Quantitative Research Notes

A short complation of knowledge

Pratim Chowdhary

Last updated: February 12, 2025

Contents

※ Introduction

This is a short compilation of notes on various topics in quantitative research. The notes are based on various sources and are intended to be a quick reference guide for students as well as for myself to brush up on concepts.

※ Decision Theory

2.1 Framework

1. A **statistical model** is a framily of distributions, formally defined as:

$$\mathcal{P} = \{P_\theta : \theta \in \Theta\} \tag{1}$$

   where Θ is the parameter space and P_θ is the distribution of the data given θ. The parameter space Θ is the set of all possible values of θ.

2. A **decision procedure** is a function $\delta : \mathcal{X} \to \Theta$ that maps the data space $\mathcal{X}$ to the parameter space Θ. For example if we take the example of a weighted coin flips, we have:

$$\mathcal{X} = \{0,1\}^n \quad \text{and} \quad \Theta = [0,1] \tag{2}$$

   If we are interested in estimating the parameter θ of the coin, we can define a decision space as $\Theta = [0,1]$ and a decision procedure as,

$$\delta(x) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3}$$

   where $x \in \mathcal{X}$ is the data and $\delta(x)$ is the estimate of the parameter θ.

3. A **loss function** is a function $L : \Theta \times \Theta \to \mathbb{R}$ that measures the loss incurred by choosing θ when the true parameter is θ'. For example in many situtations we use the squared loss function:

$$L(\theta, \theta') = (\theta - \theta')^2 \tag{4}$$

4. The **risk** of a decision procedure δ is the expected loss incurred by the decision procedure:

$$R(\theta, \delta) = \mathbb{E}_\theta[L(\theta, \delta(X))] \tag{5}$$

   where $\mathbb{E}_\theta$ denotes the expectation with respect to the distribution P_θ.

2.2 Data Reduction

The idea is that not all data is relevant for making decisions. We can hence reduce the data to a smaller set of and not lose any information.

1. A **statistic** is a function $T : \mathcal{X} \to \mathcal{T}$ that maps the data space $\mathcal{X}$ to a smaller space $\mathcal{T}$.
2. A statistic T is **sufficient** for a parameter θ if the distribution of the data given the statistic T does not depend on θ. Formally if for all t,

$$P_\theta(X|T = t) = P_{\theta'}(X|T = t) \quad \forall \theta, \theta' \tag{6}$$

3. For any matrix $X \in \mathbb{R}^{n \times p}$, $X^T X$ must be at least positive semi-definite, this is because:

$$v^T X^T X v = (Xv)^T Xv = ||Xv||^2 \geq 0 \tag{7}$$

where $v \in \mathbb{R}^p$.

※ Linear Algebra

3.1 Eigenvalues and Eigenvectors

1. The eigenvalues of a matrix $A \in \mathbb{R}^{n \times n}$ are the roots of the characteristic polynomial:

$$\det(A - \lambda I) = 0 \tag{8}$$

where λ is the eigenvalue and I is the identity matrix. The eigenvectors are the vectors v such that:

$$Av = \lambda v \tag{9}$$

2. The trace of a matrix is the sum of its eigenvalues and the determinant is the product of its eigenvalues.
3. The eigenvectors of a matrix are orthogonal if the matrix is symmetric.
4. The eigenvectors of a matrix are linearly independent if the matrix is diagonalizable.

3.2 (Semi) Positive Definite Matrices

1. A matrix $A \in \mathbb{R}^{n \times n}$ is **positive semi-definite** if for all $x \in \mathbb{R}^n$:

$$x^T Ax \geq 0 \tag{10}$$

Positive definite matrices are defined similarly with the inequality replaced by a strict inequality.

2. If a matrix is positive semi-definite, then all its eigenvalues are non-negative and if it is positive definite, then all its eigenvalues are positive.

3.3 Covariance Matrix

1. The covariance matrix of a random vector $X \in \mathbb{R}^n$ is defined as:

$$\Sigma = \mathbb{E}[(X - \mu)(X - \mu)^T] \tag{11}$$

where $\mu = \mathbb{E}[X]$ is the mean of the random vector X. The covariance matrix is a symmetric positive semi-definite matrix. With a real set of data $X \in \mathbb{R}^{n \times p}$, the empircal covariance matrix is defined as:

$$\hat{\Sigma} = \frac{1}{n-1} X^T X \in R^{p \times p} \tag{12}$$

where X is the data matrix with n samples and p features.

2. Note the covariance matrix can only be full rank if $N \geq p$ and none of the features are linearly dependent, this is because:

$$\text{rank}(\Sigma) \leq \min(n, p) \tag{13}$$

and since $n \geq p$, the rank of the covariance matrix is at most p. Some of the useful properties of the covariance matrix are:

- The covariance matrix is symmetric and positive semi-definite.
- The covariance matrix is diagonal $\iff$ the features are uncorrelated.
- It is related to the correlation matrix by:

$$\text{Corr}(X) = D^{-1} \Sigma D^{-1} \quad \text{where} \quad D = \text{diag}(\Sigma)^{1/2} \tag{14}$$

- The covariance matrix is positive definite $\iff$ the features are linearly independent.

3.4 Idempotent Matrices

1. A matrix $A \in \mathbb{R}^{n \times n}$ is idempotent if $A^2 = A$. The following are some properties of idempotent matrices:

- The eigenvalues of an idempotent matrix are either 0 or 1, this can be seen by:

$$Av = \lambda v \implies A^2 v = \lambda^2 v \implies \lambda^2 v = \lambda v \implies \lambda = 0, 1 \tag{15}$$

- The rank of an idempotent matrix is equal to its trace this is because: since the trace is the sum of the eigenvalues, the rank is the number of non-zero eigenvalues.
- The eigenvectors of an idempotent matrix are orthogonal.
- The matrix $I - A$ is also idempotent.

3.5 QR Decomposition

1. The **QR decomposition** of a matrix $A \in \mathbb{R}^{n \times p}$ is a decomposition of the form:

$$A = QR \quad \text{where} \quad Q^T Q = I \quad \text{and} \quad R = \begin{bmatrix} r_{ij} \end{bmatrix} \tag{16}$$

where Q is an orthogonal matrix and R is an upper triangular matrix.

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_p \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} r_{ij} \end{bmatrix} \tag{17}$$

each q_i in Q is an orthogonal vector with $||q_i|| = 1$ and $r_{ij} = 0$ for $i > j$.

2. The **Gram-Schmidt** process is a method for computing the QR decomposition of a matrix. The process is as follows:

   (a) Let $a_1, a_2, \cdots, a_p$ be the columns of the matrix A.

   (b) Set $q_1 = a_1 / ||a_1||$.

   (c) For $i = 2, 3, \cdots, p$, set:

$$q_i = a_i - \sum_{j=1}^{i-1} \text{proj}_{q_j}(a_i) = a_i - \sum_{j=1}^{i-1} \frac{a_i^T q_j}{q_j^T q_j} q_j \rightarrow q_i = \frac{q_i}{||q_i||} \tag{18}$$

   the intuition for this is that we are projecting the vector a_i onto the subspace spanned by $q_1, q_2, \cdots, q_{i-1}$ and then subtracting that projection from a_i.

   Here the R matrix is given by:

$$R_{ij} = q_i^T a_j \tag{19}$$

   as we can get back the original matrix A by:

$$A = [a_1, a_2, \cdots, a_p] = [q_1, q_2, \cdots, q_p] \begin{bmatrix} q_1^T a_1 & q_1^T a_2 & \cdots & q_1^T a_p \\ 0 & q_2^T a_2 & \cdots & q_2^T a_p \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & q_p^T a_p \end{bmatrix} \tag{20}$$

   as $a_i = \sum_{j=1}^{p} q_j R_{ij}$. Intuitively this is true because each a_i is a linear combination of the q_i vectors, specifically the linear component is how much of a_i is in the direction of q_i (the projection).

3. **Linear Least Squares**: QR decomposition can help aid in solving the linear least squares problem especially in cases of high collinearity, as:

$$\text{cond}(X^T X) = \text{cond}(X)^2 \implies (X^T X)^{-1} \quad \text{is ill-conditioned} \tag{21}$$

   Reframing linear least squares with the QR decomposition, we have:

$$X^T X \hat{\beta} = X^T y \implies R^T Q^T Q R \hat{\beta} = R^T Q^T y \implies R\hat{\beta} = Q^T y \tag{22}$$

   Here since R is upper triangular, we can solve for $\hat{\beta}$ by back substitution, creating a more numerically stable solution with easier computation.

3.6 Pseudo Inverse

※ Optimization

※ Statistics

5.1 Hypthothesis Testing

1.

※ Linear Regression

6.1 Ordinary Least Squares

1.

※ Trading (Game Theory)

7.1 Making Markets

1. When **making a market** the most important thing to consider is the fair value of whatever it is you are trying to price. For example that could be the expectation of a sum of random variables,

$$\mathbb{E}[X_1 + X_2 + \cdots + X_n] \quad \text{where} \quad X_i \sim \text{Coin Flip}(p) \tag{23}$$

   in this case we would be trading on the sum of n coin flips with probability of heads p.

2. The **bid-ask spread** is the difference between the price at which you are willing to buy and the price at which you are willing to sell.

   - In general for less liquid assets you would want to have a larger spread, as the risk of holding the asset is higher.
   - For higher variance assets you would also want to have a larger spread as the probability that the asset will move against you is higher. For example if we are trading,

$$X_1 + X_2 + \cdots + X_n \quad \text{where} \quad X_i \sim \text{Normal}(0, 1) \tag{24}$$

     the variance of the sum is n, while in the following case:

$$X_1 + X_2 + \cdots + X_n \quad \text{where} \quad X_i \sim \text{Normal}(0, 0.1) \tag{25}$$

     the variance of the sum is $0.1n$, hence the spread should be larger in the first case, even though the expected value of the sum is the same.

3. **Impact Function**: When someone trades (typically a large order), on your market it makes sense to adjust your theoretical price, for example if you had a market on X,

$$\text{Bid} \quad [100] \quad 95.0 \quad - \quad 105.0 \quad [100] \quad \text{Ask} \tag{26}$$

   and someone buys 100 shares at 105, you would want to think to yourself, "*why did they buy at* 105*?*".

- The buyer could have information that moves the fair value up, and they wanted to profit off of that.
- The buyer could just be willing to pay the spread to get the shares quickly to hold for longer
- The buyer could be uninformed and just buying because they think the price will go up.

In the first case (espcially if you think the buyer is informed), you would want to adjust your theoretical price up so that you don't get crushed by future informed traders.

In the other cases you might not want to adjust your price as much, as the buyer might not have any information that you don't have and you would be losing money by adjusting your price.

※ Practical Statistical Learning

※ Pandas

9.1 DataFrames

1. A DataFrame is a 2-dimensional labeled data structure with columns of potentially different types, some common operations on DataFrames are:

   - **Selecting columns:** Columns can be selected using the column name as an attribute or as a key.
   - **Selecting rows:** Rows can be selected using the `loc` and `iloc` methods.
   - **Filtering rows:** Rows can be filtered using boolean indexing.
   - **Applying functions:** Functions can be applied to columns using the `apply` method.
   - **Grouping data:** Data can be grouped using the `groupby` method.
   - **Merging data:** Data can be merged using the `merge` method.

   Some examples of these operations are:

```python
import pandas as pd
# Create a sample DataFrame
data = {
    'name':   ['Alice', 'Bob', 'Charlie', 'Dave', 'Eve'],
    'age':    [24, 42, 18, 68, 32],
    'city':   ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix'],
    'salary': [50000, 60000, 45000, 70000, 65000]
}
df = pd.DataFrame(data)
# 1. Selecting columns
# Using the column name as a key
names = df['name']
# Using the column name as an attribute (if it doesn't conflict with method names)
ages = df.age
# 2. Selecting rows
```

```
# Using label-based indexing (loc)
first_row_loc = df.loc[0]      # Select row with label 0
subset_loc = df.loc[1:3]       # Select rows with labels 1 through 3
# Using integer-based indexing (iloc)
first_row_iloc = df.iloc[0]    # Select the first row
subset_iloc = df.iloc[1:3]     # Select the 2nd and 3rd rows
# 3. Filtering rows (boolean indexing)
older_than_30 = df[df['age'] > 30]
# 4. Applying functions
# Apply a lambda function to increase salary by 10%
df['salary_with_bonus'] = df['salary'].apply(lambda x: x * 1.1)
# 5. Grouping data
# Calculate the mean salary for each city
mean_salary_by_city = df.groupby('city')['salary'].mean()
# 6. Merging data
# Suppose we have another DataFrame df2 that shares a common key 'name'
df2 = pd.DataFrame({
    'name':   ['Alice', 'Bob', 'Eve'],
    'bonus':  [5000, 7000, 4000]
})
merged_df = pd.merge(df, df2, on='name', how='left')
```

9.2 Summary Statistics

1. Some common summary statistics for a DataFrame `df` are:

   - **Descriptive statistics:** The `describe` method provides summary statistics for numerical columns.
   - **Correlation matrix:** The `corr` method provides the correlation matrix for numerical columns.
   - **Unique values:** The `nunique` method provides the number of unique values for each column.
   - **Value counts:** The `value_counts` method provides the frequency of each unique value in a column.
   - **Missing values:** The `isnull` method provides a DataFrame of missing values.

   Some examples of these operations are:

```
# 1. Descriptive statistics
summary_stats = df.describe()
# 2. Correlation matrix
correlation_matrix = df.corr()
# 3. Unique values
unique_values = df.nunique()
# 4. Value counts
value_counts = df['city'].value_counts()
# 5. Missing values
missing_values = df.isnull()
```

9.3 Data Cleaning

1. A lot of the time data is not clean and therefore needs preprocessing before it can be used for analysis. Some of the basic operations for data cleaning are:

   - **Removing duplicates:** Duplicates can be removed using the `drop_duplicates` method.
   - **Filling missing values:** Missing values can be filled using the `fillna` method.
   - **Replacing values:** Values can be replaced using the `replace` method.
   - **Changing data types:** Data types can be changed using the `astype` method.

   Some examples of these operations are:

```
# 1. Removing duplicates
df_no_duplicates = df.drop_duplicates()
# 2. Filling missing values as 0
df_filled = df.fillna(0)
# 3. Replacing values
df_replaced = df.replace('New York', 'NY')
# 4. Changing data types
df['age'] = df['age'].astype(float)
```

2. **One-Hot Encoding**: for data given in a categorical form, it is often useful to convert it to a numerical form. This can be done using the `get_dummies` method.

```
# Convert the 'city' column to dummy variables
df_with_dummies = pd.get_dummies(df, columns=['city'])
```

   This will create a new column for each unique value in the 'city' column with a 1 if the value is present and a 0 otherwise.

3. **Clipping**: Sometimes it is useful to clip the values of a column to a certain range. This can be done using the `clip` method.

```
# Clip the 'age' column to be between 18 and 65
df_clipped = df['age'].clip(18, 65)
```

9.4 Time Series

1. Time series data is data that is indexed by time. Some common operations on time series data are:

   - **Resampling:** Time series data can be resampled using the `resample` method.
   - **Shifting:** Time series data can be shifted using the `shift` method.
   - **Rolling windows:** Rolling windows can be applied to time series data using the `rolling` method.

   Some examples of these operations are:

```
# 1. Resampling
# Resample the data to monthly frequency
df_resampled = df.resample('M').mean()
# 2. Shifting
# Shift the data by 1 period (move the data down by 1)
df_shifted = df.shift(1)
# 3. Rolling windows
# Calculate the 7-day rolling mean
df_rolling = df.rolling(window=7).mean()
```

※ NumPy

10.1 Arrays

1. NumPy arrays are the core data structure for numerical computations in Python. Some common operations on NumPy arrays are:

   - **Creating arrays:** Arrays can be created using the `array` function or using convenience functions like `zeros`, `ones`, and `arange`.
   - **Indexing:** Elements of an array can be accessed using square brackets.
   - **Slicing:** Subarrays can be accessed using slicing.
   - **Reshaping:** Arrays can be reshaped using the `reshape` method.
   - **Stacking:** Arrays can be stacked vertically or horizontally using the `vstack` and `hstack` functions.
   - **Broadcasting:** Operations can be performed on arrays of different shapes using broadcasting.

   Some examples of these operations are:

```
import numpy as np
# 1. Creating arrays
a = np.array([1, 2, 3])
b = np.zeros((2, 3))
c = np.ones((3, 2))
d = np.arange(10)
# 2. Indexing
first_element = a[0]
# 3. Slicing
first_two_elements = a[:2]
# 4. Reshaping
e = d.reshape((2, 5))
# 5. Stacking
f = np.vstack((b, c))
g = np.hstack((b, c))
# 6. Broadcasting
h = a + 1
```

※ Scikit-Learn

References

[1] Williams, David. *Probability with Martingales*. Cambridge University Press, 1991. Print.