ECE4710J Note

Lecture 2 Data Sampling and Probability

1. Censuses(普查)

Problems: Some data may be uncounted.

2. Surveys

Problems: What is asked and how is asked matters.

3. Sampling

Chance error vs. bias

Chance error in any direction, bias in one direction. Each sample has different chance error direction, has bias in one direction.

Random

1. Convenience samples: **Not random**, select the ease samples. Usually used in pilot testing
2. Quota samples: **Not random**. First group by one feature, and then do convenience sampling
3. Stratified samples: **Random.** First group by one feature, and then do random sampling in each of the subgroups.
4. Simple random samples: **Random.** Use random number generator to generate a sample.
5. Systematic samples: **Random.** Choose one candidates every k elements
6. Cluster samples: **Random.**
7. Voluntary response samples and snowball samples

4. Population and sample

Population is the target set, the sample is a subset of your sample frame. **Sample** set may contain elements that are not in your population.

5. Bias

1. Selection Bias.
2. Response Bias. People don't tell the truth. Usually you don't know you have response bias.
3. Non-response Bias. Certain people don't answer your question.

6. Probability Sampling (Random Sampling)

1. You must be able to tell the probability of selecting each candidates respectively
2. No need to have equal probability for each individual.

7. Simple Random Sampling vs. Random Sample with replacement

Both these two sampling requires uniformly at random (all individuals, pairs, tuples should have same probability), in addition to random sampling. Simple random sampling has no replacement.

Note: If the population is huge enough, then these two samples has no difference

---

Lecture 3 Data Sampling and Probability

1. Multinomial Distribution

Consider the binomial distribution has more than two choices:

$$P[k_1 k_2 k_3] = \frac{n!}{k_1! k_2! k_3!} P_1^{k_1} P_2^{k_2} P_3^{k_3}$$

Use `np.random.multinomial` to compute

2. Distribution

3. Expectation (linear)

4. Variance

Lecture 4 Pandas I

1. Dataframe

Ways to create a Dataframe

The most important data structure in Pandas is Dataframe. Usually, create a dataframe using

```
d = pd.read_csv("a.csv") # read in a pre-exist table
d = pd.dataFrame({dict_t}) # convert a dict to a data frame
d = Series_t.to_frame() # convert a pd.Series type to frame
```

Ways to show rows

```
df.head(5)
df.tail(5)
df.loc[0:4] # row number
```

Ways to show certain rows and columns

```
#loc
df.loc[0:4,"Year":"Party"] # Continuous columns and rows
df.loc[[1,2,5],["Year","Party"]] # Discrete columns and rows
df.loc[:,["Year","Party"]] # All rows but discrete columns. Differentiate this
from loc[0:4] where all columns and some rows
df.loc[:,"Year"] # loc is inclusive, return a pd.Series instead of dataframe
df.loc[1,"Year":"Party"] # Return a pd.Series
```

```
# iloc uses numbers to represent rows and columns, loc can use string
df.iloc[0:4,0:3] # Note that the iloc is exclusive
df.iloc[0,0:3] # Return a pd.Series
```

```
# []
df[["Year","Party"]] # use [] to show discrete columns
df["Party"] # Return a pd.Series
df[["Party"]] # Return a Dataframe with one row instead of a pd.Series
df[1:2] # Exclusive, return a dataframe with certain row. Will not return a
Series
```

Index of Dataframe

If we have label for index, they we expect dataframe to show it. If not, we have no name for index. By default, the dataframe has labels from 1 to n

Not that index do not have to be unique. To define an entry uniquely, we use other ways

Column Name

Similar to index name, column name can be none unique, but we usually expect to have unique column names.

2. Conditional Selection

```
# Provide conditions in the [], this uses the Boolean array input of Python, return index list obeying the condition
df[df["Party"] == "Repubilic"]

df.loc[df["Party"] == "Repubilic"] # Can also use loc/iloc

#Note, the AND and OR are using "&" and "|" in Boolean array input

#Other methods that can be used in Boolean array input
.isin()
.str.startwith()

#Or directly use
df.query("condtion")
...
```

3. Handy Utility Functions

```
# Take in array-like input, usually np.Series
max()
df.size
df.sample(5 , replace = True) # sample the data frame
Series_t.value_counts(sort = True, ascending = False) # Count and sort (or not sort) the Series by the number of occurrences, return a Series
Series_t.unique() # Return a unique Series
Series_t.sort_values() # Sort elements in the Series
...
```

Lecture 5 Pandas II

1. Custom Sorts

```
# Another way of using sort_values, which is done in df
df.sort_values("Count", ascending = False) # Here we use "Count" as input so that we can regard it as a Series

#Customed Key
df.sort_values("Name", key = lambda x: x.str.len(), ascending = False) # Use lambda function to indicate the sort way
```

2. Column Manipulation

```
# Another way of sorting, using additional columns-----------------------
length = df["Name"].str.len()

# adding a new column is like dict
df["length"] = length

# then sort the length directly

# then drop the length column
df = df.drop("length", axis = 'columns') # default axis is row

# sort by arbitrary function, the usage of "map" function is just taking in a
Series, do some manipulation and return a Series---
def f(string):
    return string.count('a')

df["a_count"] = df["Name"].map(f) # treat this as a function
df = df.sort_values(by = "a_count", ascending = False)

# rename a column
df.rename(columns = {"Count":"Count_RTP"}) # Here we use dict to represent the
original name and target name, we can have several
# entries to rename several columns in one time
```

But there are some cases that the `map` should take in all the columns as input. But `map` can only map one to one relation.

3. `Groupby` and `Agg`

`Groupby` creates sub-dataframes and `agg` aggregates all the columns with corresponding data type to one row of output to represent the group.

```
def ratio_to_peak(Series_t):
    return Series_t.iloc[-1]/max(Series_t)

df.groupby("Name").Agg(ratio_to_peak) # Note this will aggregate all the columns
after groupby

# do this instead
df.groupby("Name")[["Count"]].Agg(ratio_to_peak) # Note that here we need to use
[["Count"]], since agg only take in dataframe

# can also use lambda function to agg
df.groupby("Name")[["Count"]].Agg(lambda x: x.iloc[0])

# get raw group by data
gb_df = df.groupby("Name")

# Can use filter to filter subtables that satisfy certain condition
gb_df.filter(lambda x: x["num"].sum() > 10) # return all the columns in subtables
such that the subtable sum is > 10
```

4. `Groupby` and Pivot-Table

```
# Sometimes we want to group by first feature, then by second feature, use a list
to implement
df.groupby(["Year","Sex"]).agg(sum).head()

# Or may use Pivot Table to do better
df_pivot = df.pivot_table(
    index = "Year", # rows labels
    columns = "Sex", # column labels
    values = ["Count", "Name"] # some values to fit in, can also have multiple
values, which leads to multiple of "tupled
columns"
    aggfunc = np.max
).head()
```

5. Join Tables

```
# Match same first Name on left side and Name on right side
merged = pd.merge(left = df1, right = df2, left_on = "First Name", right_on =
"Name", left_index = False, right_index = False)
```

Lecture 6 Regex

```
import re
# metacharacter
|, *, ()

# wildcard
.

# character class
[]

# Repeat for certain times
{a} # Repeat a times
{a,b} # Repeat a or b times

# Repeat at least once
+

# Have zero or one element
?

# Convience Syntax
\w+
\d+
\s
```

How to use?

```
pattern = r"<[^>]+>"
re.sub(pattern, '', text) # text is the input as well as output string

# Pandas allows a continuous expression
repl = ''
Series_t.str.replace(pattern, repl, regex = True) # Replace each entry of the
series using the regex pattern

# Pattern Extraction
Series_t.str.findall(pattern) # return a Series, each entry contains a list of
found patterns


# Can write pattern in groups
pattern = "(\d\d):(\d\d)"

Series_t.str.findall(pattern) # will return a list of tuples, with each tuple
element to be a content in ().

Series_t.str.extractall(pattern) # return a dataframe of all matches, one group("
()") per column
```

Lecture 7 Data Wrangling and EDA

1. Tables vs. matrices

Table can have different element types, matrices can only have one element type.

2. TSV, CSV, JSON

3. Variable Feature Type

1. Quantitative: Can only be interpreted as int/float
    1. Discrete
    2. Continuous (ration of classes have meaning)
2. Qualitative: May be interpreted as strings, even we have numbers.
    1. Ordinal (ratio of classes no meaning) (educational level, say from 1 to 5)
    2. Normal: No meaning ID (Person ID)

4. Primary Key and Foreign Key

Columns determining the value of rest columns, i.e., make the rest unique

5. Scope

6. Temporality

Convert time zone

```
Series_t.tz.convert("target time zone name")
```

7. Deal with missing data

1. Drop records with missing values
2. Use average value to fill in
3. Use random value to fill in

Lecture 8 Data Visualization I

Not important point for code

1. Bar Plot

```
# Matplotlib
plt.bar(x,y)

#Seaborn
sns.countplot(data = y, x = x)

#Plotly
px.histogram(y, x = x, color)
```

2. Histogram and Rugplots

3. Mode, Tail and Skewness

4. Density function

5. Box Plot and Violin Plots

Lecture 9 Data Visualization II

1. KDE

1. Place a Gaussian Kernel at each point, which bandwidth $\alpha = 1$.
2. Normalize so that all kernels add to 1
3. Add all kernels

Choose of α

1. Too small makes it not smooth
2. Too big loses features, may turn multimodal to unimodal

2. Encoding

You have different marks and different encoding ways.

4. Harnessing X/Y

Consider scale of the data. When two plots are combined, use the same scale. Or consider using percentage, or you simply look at a part of the data

5. Harnessing Color

Use Turbo

Sequential data: Light for extreme values

diverging data: Light for middle values

6. Harnessing Markings

Length is good, area is bad

Do not pile up baseline

7. Harnessing Conditioning

When comparing, consider line plot instead of bar plot. Consider ratio to show two data that are near and not easy to compare.

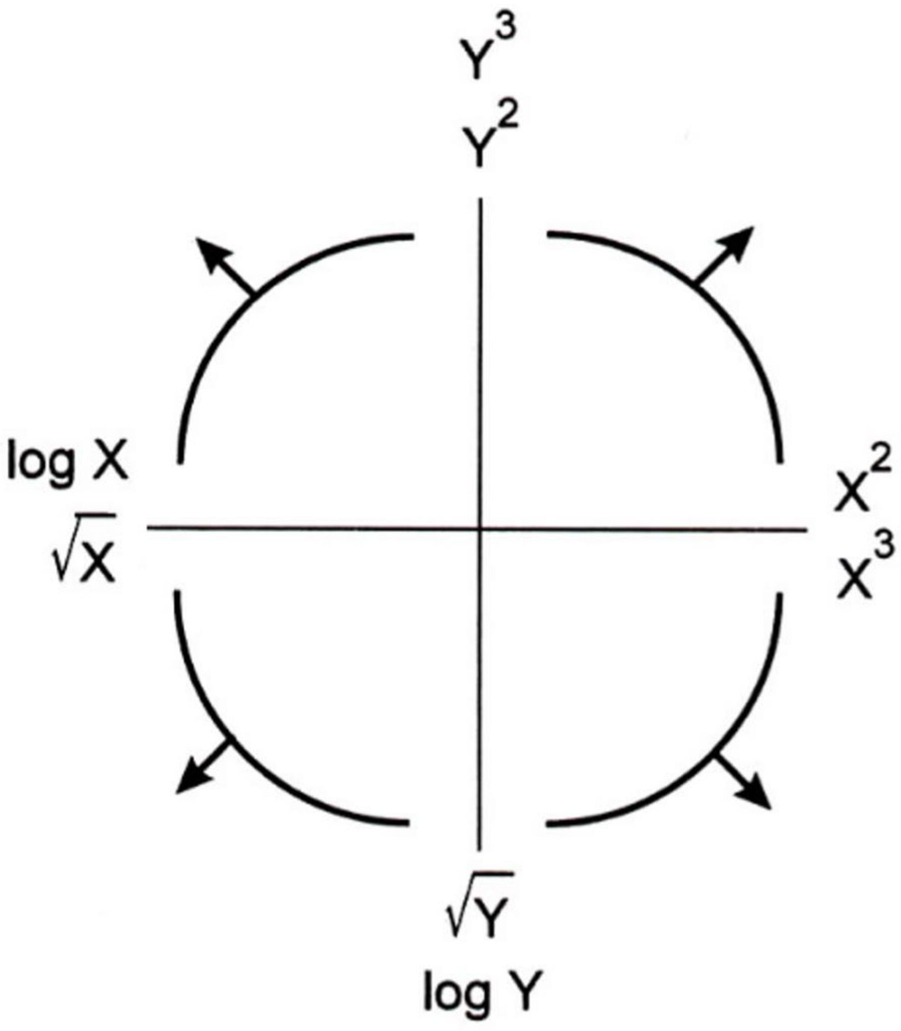
Consider superposition to put lines together. Consider Juxtapostion when there are lots of similar plots.

8. Harnessing Context

Tell people directly using captions

9. Transformation

Tukey-Mosteller Bulge Diagram



Lecture 10 Intro to Modeling

1. Linear Regression

$$\text{Regression Formula} := \hat{y} = \hat{a} + \hat{b}x$$
$$\hat{a} = \bar{y} - \hat{b}\bar{x}$$
$$\hat{b} = r\frac{\sigma_y}{\sigma_x}$$
$$r = \frac{1}{n}\sum_{i=1}^{n}(\frac{x_i - \bar{x}}{\sigma_x})(\frac{y_i - \bar{y}}{\sigma_y})$$
$$e_i = y_i - \hat{y}_i$$

2. Loss Function

L2 loss & L1 Loss

Usually consider **Average Loss**, with a certain loss function

$$R(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$$

This gives MSE and MAE

3. How to interpret

Linear regression does not represent cause-result relation. It represents association.

4. Visualization

Ensure no pattern in residual plot. If you have patterns, consider tranformation.

Lecture 11 Constant Model

1. Estimate & Predict

Estimate is to solve for the optimal estimator. Then we can predict.

2. Constant Model

$$\hat{y} = \theta = \bar{y}$$

and the **Average Loss**

$$R(\theta) = \sigma_y^2 + (\bar{y} - \theta)^2$$

If we use MAE, then

$$\hat{y} = \text{median}(y)$$

3. MSE vs. MAE

$R(\theta)$ for MSE is sensitive to outliers, but smooth. $R(\theta)$ for MAE is robust to outliers, but not smooth. This difference is the difference of mean and median.

Lecture 12 OLS

1. Multiple linear Regression

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$
$$\text{or}$$
$$\hat{y} = x^T\theta$$

$\mathbb{X}$ is the design matrix, with size $n \times (p+1)$, that is, p features and n observations

2. Minimize Average Loss Using Geometry

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

3. Coefficient of determination

Has no correlation coefficient, but coefficient of determination

$$R^2 = \frac{\sigma_{\hat{y}}^2}{\sigma_y^2}$$

4. Properties

If the OLS has intercept term, we have

$$\sum_{i=1}^{n} e_i = 0$$

$$\bar{y} = \bar{\hat{y}}$$

About the uniqueness of $\hat{\theta}$, it is unique iff $\mathbb{X}$ is full rank.

1. Number of features cannot be bigger than number of observations
2. All features should be independent

Lecture 13 Feature Engineering

1. Feature Function

Transform input data to higher dimensional features. e.g. consider the transformation from x to x and x^2.

2. One Hot Encoding

Use 0 or 1 to indicate that they observation has or does not have this feature

Another way is to treat the features separately.

3. Word Encoding

Use a long vector, each entry represents one word or several sequential words.

4. Overfit

To detect overfitting, we need collect new data and calculate MSE, we will see that after some complexity, we will have MSE increasing instead of decreasing, then we are overfitted.

Lecture 14 Cross Validation

1. Holdout Method

Shuffle the data, split to training set and validation set

2. K-fold Cross Validation

Divide the data into k part, each time use $k-1$ part to train and the remaining part to validate. The MSE will be the average of all MSEs.

Choose K

Can be N, which gives good result, but expensive.

3. Test Set

Usually divide the data to training set, validation set and test set. Note that the validation set is used to select hyperparameter. And test set is used to test the performance of your model with hyperparameter selected.

4. L2 Regularization

Consider a ball and the choice of parameters can only be in that ball. A small radius means less complex model.

5. L1 Regularization

Consider a square, others are same to L2.