Section Zero (Supplement to Joe's Math Review)

Hanning Luo

2025-01-31

1 Calculus

1.1 Differentiation

You should be familiar with single-variable differentiation, including properties like:

$$\begin{aligned}
\text{Linearity: } & \frac{d}{dx}(af(x)+bg(x)) = af'(x)+bg'(x) \\
\text{Chain rule: } & \frac{d}{dx}f(g(x)) = g'(x)f'(g(x)) \\
\text{Product rule: } & \frac{d}{dx}f(x)g(x) = f'(x)g(x)+f(x)g'(x) \\
\text{Quotient rule: } & \frac{d}{dx}\frac{f(x)}{g(x)} = \frac{f'(x)g(x)-f(x)g'(x)}{(g(x))^2}
\end{aligned}$$

for scalars a and b.

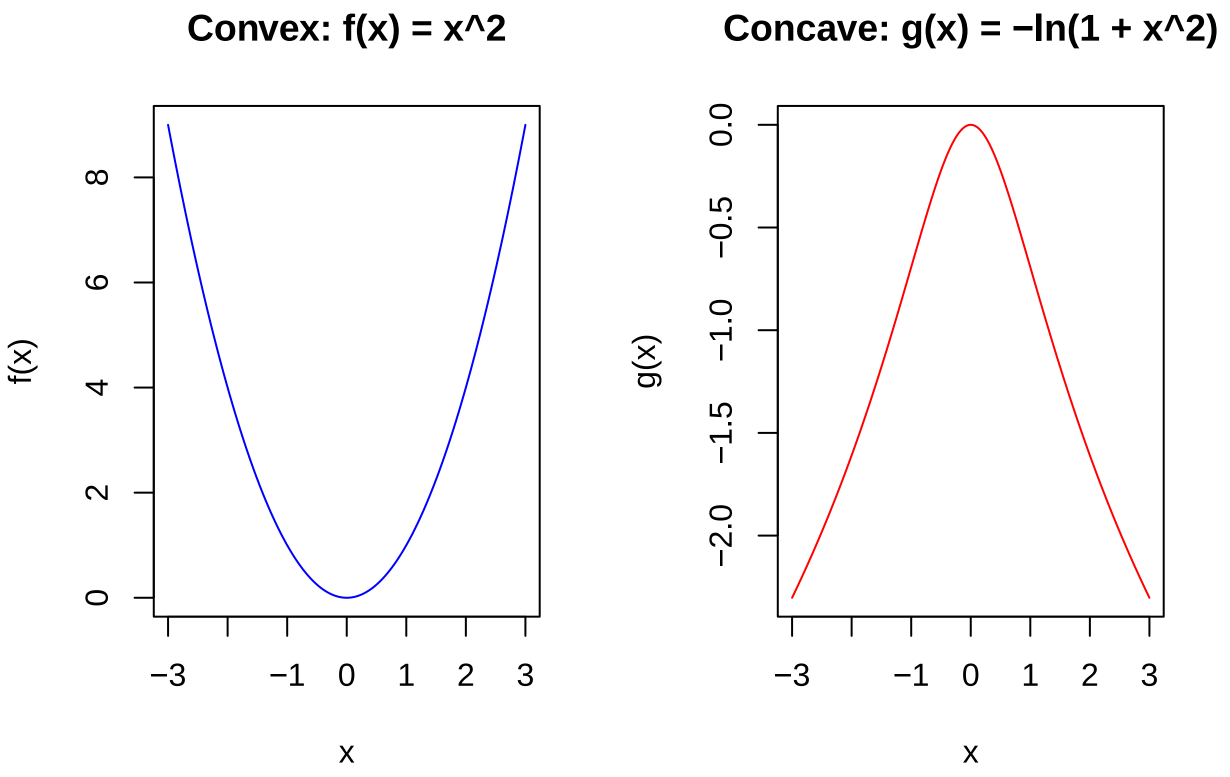
1.2 Integration

You should be familiar with the following properties:

$$\begin{aligned}
\text{Fundamental theorem of calculus: } & \frac{d}{dx}\left(\int_a^x f(t)\,dt\right) = f(x) \\
\text{Let } F(x) = \int_a^x f(t)dt, \text{ then } & \int_a^b f(x)\,dx = F(b)-F(a) \\
\text{Integration by parts: } & \int u\,dv = uv - \int v\,du \\
\text{Integration by substitution: } & \int_a^b g'(x)f(g(x))\,dx = \int_{g(a)}^{g(b)} f(u)\ du
\end{aligned}$$

1.3 Convex and Concave Functions

If f is twice-differentiable, then f is concave iff $f'' \leq 0$. If $f'' \leq 0$ everywhere, then the critical point x^* such that $f'(x^*) = 0$ is a maximizer. Similarly, f is convex iff $f'' \geq 0$. If $f'' \geq 0$ everywhere, then the critical point x^* is a minimizer.



2 Linear Algebra

2.1 Scalars and Vectors

A scalar is a single element of the real numbers. $a \in \mathbb{R}$ is a scalar. We usually denote scalars using lowercase letters, such as a or x. A vector of n dimensions is an ordered collection of n coordinates, where each coordinate is a scalar. By default, vectors will be *columns* and their transposes will be rows. We write vectors in bold lowercase, and the vector itself as a column of scalars:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$$

The magnitude (length) of a vector is

$$||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

The inner product (or dot product) between vectors of the same dimension is written and defined as

$$\mathbf{u} \cdot \mathbf{v} = \langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^T \mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

Dot product is commutative and distributive:

$$\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} \ + \ \mathbf{u} \cdot \mathbf{w}$$

Two vectors are **orthogonal** if $\mathbf{u} \cdot \mathbf{v} = 0$. Geometrically, in ordinary 2D or 3D space, orthogonal vectors point in directions that meet at a right angle (90 degrees). Intuitively, if two vectors are orthogonal, knowing how far you go in the direction of one tells you nothing about how far you go in the direction of another - they are "independent directions."

A set of vectors $\{\mathbf{v_1}, \dots, \mathbf{v_n}\}$ is linearly independent iff the equation

$$c_1\mathbf{v_1} + c_2\mathbf{v_2} + \cdots + c_n\mathbf{v_n} = 0$$

for scalars $c_1, \dots, c_n$ can only be satisfied when $c_1 = \cdots = c_n = 0$. It means that none of the vectors (or linear combinations of them) are parallel.

Question: Does linear independence of a set of vectors imply orthogonality? How about the other way around?

Hint: try creating two vectors that are linear independent but not orthogonal. For the inverse, try multiplying both sides of the above equation by a random vector in the set $\{\mathbf{v_1}, \dots, \mathbf{v_n}\}$.

2.2 Matrices

Conceptually, you can think of matrices as linear operators transforming m-dimensional vectors to n-dimensional vectors. For example, $\mathbf{y} = \mathbf{A}\mathbf{x}$ where $\mathbf{y} \in \mathbb{R}^n, \mathbf{x} \in \mathbb{R}^m, \mathbf{A} \in \mathbb{R}^{n \times m}$. We can think of n-dimensional vectors as matrices with shape $n \times 1$.

Diagonal matrices have non-zero values on the main diagonal (top left to bottom right) and zeros elsewhere. Taking powers of a diagonal matrix gives you another diagonal matrix. The identity matrix $\mathbf{I}$ (written as $\mathbf{I}_n$ for an $n \times n$ shape) has all ones on the diagonal. It has the convenient property:

$$\mathbf{I}_m\mathbf{A} = \mathbf{A}\mathbf{I}_n = \mathbf{A}$$

where $\mathbf{A}$ is an $m \times n$ matrix.

Multiplication of matrices is associative, left and right distributive over addition, but generally not commutative.

2.3 Rank and Inverse

The column rank of a matrix is the dimension of the vector space spanned by its column vectors, i.e., *the number of linearly independent columns*. The row rank is the dimension of the space spanned by its row vectors. A fundamental result in linear algebra is that the column rank and the row rank are always equal and this number is the rank of a matrix. A matrix is full rank if its rank equals the largest possible for a matrix with the same dimensions, i.e. $\text{rank}(\mathbf{A}) = \min(n, m)$, for matrix $\mathbf{A}$ with dimension $n \times m$. For all matrices, $\text{rank}(\mathbf{A}) \leq \min(n, m)$.

The inverse of a $n \times n$ matrix $\mathbf{A}$ is another $n \times n$ matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. $\mathbf{A}$ is invertible iff it is full rank. A non-invertible matrix may also be called *singular*. Important properties: $(\mathbf{A}')^{-1} = (\mathbf{A}^{-1})'$, and $(\mathbf{A}\mathbf{B})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$.

Question 1: Assume matrix $\mathbf{X}$ has shape (or dimension) $n \times d$, and vector $\mathbf{w}$ has shape $d \times 1$. What shape is $\mathbf{y} = \mathbf{X}\mathbf{w}$? What shape is $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$?

Question 2: Prove that if columns of matrix $\mathbf{X}$ are linearly independent, $\mathbf{X}'\mathbf{X}$ is invertible.

Proof: Let $\mathbf{X}$ be $n \times d$ matrix with $n \geq d$. Its d columns are linearly independent, so $\text{rank}(\mathbf{X}) = d$. Let $\mathbf{v}$ be a $d \times 1$ vector such that $\mathbf{X}'\mathbf{X}\mathbf{v} = 0$. We want to show that $\mathbf{v}$ can only be 0. Multiply both sides on the left by $\mathbf{v}'$:

$$\mathbf{v}'(\mathbf{X}'\mathbf{X})\mathbf{v} = (\mathbf{X}\mathbf{v})'(\mathbf{X}\mathbf{v}) = ||\mathbf{X}\mathbf{v}||_2^2 = 0$$

This means that the sum of the square of each element in vector $\mathbf{X}\mathbf{v}$ is 0. Therefore, we must have $\mathbf{X}\mathbf{v} = 0$. Since $\text{rank}(\mathbf{X}) = d$, by the definition of linear independence, we know that $\mathbf{v}$ can only be 0.

3 R Coding

Note: for most R coding issues that do not involve very recent packages, ChatGPT is always a better instructor than me. But please consult Mr. GPT only when you are stuck with a relatively specific problem, for example, when you forgot certain commands in ggplot2 or encountered an error output.

About assignment submission: you always have the option of hand-writing your Pset/exam and scan it as a PDF for submission, as long as the key elements in your answers are recognizable. If you could kindly Latex your answers, for Psets that do not involve coding, Overleaf should be the best platform. For coding tasks, writing in R Markdown and knitting as a PDF should be the default option. You can use this section 0 file as an example.

3.1 Data manipulation

Basic vector creation, calculation, and broadcasting

```r
a <- 1:6
b <- seq(2, 6, by = 2)
c <- rep(1, 6)

p <- a*2 + b*3 + c # b is now effectively c(2, 4, 6, 2, 4, 6)
p
```

```
## [1]  9 17 25 15 23 31
```

Calculation of matrices

```r
I <- diag(1, 3, 3)
X <- matrix(c(1,0,1,3,4,2), nrow = 3, ncol = 2, byrow = T)
Y <- c(1, 2, 3)

solve(t(X) %*% X) %*% t(X) %*% Y
```

```
##           [,1]
## [1,] 0.5575221
## [2,] 0.4513274
```

```r
lm(Y ~ X - 1)
```

```
##
## Call:
## lm(formula = Y ~ X - 1)
##
## Coefficients:
##     X1      X2
## 0.5575  0.4513
```

Manipulation of data frame

```r
df <- mtcars

# Subset by row/column
df[1:5,2:4]
```

```
##                   cyl disp  hp
## Mazda RX4           6  160 110
## Mazda RX4 Wag       6  160 110
## Datsun 710          4  108  93
## Hornet 4 Drive      6  258 110
## Hornet Sportabout   8  360 175
```

```r
# Subset by row/column name
df[c("Mazda RX4", "Datsun 710"), c("mpg", "carb")]
```

```
##             mpg carb
## Mazda RX4  21.0    4
## Datsun 710 22.8    1
```

```r
# Subset by value of variables
df_sub <- df[df$cyl %in% c(6,8), ]
head(df_sub)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
```

```r
# Adding variables
df$whatever <- df$mpg * df$cyl
df_new <- df %>% #requires tidyverse
  group_by(cyl) %>%
  mutate(mean_mpg = mean(mpg)) %>%
  ungroup()

# Joining data frame
employees <- data.frame(
  emp_id = c(101, 102, 103, 104),
  name   = c("Alice", "Bob", "Charlie", "Dora"),
  dept   = c("HR", "Finance", "IT", "IT"),
  stringsAsFactors = FALSE
)

salaries <- data.frame(
  emp_id  = c(101, 102, 104, 105),
  salary  = c(60000, 55000, 70000, 45000),
  stringsAsFactors = FALSE
)

cbind(employees, salaries)
```

```
##   emp_id    name    dept emp_id salary
## 1    101   Alice      HR    101  60000
## 2    102     Bob Finance    102  55000
## 3    103 Charlie      IT    104  70000
## 4    104    Dora      IT    105  45000
```

```
left_join(employees, salaries, by = "emp_id")
```

```
##   emp_id    name    dept salary
## 1    101   Alice      HR  60000
## 2    102     Bob Finance  55000
## 3    103 Charlie      IT     NA
## 4    104    Dora      IT  70000
```

```
full_join(employees, salaries, by = "emp_id")
```

```
##   emp_id    name    dept salary
## 1    101   Alice      HR  60000
## 2    102     Bob Finance  55000
## 3    103 Charlie      IT     NA
## 4    104    Dora      IT  70000
## 5    105    <NA>    <NA>  45000
```

3.2 Sampling

Basic sampling:

```
items <- c("apple", "banana", "cherry", "date", "elderberry")

set.seed(123)
sample(items, size = 3)
```

```
## [1] "cherry"     "banana"     "elderberry"
```

```
sample(items, size = 5, replace = T)
```

```
## [1] "banana"     "banana"     "cherry"     "elderberry" "date"
```

```
# Weighted probabilities
prob_vec <- c(0.1, 0.2, 0.3, 0.1, 0.3)
set.seed(123)
sample(items, size = 3, replace = T, prob = prob_vec)
```

```
## [1] "elderberry" "banana"     "cherry"
```

Sampling from common distributions and sampling rows from data frame:

```r
df <- data.frame(
  x = runif(100, 0, 1),
  y = rnorm(100, 5, 2)
)

set.seed(123)
sample_n(df, 5, replace = F) #requires dplyr(tidyverse)
```

```
##           x        y
## 1 0.7954674 4.361212
## 2 0.6680556 7.234973
## 3 0.1218993 5.075366
## 4 0.2460877 5.991741
## 5 0.4398317 4.478548
```

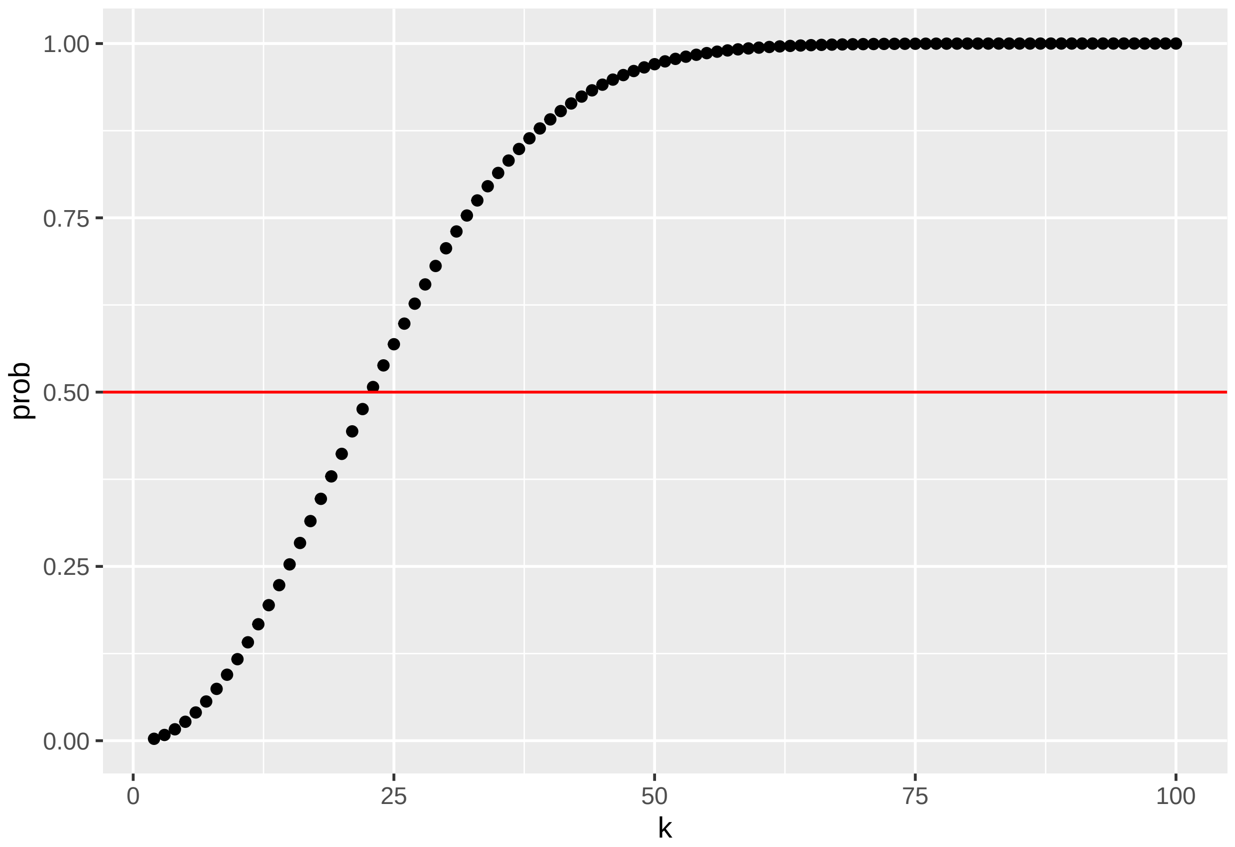
3.3 Writing Functions

Let's write a function to calculate the probability of at least one birthday match as a function of the number of people k and visualize the result.

```r
get_prob <- function(k) {
  denominator <- 365^k
  numerator <- prod((365-k+1):365)
  prob <- 1 - numerator/denominator
  prob_data <- data.frame(k, prob)
  return(prob_data)
}

k_vec <- 2:100
get_prob(23)
```

```
##    k      prob
## 1 23 0.5072972
```

```r
prob_data <- do.call(rbind, lapply(k_vec, get_prob))
prob_data <- map_dfr(k_vec, get_prob) #requires tidyverse

ggplot(data = prob_data, aes(x = k, y = prob)) +
  geom_point() +
  geom_hline(yintercept = 0.5, color = "red")
```



```r
# R actually has a built-in function to do birthday problem!
pbirthday(23, classes = 365, coincident = 2)
```

```
## [1] 0.5072972
```

```r
# Get the number of people required to achieve a given probability
qbirthday(prob = 0.5, classes = 365, coincident = 2)
```

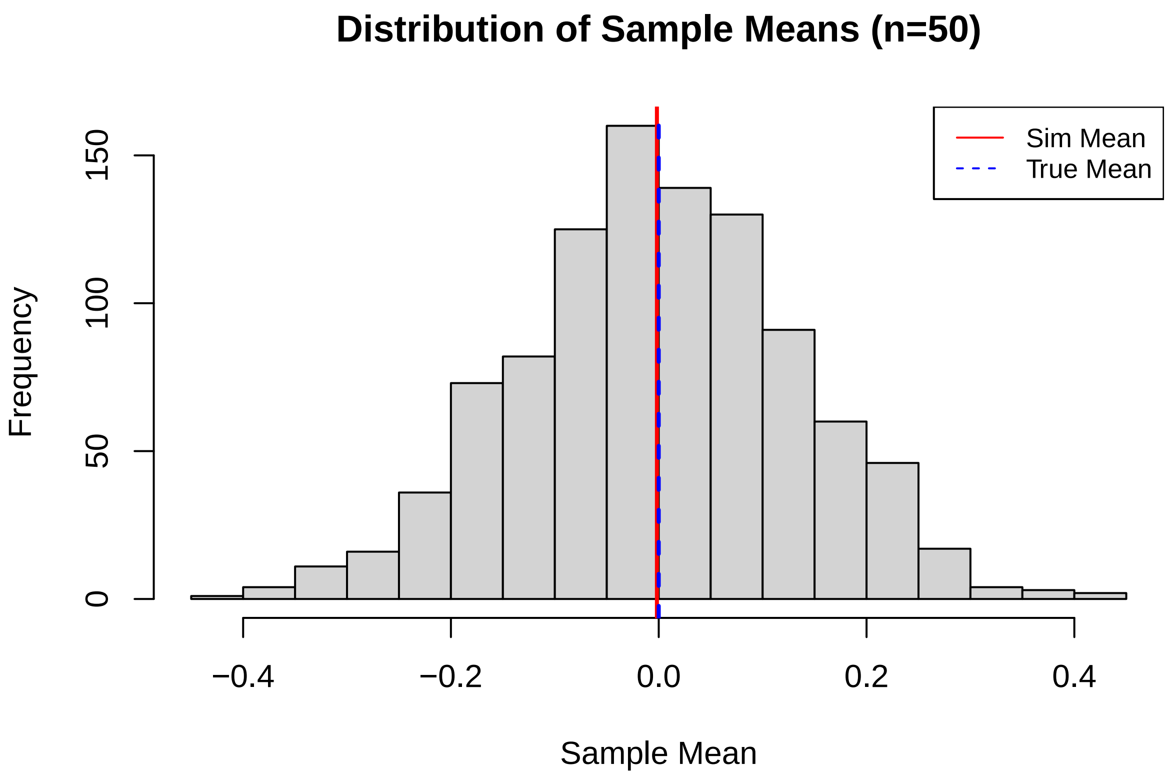
```
## [1] 23
```

3.4 Writing Loops

Let's write a loop that carries out a Monte Carlo simulation. It generates repeated samples from a standard normal distribution and estimates the parameter (mean) using a statistic (the sample mean).

```r
n_sim <- 1000 #Number of simulations

n <- 50 #Sample size

true_mean <- 0 #Parameter of interest
true_sd <- 1

sample_means <- numeric(n_sim) #Storage vector
```

```r
set.seed(123)

for (i in 1:n_sim) {
  # 1) Draw a random sample
  x <- rnorm(n, mean = true_mean, sd = true_sd)

  # 2) Compute the sample mean
  sample_means[i] <- mean(x)
}

hist(sample_means, breaks = 30,
     main = "Distribution of Sample Means (n=50)",
     xlab = "Sample Mean")
abline(v = mean(sample_means), col="red", lwd=2) # average of simulated means
abline(v = true_mean, col="blue", lwd=2, lty=2) # true mean (0)
legend("topright", legend=c("Sim Mean", "True Mean"),
       col=c("red", "blue"), lty=c(1,2), cex=0.8)
```



4 Counting

Selecting a sample of size k from a population of size n:

	Order Matters	Order Does Not Matter
With Replacement	n^k	$\binom{n+k-1}{k}$
Without Replacement	$\frac{n!}{(n-k)!}$	$\binom{n}{k} = \frac{n!}{(n-k)!k!}$

Sampling with replacement (order matters):
You have a jar with n balls, numbered 1 to n. Each time you take a ball out of the jar, you return it to the jar.
Thus, there are n possibilities for each sampled ball, and there are n^k ways to obtain a sample of size k.

Sampling without replacement (order matters):
You have a jar with n balls, numbered 1 to n. Each time you take a ball out of the jar, you do **not** return it to the jar.
Thus, there are $n(n-1)\cdots(n-k+1)$ possibilities for k samples from n balls total, for $1 \leq k \leq n$.

Sampling without replacement (order does not matter):
How many ways are there to choose k times from n objects without replacement, if order doesn't matter? The number of ways to do this is given by the binomial coefficient:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Sampling with replacement (order does not matter): This is equivalent to: How many ways are there to put k indistinguishable balls into n distinguishable jars? This is the "stars and bars" problem.

$$* * * * \,|\, * * | * \,|\, * **$$

We need $n-1$ bars to separate the balls (stars). The total number of bars and balls is $n+k-1$. And the number of ways to choose the positions of the k balls is just $\binom{n+k-1}{k}$.