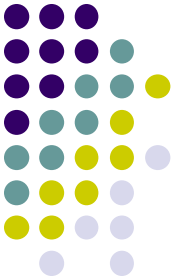
- **Calculating the likelihood**: $argmax_{C_j} P(C_j) \prod_{i=1}^{K} \boldsymbol{P(X_i|C_j)}$
  - If x is categorical, just count values.
  - If x is continuous, either discretize or estimate predetermined distribution (e.g., Gaussian, Bernoulli, multinomial, etc.)
  - Gaussian Naïve Bayes: $\mathrm{P}(x|C_j) = \frac{1}{\sigma_j\sqrt{2\pi}} \exp(-\frac{(x-\mu_j)^2}{2{\sigma_j}^2})$
    - Calculate μ_k and σ^2 in X^c (X^c is X associated with class C_k).
  - Bernoulli Naïve Bayes: $P(x|C_j) = \prod_{i=1}^{K} p_{ij}^{x_i} (1-p_{ij})^{1-x_i}$
    - Discretize the values to obtain a new set of Bernoulli-distributed features.

Example: Naïve Bayes Classification

- Will a customer with the following profile "age<30, income=medium, student=Yes" buy a computer?
- **What is the naïve Bayes answer?**
  - **P(Bought=yes)**=9/14=0.643, **P(Bought=no)**=5/14=0.357
  - P(Age<30|yes)=2/9=0.222. P(Age<30|no)=3/5=0.6.
  - P(Income=medium|yes)=4/9=0.444, P(Income=medium|no)=2/5=0.4
  - P(Student=yes|yes)=6/9=0.667,
  - P(Student=yes|no)=1/5=0.2
- **Calculated probabilities**:
  - P(X|yes) = 0.222*0.444*0.667 = 0.066.
  - P(X|no) = 0.6*0.4*0.2 = 0.048.
  - P(X|yes)P(yes) = 0.066*0.643 = 0.042.
  - P(X|no)P(no) = 0.048*0.357 = 0.017
- **Answer**:
  - By the MAP estimation, the naïve Bayes classifier predicts the customer will buy a computer.

Age	Income	Student	Bought
20	High	Yes	Yes
25	Low	No	No
50	High	No	Yes
70	Medium	Yes	Yes
15	Medium	Yes	Yes
30	High	No	No
10	Low	Yes	No
40	Medium	Yes	Yes
35	Medium	Yes	Yes
41	Medium	No	No
45	High	Yes	Yes
22	Medium	No	No
55	High	No	Yes
50	High	No	Yes

Multinomial Naïve Bayes



- **Multinomial model**
  - Multinomial is a generalization of binomial.
  - A feature vector $X = (p_1, \dots, p_k)$ represents the frequencies of certain events $(p_1, \dots, p_k)$ where p_j is the probability that an event occurs.
  - A feature vector $X = (x_1, \dots, x_k)$ is a histogram, with x_j counting the number of times event x was observed in a particular instance.
  - Popularly used for text classification and information retrieval.

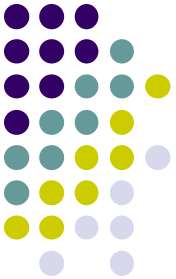
- **The likelihood of observing** x: $P(x|C_m) = \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_j p_{mj}^{x_j}$
  - M multinomial: A multinomial in each class
  - The multinomial Naïve Bayes classifier becomes a linear classifier when expressed in log-space:

$$logP(C_m|x) \propto \log\left(P(C_m)\prod_j p_{mj}^{x_j}\right) = logP(C_m) + \sum_{j=1}^{M} x_j logP_{mj} = w_m^T x$$

where $w_{mj} = logp_{mj}$

Example: Sentiment Classification

(Speech and language process by D. Jurafsky & J.H Martin, 2021)



Assumptions:

- A bag of words from all the documents (order not important).
- Conditional independence

function TRAIN NAIVE BAYES(D, C) **returns** log $P(c)$ and log $P(w|c)$

for each class $c \in C$ # Calculate $P(c)$ terms
  N_{doc} = number of documents in D
  N_c = number of documents from D in class c
  $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$
  $V \leftarrow$ vocabulary of D
  $bigdoc[c] \leftarrow$ **append**(d) **for** d $\in$ D **with** class c
  for each word w in V # Calculate $P(w|c)$ terms
    $count(w,c) \leftarrow$ # of occurrences of w in $bigdoc[c]$
    $loglikelihood[w,c] \leftarrow \log \frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$
return $logprior$, $loglikelihood$, V

function TEST NAIVE BAYES($testdoc$, $logprior$, $loglikelihood$, C, V) **returns** best c

for each class $c \in C$
  $sum[c] \leftarrow logprior[c]$
  for each position i in $testdoc$
    $word \leftarrow testdoc[i]$
    if $word \in V$
      $sum[c] \leftarrow sum[c] + loglikelihood[word,c]$
return argmax$_c$ $sum[c]$

Figure 4.2 The naive Bayes algorithm, using add-1 smoothing. To use add-α smoothing instead, change the +1 to +α for loglikelihood counts in training.

Naïve Bayes multinomial classifier

$argmax_{c \in C} P(c) \prod_{i \in positions} P(w_i|c)$ where C=all the classes; positions=all the word positions in a test document

To avoid underflow and increase speed:

$argmax_{c \in C} P(c) + \sum_{i \in positions} logP(w_i|c)$

Prior: $P(c) = \frac{N_c}{N_{doc}}$

Likelihood $P(w_i|c) = \frac{count(w_i,c)}{\sum_{w \in V} coun(w,c)}$

*Set all the words not in a class in training document to zero.

Add-α Laplace smoothing:

$P(w_i|c) = \frac{count(w_i,c)+1}{(\sum_{w \in V} coun(w,c))+|V|}$

Example: Sentiment Classification

(Speech and language process by D. Jurafsky & J.H Martin, 2021)

- **Training document**

$$P(-) = \frac{3}{5} \qquad P(+) = \frac{2}{5}$$

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

- Is a sentence **S** = "predictable with no fun" positive or negative?
  - "with" is dropped as it does not occur in the training data.

$$P(\text{"predictable"}|-) = \frac{1+1}{14+20} \qquad P(\text{"predictable"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"no"}|-) = \frac{1+1}{14+20} \qquad P(\text{"no"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"fun"}|-) = \frac{0+1}{14+20} \qquad P(\text{"fun"}|+) = \frac{1+1}{9+20}$$

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

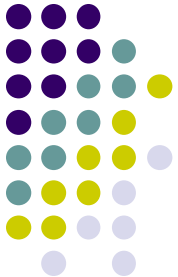
$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

UNION of all words |V| = 20
bigdoc[–] = 14, bigdoc[+] = 9

count(w, c) = # of occurrence of the word w in bigdoc[c]

- The Naïve Bayes classifier **predicts** the sentiment of the sentence **S** "negative".

Strengths and Weakness of NB

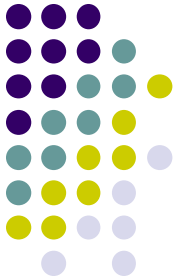


- **Strengths**
  - Highly scalable
  - Can handle both real and discrete data including streaming data
  - Not sensitive to irrelevant features and can handle missing values
    - Text document classification, SPAM filtering, this assumption works reasonably well.
- **Weaknesses**
  - When the independence assumption does not hold, it may not work.
    - e.g., NB cannot model any within-class dependencies.
  - Naïve Bayes is a linear classifier:
    - Theorem: Assume that $y_i \in \{0,1\}$. Then the Naïve Bayes classifier is defined by:

      $$h(x) = argmax_y P(y) \prod_j^M P(x_j|y) = sign(w^T x)$$

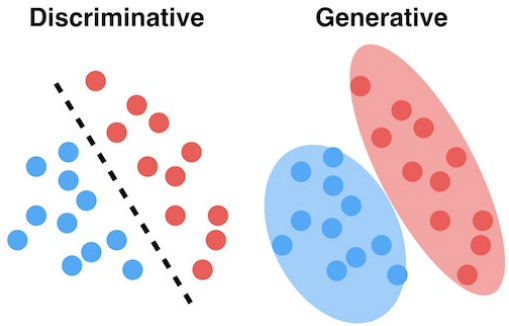
    - We can show that: $w^T x$ >0 <=> $h(x)$ = +1
  - For continuous features (Gaussian Naïve Bayes), we can use:

    $P(y|x) = \frac{1}{1+e^{-y(w^T x+w_0)}}$ (This is logistic regression.)
    - The Naïve Bayes and logistic regression produce asymptotically the same model if the Naïve Bayes assumption holds.



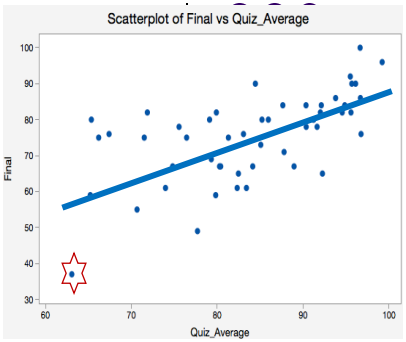
Generative Modeling

Discriminative vs. Generative



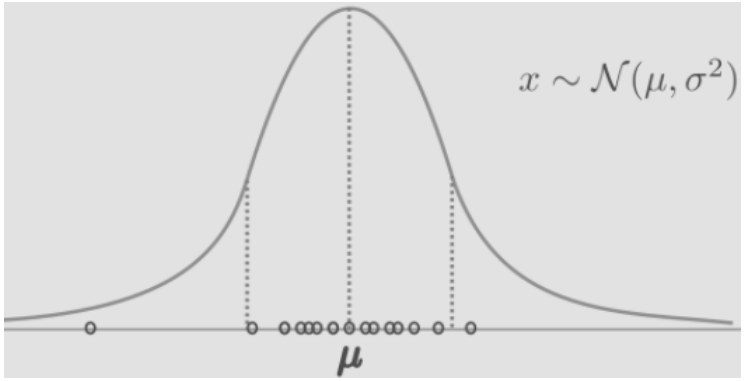
- **Generative modeling**
  - Model the original data set in order to predict the example.
  - To find P(Y|X), estimate the prior probability **P(Y)** and likelihood **P(X|Y)**, focusing on how the data X was generated that allows P(X,Y)
    - What's the likelihood of the class Y that generates X?
  - **Examples**: Probability distribution-based methods
    - Naïve Bayes, Hidden Markov Model, Gaussian mixture model, etc.
- **Discriminative modeling**
  - Learn the boundaries of classes without caring about distribution.
  - To find P(Y|X), model the conditional probability directly from the **training data**.
    - Better handle outliers
  - **Examples**: Non-probability distribution-based methods
    - Many supervised learning methods such as Logistic Regression, K-NN, SVM, ANN, etc.

Generating a Training Data Set



- **An estimated model $f(x; w) = \hat{w}^T x + \varepsilon$ with noise, $\varepsilon \sim N(0, \sigma^2)$**
  - A random variable y has the following density function:
    $P(y|x, w, \sigma^2) = N(\hat{w}^T x, \sigma^2)$
  - If we plug in the values $\hat{w}$, we can generate y values. How?
- **To get the original training data set**
  - We need to add errors ε: $f(x; w) = \hat{w}^T x + \varepsilon$
    - As errors can vary, we can generate random values roughly the same size as the errors. How?
    - Incorporate the variability σ^2 into our model.
    - Now, with $\hat{w}$ and σ^2, the likelihood of the data is highest in the distribution.

Parameter Estimation based on Probability Distribution



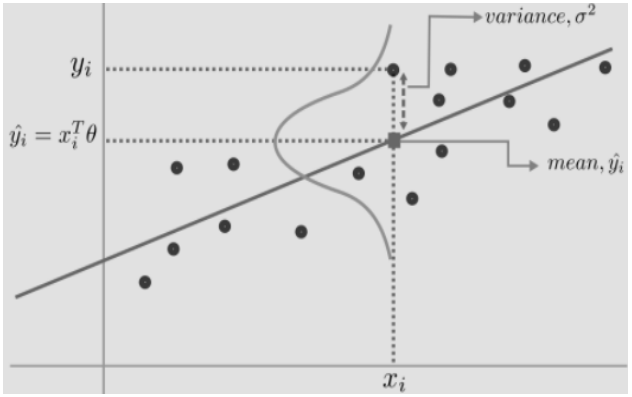
- The law of large numbers
- The central limit theorem

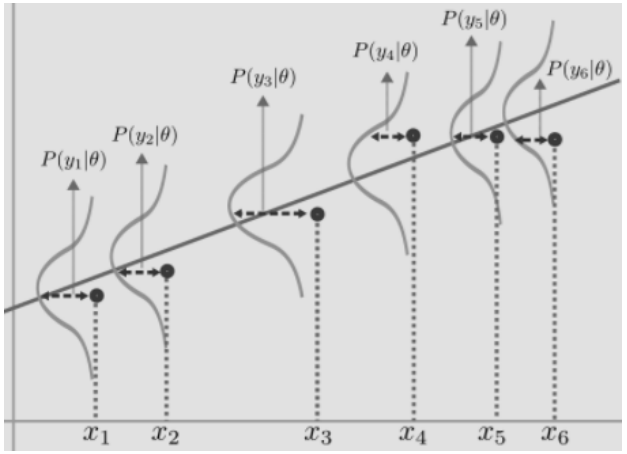
- **A date set and its probability distribution**
  - Each data point in a data set can be considered as a value of a random variable **sampled** through a stochastic process under a probability distribution.
    - If the distribution is Gaussian, most values will fall around near the mean.
- **Estimating the parameter using the given data set**
  - **Generate** the original **distribution** using the selected parameters and **confirm** the parameters by **checking if** the data **fit** the distribution well.
  - This type of method is called "**generative**" approach.

Generative Regression Modeling

$$y_i = \mathcal{N}(x_i^T\theta, \sigma^2) = x_i^T\theta + \mathcal{N}(0, \sigma^2)$$

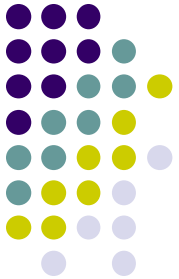
$$prediction, \hat{y}_i = x_i^T\theta$$





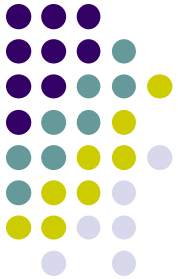
- **Idea of generative regression modeling**
  - Find $\hat{w}$ and σ^2 that maximizes the data likelihood: $\boldsymbol{P(y|x, w, \sigma^2) = N(\hat{w}^T x, \sigma^2)}$
  - Given a training data {(X, y)}, if we assume **y** is **gaussian** distributed, then the regression model will have $\mu = x_i^T \hat{w}$ and variance σ^2.
    - $x_i^T \hat{w}$ represents the best fit line.
    - The data points will vary about the line and $\mathcal{N}(\mu,\sigma^2)$ captures this variance.
- **Learning a regression line**
  - Find the **parameters** $\widehat{y_{i_{MLE}}}$ and $\widehat{\sigma^2}_{MLE}$ from **training data** that maximize the likelihood:

    $N(y_i|\mu, \sigma^2) = \frac{1}{2\pi\sigma}\exp(-\frac{(y_i-\mu)^2}{2\sigma^2})$ where $\mu = x_i^T \hat{w}$
  - $\hat{w}$ can be computed by the linear regression or the posterior probability using Bayes' rule.
- **For prediction:**
  - Use the posterior distribution for each parameter.



Assessing Classifier Performance

Confusion Matrix



- **Confusion matrix**
  - A square matrix that shows a summary of classification results in a table form.
    - The element of the matrix at *(i,j)* can indicate that # of examples predicted in class *i* but class j as the true class.
    - Diagonal elements are correct responses.
    - Accuracy = (sum of diagonal elements) / (sum of all elements in the matrix).
  - Examples:
    - For three classes, C_1, C_2, C_3, the table tells us that 1 example of class C_1 as C_2 and 2 examples of class C_1 were misclassified as C_3.

Confusion matrix form

		True class 1	True class 0
Predicted class	1	*TP*	*FP*
	0	*FN*	TN

TP: true positive, FP: false positive,
TN: true negative, FN: false negative

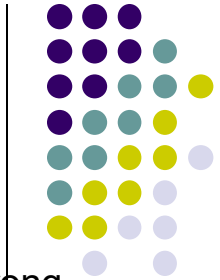
Example 1

	C_1	C_2	C_3
C_1	5	1	0
C_2	1	4	1
C_3	2	0	4

Example 2

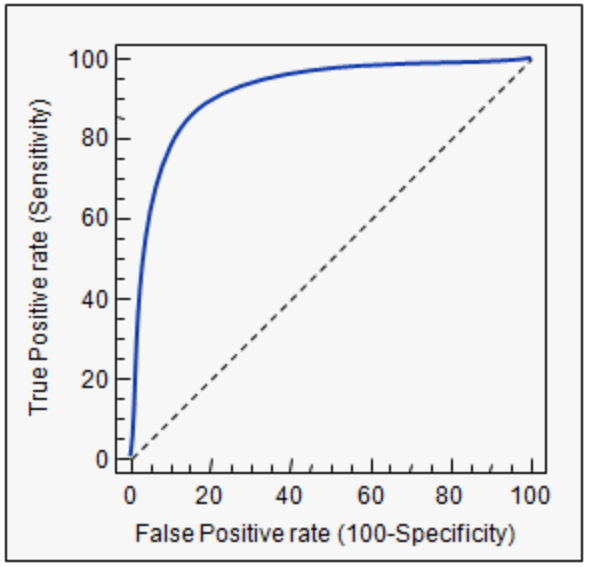
n=192	**Predicted: 0**	**Predicted: 1**
Actual: 0	118	12
Actual: 1	47	15

Evaluation Metrics



- **Accuracy by % error:** The proportion of objects for which the classifier is wrong.
  - Problem: The result may **mislead** when classes are imbalanced.
    - 1% of objects belong to class 1 (abnormal case such as card fraud, disease, etc.) while only 99% of objects belong to class 2 (normal case).
- **Alternative metrics**
  - **Sensitivity** or recall (true positive rate) r = **TP**/(**TP**+**FN**)
    - The model's ability to correctly identify positive instances
    - The higher the sensitivity is, the lower the # of FN error.
    - Useful when the cost of false negatives (missed positive instances) is high.
    - If a model assigns every example to the positive class, what will be the value of sensitivity?
  - **Specificity** (true negative rate) s = **TN**/(**TN**+**FP**)
    - The model's ability to correctly identify negative instances
    - Useful when the cost of false positives (false alarms, missed negatives) is high.
  - **Precision** p = **TP**/(**TP**+**FP**)
    - The higher the precision is, the lower the # of FP error
  - **Harmonic mean** F1= sensitivity + precision
    - High value of F1 ensures that **both sensitivity** and **precision** are reasonably high.

The Receiver Operating Characteristic (ROC) Curve



- **ROC curve**
  - A graphical approach for displaying the **tradeoff** between true positive rate and false positive rate.
  - A good classifier should be located as close as possible to the upper-left corner, while a random classifier should reside along the main diagonal.
    - Useful for comparing the relative performance among different classifiers.
    - The **area** under the ROC curve provides information on which classifier is better on average.
    - If the model is perfect, its area would be 1. random guess = 0.5.