Data 8 Final Study Guide — Page 1

Higher-order function: A function that returns a function

```
def <name>(x):
    def <inner name>(y):
        return <some expression involving x and y>
    return <inner name>
```

The returned *function* takes one argument (`y`) but computes its result from two values (`x` and `y`).

The regression line is the one that minimizes the (root) mean squared error of a collection of paired values

- The slope and intercept are unique for linear regression

```
def mean_squared_error(table, x, y):
    def for_line(a, b):
        estimate = (a * table.column(x) + b)
        return np.average((table.column(y) - estimate) ** 2)
    return for_line
long = faithful.where('eruptions', are.above(3))
mse_long = mean_squared_error(long, 1, 0)
a, b = minimize(mse_long) # regression line is a * x + b
```

Simple regression: one input ➔ one output
Multiple regression: many inputs ➔ one output

$$\text{GPA} = a_{\text{days}} * \text{days} + a_{\text{contributions}} * \text{contributions} + b$$

We could find `a`'s and `b` by minimizing mean squared error

Inference: Making conclusions from random samples
Population: The entire set that is the subject of interest
Parameter: A quantity computed for the entire population
Sample: A subset of the population
In a **Random Sample**, we know the chance that any subset of the population will enter the sample, in advance
Statistic: A quantity computed for a particular sample

Estimation is a process with a random outcome
Population (fixed) → Sample (random) → Statistic (random)
A 95% **Confidence Interval** is an interval constructed so that it will contain the parameter for 95% of samples
For a particular sample, the interval either contains the parameter or it doesn't; the process works 95% of the time
Resampling: When we wish we could sample again from the population, instead sample from the sample

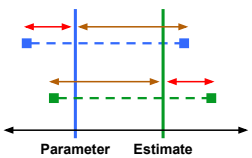
---

The 80th percentile is the value in a set that is at least as large as 80% of the elements in the set
For `s = [1, 7, 3, 9, 5]`, `percentile(80, s)` is 7
The 80th percentile is ordered element 4: `(80/100) * 5`
For a percentile that does not exactly correspond to an element, take the next greater element instead

Percentile | Size of set

`percentile(10, s)` is 1 `percentile(20, s)` is 1
`percentile(21, s)` is 3 `percentile(40, s)` is 3

If an interval **around the parameter** contains the estimate, then a (reflected) interval of the same width **around the estimate** contains the parameter (and vis versa)



K-nearest Neighbor classifier

Goal: Using a set of labeled examples, define a classifier that can predict the label of an unlabeled example
Example: A set of features (i.e., a row in a table of features)
Data preparation: *Randomly* divide all available examples into three sets: training, validation, and test
Classification: Find the K examples in the training set that are most similar to the example and average the labels
Accuracy: The proportion of examples in a set (e.g., the test set or validation set) that are classified with the correct label

```
def distance(pt1, pt2):
    tot = 0
    for i in range(len(pt1)):
        tot = tot + (pt1[i] - pt2[i])**2
    return math.sqrt(tot)

def computetablewithdists(training, p):
    dists = np.zeros(training.num_rows)
    attributes = training.drop('Class')
    for i in np.arange(training.num_rows):
        dists[i] = distance(attributes.row(i), p)
    return training.with_column('Distance', dists)

def closest(training, p, k):
    withdists = computetablewithdists(training, p)
    sortedbydist = withdists.sort('Distance')
    topk = sortedbydist.take(np.arange(k))
    return topk

def majority(topk):
    if topk.where('Class',1).num_rows>topk.where('Class',0).num_rows:
        return 1
    else:
        return 0

def classify(training, p, k):
    closestk = closest(training, p, k)
    topkclasses = closestk.select('Class')
    return majority(topkclasses)

def evaluate_accuracy(training, test, k):
    testattrs = test.drop('Class')
    numcorrect = 0
    for i in range(test.num_rows):
        # Run the classifier on the ith patient in the test set
        c = classify(training, testattrs.rows[i], k)
        # Was the classifier's prediction correct?
        if c == test.column('Class').item(i):
            numcorrect = numcorrect + 1
    return numcorrect / test.num_rows
```

Feature Selection Using the Validation Set

Goal: Select features and K to maximize accuracy
Overfitting: Accuracy is higher on the set used for feature selection than another set because of peculiarities of the set
Selecting Features: Maintain a set of selected features, which starts out empty; Each added feature is the feature that improves validation set accuracy the most, in combination with the features already added

---

Inferential idea: The variability of a statistic for different samples is similar to the variability of that statistic for resamples
Computing a confidence interval for an estimate from a sample:

- Collect a random sample
- Compute your estimate (e.g., sample average)
- Resample K samples from the sample, with replacement.
  - Compute the same statistic for each resampled sample
  - Take *percentiles* of the deviations around the estimate
  - 95%: (estimate - 97½% deviation, estimate - 2½% deviation)

```
def confidence_interval_95_percent(sample, label, f):
    deviations = Table(['Resample #', 'Deviation'])
    n = sample.num_rows
    stat = f(sample.column(label))
    for i in np.arange(400):
        resample = sample.sample(n, with_replacement=True)
        dev = f(resample.column(label)) - stat
        deviations.append([i, dev])
    return (stat - percentile(97.5, deviations.column(1)),
            stat - percentile( 2.5, deviations.column(1)))
```

Data 8 Final Study Guide — Page 2

Total Variation Distance (TVD):

- For each category, compute the difference in proportions between two distributions
- Take the absolute value of each difference
- Sum and divide by 2

```
def total_variation_distance(column, other):
    return sum(np.abs(column - other)) / 2
```

`np.random` is a collection of sampling procedures

```
np.random.randint(lower, upper)
```

An integer lower bound (inclusive) | An integer upper bound (exclusive)

```
np.random.multinomial(n, distribution)
```

The number of samples to draw | An array or list containing the chance for each category

---

Does a sample look like a random sample from a distribution?

- E.g., Do the counts of races on a jury panel look like a random sample from the proportion of races in a county?
- Sample many times from the distribution and compute the TVD of each sample from the original distribution
  - The result is called an *empirical distribution* of a statistic
  - It approximates the *sampling distribution* of the statistic
- Compare the observed TVD between the sample and the original distribution to this *empirical distribution*

```
def empirical_distribution(table, label, size, k, f):
    stats = Table(['Sample #', 'Statistic'])
    for i in np.arange(k):
        sample = table.sample_from_distribution(label, size)
        statistic = f(sample)
        stats.append([i, statistic])
    return stats
```

Step 1: The Hypotheses

- A test chooses between two views of how data were generated
- *Null hypothesis* proposes that data were generated at random
- *Alternative hypothesis* proposes some effect other than chance

Step 2: The Test Statistic

- A value that can be computed for the data and for samples

Step 3: The Sampling Distribution of the Test Statistic

- What the test statistic might be if the null hypothesis were true
- Approximate the sampling distribution by an empirical distribution

Step 4: Resolve choice between null and alternative hypotheses

- Compare observed test statistic to its empirical distribution under the null hypothesis
- If the observed value is **consistent** with the distribution, then the test *does not* support the alternative hypothesis

P-Value: The chance, under the null hypothesis, that the test statistic is equal to the value that was observed or is even further in the direction of the alternative.

Statistically Significant: The P-value is less than 5%

Highly Statistically Significant: The P-value is less than 1%

Bootstrap A/B test using means

- *n* individuals in Category A; *m* in Category B
- Under the null hypothesis, samples in both categories are drawn from the same underlying distribution.
- So draw (*n*+*m*) times at random with replacement from the entire sample. Assign *n* of the draws to Category A and *m* to Category B.
- Find the difference between the means of these two new groups.
- Repeat the last two steps many times and compare with the difference between the two original sample means.

We observed a positive slope and used it to make our predictions.

But what if the scatter plot got its positive slope just by chance?

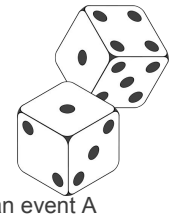
What if the true line is actually FLAT?

- **Bootstrap the scatter plot & find the slope of the regression line through the bootstrapped plot many times.**
- Draw the empirical histogram of all the resampled slopes.
- Get the "middle 95%" interval: that's an approximate 95% confidence interval for the slope of the true line.
- **Null hypothesis:** The slope of the true line is 0.
  - Construct a bootstrap confidence interval for the true slope.
  - If the interval doesn't contain 0, reject the null hypothesis.
  - If the interval does contain 0, there isn't enough evidence to reject the null hypothesis.

- **Residual**: Difference between y and regression line height at x.
- **Regression Model**: y is a linear function of x + normal "noise"
- Residual plot looks like a formless "noise" cloud under this model
- Average of residuals is always 0 for any scatter diagram

$$|r| = \frac{\text{SD of fitted values of } y}{\text{SD of observed values of } y} \qquad \text{SD of residuals} = \sqrt{1-r^2} \times \text{SD of } y$$

- **Experiment**: An occurrence with an uncertain outcome
- **Outcome**: The result of an experiment
- **Sample Space**: The set of all possible outcomes for the experiment
- **Event**: A subset of possible outcomes that have some property of interest
- **Probability**: The proportion of experiments for which the event occurs
- **Distribution**: The probability of all events



Multinomial distributions have a finite sample space

- Distribution with a fixed number of possible outcomes
- All outcomes are mutually exclusive for an experiment
- The sum of the chances of the outcomes is 1
- The probability of an event is the sum of the chances of the outcomes in which the event occurs

Conditional Probability: The chance of an event B **given** an event A

Both events describe the same experiment & sample space

- Find all outcomes in which A occurs
- Divide the chances of A's outcomes by P(A)

Conditional Distribution

- Find all outcomes in which B also occurs
- Sum the updated chances of these outcomes: P(B | A) = P(A, B) / P(A)
- Bayes' Rule: P(A | B) = P(B | A) * P(A) / P(B)

Comparing samples: Do two groups differ in some attribute?

- **E.g.**: Among U.S. men who are in heterosexual relationships, is there an association between marriage and employment status?
- **Null hypothesis**: In the United States, the distribution of employment status among married men is the same as among unmarried men who live with their partners.
- **Inferential Idea:** If marital status and employment status were not connected in any way, then we could replicate the sampling process by replacing each man's employment status by a randomly picked employment status from among all the men
  - Permute (shuffle) the outcome column K times
  - A shuffled table pairs each example with a random outcome
  - Compute the total variation distance between the two treatment conditions to create an empirical distribution

```
repetitions = 500
tvds = Table().with_column("TVD between married and partnered men", [])
for i in np.arange(repetitions):
    # Construct a permuted table
    shuffled = males.select('Employment Status').sample()
    combined = Table().with_columns([
            "Marital Status", males.column('Marital Status'),
            "Employment Status", shuffled.column('Employment Status')])
    employment_shuffled = combined.pivot('Marital Status','Employment Status')
    # Compute TVD
    e_s = employment_shuffled
    married = e_s.column('married')/sum(e_s.column('married'))
    partner = e_s.column('partner')/sum(e_s.column('partner'))
    permutation_tvd = 0.5*sum(abs(married - partner))
    tvds.append([permutation_tvd])
```