Prob Set 7: due Wed 3 May

Prob Set 8: due 11 May (end of classes)
Reaching Out to the Voters the Left Left Behind

Thomas B. Edsall  APRIL 13, 2017

White ‘Deaths of Despair’
Deaths per 100,000 non-Hispanic whites aged 50 to 84, from suicide, alcohol or other drugs, by education level.

Mortality rates for whites were stable or declining in all counties with populations of one million or more — the counties, in other words, that voted decisively for Clinton. Conversely, white mortality rates rose by roughly one percent a year in all counties of less than a million — the counties that voted for Trump.

Bill Bishop, co-author of the book “The Big Sort” and a founder of The Daily Yonder, makes the case that the political split in America is not an urban-rural divide. Instead, he argues, it is between the largest cities and the rest of America.

Source: Brookings Papers on Economic Activity
By The New York Times


ps7#4:

https://wonder.cdc.gov/ucd-icd10.html
Defined as the Pearson correlation for the ranks, the Spearman correlation is written

\[ \rho = \frac{\text{Cov}[r, s]}{\sigma[r] \sigma[s]}, \]

where \( \text{Cov}[r, s] = E[(r - E[r])(s - E[s])] \)

(generalizing the \( \text{Var}[x] = E[(x - E[x])^2] \), with \( \text{Cov}[x, x] = \text{Var}[x] \)).

The formula for the Spearman correlation coefficient is given at
http://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient
in terms of the difference \( d_i = r_i - s_i \) between ranks, in this easily calculable form:

\[ \rho = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)} . \]

It is straightforward to verify that (1) reduces to (2) (see linked notes).
\text{pearsonr}(xdata,ydata) = .91

\text{spearmanr}(xdata,ydata) = 1
\text{pearsonr}([1,2,3],[1,2,3]) = 1

\text{pearsonr}(xdata,ydata) = .091
\text{spearmanr}(xdata,ydata) = .5
\text{pearsonr}([1,3,2],[1,2,3]) = .5

\text{var}([1,2,3])=2/3
\text{cov}([1,2,3],[1,3,2]) = \frac{((1-2)(1-2) + (2-2)(3-2) + (3-2)(2-2))}{3} = 1/3
Linear Regression (Least Square Fit)

The best line $y=ax+b$ is determined by parameters $a,b$ that minimize the sum

$$\sum_{i=1}^{n} ((ax_i + b) - y_i)^2$$

over the $n$ data points $(x_i, y_i)$

The solution was

$$b = \mathbb{E}[y] - a\mathbb{E}[x]$$

$$a = \frac{\text{Cov}(x,y)}{\text{Var}(x)}$$
Fit power law data

\[ p(x) = Cx^{-k} \]

\( C \) is determined by the normalization that the sum over the probabilities (in this case the integral from some \( x_{\text{min}} \) to \( \infty \)) is equal to 1.

The overall probability of the data for some given \( k \) is

\[ p(x \mid k) = p(x_1 \mid k) p(x_2 \mid k) p(x_3 \mid k) \ldots p(x_n \mid k) \]

so the best fit is given by the value of \( k \) that maximizes the probability of the data. Setting to zero the derivative of the above with respect to \( k \) gave

\[ k = 1 + n / \sum_{i=1}^{n} \ln(x_i / x_{\text{min}}) \]
In each case, there’s a model \( M \), with some parameters, intended to describe some data \( x \).

How to determine the model parameters?

A. **Maximum Likelihood Estimate** (MLE)
   
   find the parameters that maximize \( p(x \mid M) \)

B. **Maximum a Posteriori Estimate** (MAP)
   
   find the parameters that maximize \( p(M \mid x) \)
modification of regression scheme for predicting a binary outcome rather than a linear relation

logistic function: \[ \text{logit}(x) = \frac{1}{1 + e^{-x}} \]

For large positive values of the argument \( x \), \( e^{-x} \) becomes very small, so the function evaluates to 1,

whereas for large negative values of \( x \), \( e^{-x} \) becomes very large and the function evaluates to zero.

The value for \( x=0 \) is \( \text{logit}(0)=.5 \), and that marks the middle of the transition region between 0 and 1.
$x$ can be considered to be any linear combination of feature values $f_i$ being used to make the binary prediction, $x = \alpha_1 f_1 + \ldots + \alpha_n f_n + \beta$, where the weights $\alpha_i$ and $\beta$ are determined by some training procedure (to provide the best fit to labelled training data). Intermediate values of the logit function can be interpreted as the probability of the label being 1 for those values of the features.

For a single feature $x = \alpha f_1 + \beta$, the logit is equal to .5 for $f_1 = -\beta/\alpha$, and the width of the transition region is proportional to $1/\alpha$:

$$\logit(x) = \frac{1}{1 + e^{-x}}$$
Logistic Regression

again determine parameters by MLE
Markov chains

Consider a system with M states, labelled 1,2,3,…M which undergoes a series of transitions 
2,1,8,5,6, …

\[ X_i = \text{state after } i \text{ transitions} \]

\[ X_0 \ X_1 \ X_2 \ X_3 \ X_4 \ X_5 \ X_6 \ldots \]

Markov chain property:

\[ T_{ij} = p(X_{n+1} = j | X_n = i) = p(X_{n+1} = j | X_n = i, X_{n-1} = i - 1, \ldots, X_0) \]
States with probabilistic transitions

\[ T = \begin{pmatrix} .5 & .5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & .2 & .8 \\ 0 & 1 & 0 & 0 \end{pmatrix} \]
A state is **recurrent** if wherever you go from it, there’s always a way back.

If a state is not recurrent, then it’s called **transient**.

A state is **periodic** if recurrent and there exists a number $d$ such that probability of returning is zero except in steps of multiples of $d$.

A **recurrent class** is a set of recurrent states which can all reach one another: they are said to communicate.

A Markov chain is **ergodic** if it has only one recurrent class and no periodic states. (These are the “good” Markov chains, where the states all communicate, and are inter reachable via any number of steps.)
def make_trigrams(filename):

    with open(filename) as f: words = f.read().split()

    trigrams = defaultdict(list)

    bigram = tuple(words[:2])

    for w in words[2:] + words[:2]:
        # keys of trigram dict are tuples, values are lists
        trigrams[bigram].append(w)
        bigram = (bigram[1], w)

    return trigrams
def random_text(trigrams, startwords, num_words=100):
    current_pair = random.choice(startwords)
    random_text = list(current_pair)

    # continue past num_words until ends in .
    while len(random_text) < num_words:
        next = random.choice(trigrams[current_pair])
        random_text.append(next)
        current_pair = (current_pair[1], next)
        # avoid long loops if too few periods in training text

    return ' '.join(random_text)

(We, both) sprang
(both, sprang) in
(sprang, in) and
in and away
and away we
away we went
we went to
to the ventilator
to the ventilator and
ventilator and to
and to carry
to carry out
carry out my