Natural Language Processing Course

March 15, 2020

1 Piotr Styczyński NLP Ex. 1

NLP @ 24 Feb 2020
The following code reads tab file from NKJP and calculates frequencies counts, plots them and tries to estimate Zipf coefficient for the given text data.

I used the builtin numpy optimization algorithms for minimization to find optimal zipf coefficient. From zipf law:

$-log(x) * a + b = log(y)$
$e^{log(x)*(-a)+b} = y$
$x^{-a} + e^{b} = y$
$x^{-a} + c = y$

Hence zipf distribution for $x = 1, 2, ..., n$ states that the probability of occurrence of x i proportional to x^{-a}. Let's suppose we've got arbitrary fixed factor a and frequencies of occurrence for each x_i that comes from the experiment result $f_1, f_2, f_3, ..., f_n$ Then logarithm from probability of getting x is equal to:

$log(P(x)) = log(\frac{x^{-a}}{k})$

Note that k is a constant equal to $\sum_{i=0}^{n} x^{-a}$ so that $\sum_{i=0}^{n} P(x) = 1$, so that is a valid probability distribution (we perform normalization).

Probability of getting f_1 times number 1, f_2 times number 2, ... f_n times number n is equal to:

$Q(f_1, f_2, \ldots f_n) = P(1)_1^f + P(2)_2^f + \ldots + P(n)_n^f$

Logarithm from Q is equal to:

$log(Q) = log(\sum_{i=0}^{n} P(i) * f_i)$
$log(Q) = \sum_{i=0}^{n} f_i * log(P(i))$

Hence: $log(Q) = f_i * log(x^{-a} / (\sum_{j=0}^{n} x^{-a})))$

So we've got function $F(a) = \sum_{i=0}^{n} f_i * log(\frac{x^{-a}}{\sum_{j=0}^{n} x^{-a}}))$ And we want to minimize it over F

Property	Value
Case	Zipf curve for forms
Best coeffs from log likelihood	11.64 + x*-2.01
Curve slope	63.5

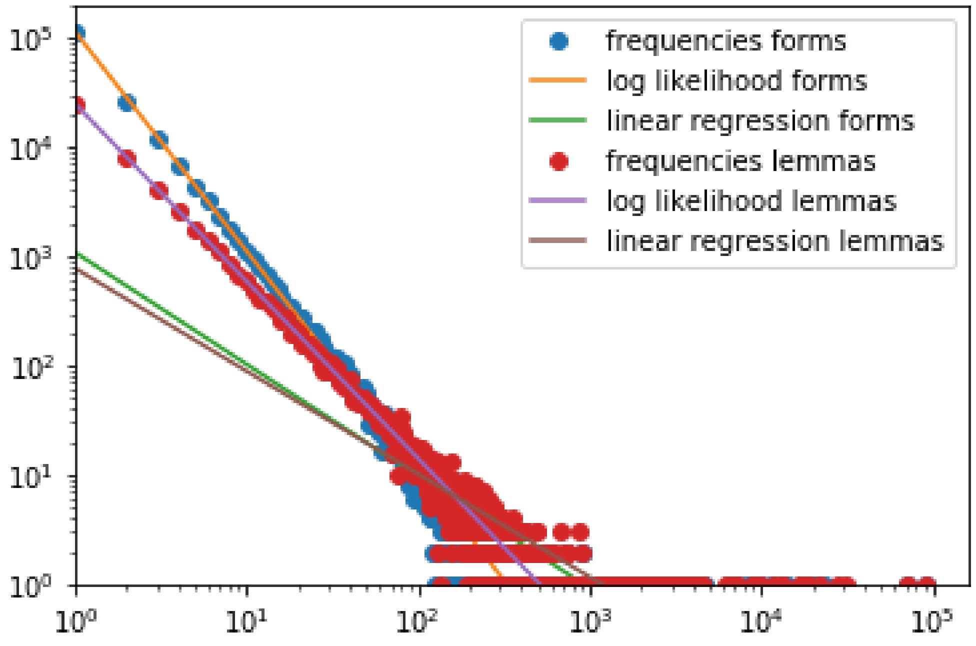
Property	Value
Case	Zipf curve for lemmas
Best coeffs from log likelihood	10.12 + x*-1.62
Curve slope	58.33

(i.e calculate $argmax_a(F(a))$) That way we've got most probable zipf coefficient.

We use scipy builtin engine for finding optimums in linear sets of constrained equations. The logarithm is monotonic so it does not affect local optimiums locations and eliminates exponentiation We can simplify the formula to find a minimum of:

$F(a) = \sum_{i=0}^{n} f_i * -a * log(x) - log(H_a(n)))$
Where $H_k(n) = \frac{1}{1^k} + \frac{1}{2^k} + \frac{1}{3^k} + \ldots + \frac{1}{n^k}$ (harmonic sum of rank k)

But this not simplify computation a lot. This approach is referred to as minimization of log likelihood



```
In [ ]: import csv
        import sys
        import collections
        import math
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy.optimize import minimize
        from scipy.stats import linregress
        import scipy

        from format_utils import ListTable
        from IPython.core.display import display

        # Some fields overflow default CSV buffer so we set incredibly huge default
        csv.field_size_limit(sys.maxsize)

        for [case_no, case_name] in [[0, 'forms'], [1, 'lemmas']]:
            # Open input file
            freqs = dict()
            with open('1_NKJP1M-frequency.tab', newline = '') as file:
                reader = csv.reader(file, delimiter='\t', quoting=csv.QUOTE_NONE)
                counts = []
                if case_no == 0:
                    counts = np.array([int(r[-1]) for r in reader])
                else:
                    lem_freq = dict()
                    for r in reader:
                        if r[1] in lem_freq:
                            lem_freq[r[1]] += int(r[-1])
                        else:
                            lem_freq[r[1]] = int(r[-1])
                    counts = list(lem_freq.values())
                freqs = collections.Counter(counts)

            # Get x and y axis samples for plotting
            # this changes dict(key: count) into two lists [keys..] [counts..]
            # and creates numpy arrays from them
            [x, y] = map(np.array, map(list, zip(*sorted(freqs.items()))))

            # Function used for minimization
            def opt(alpha):
                p_i = x**(-alpha)
                return -(np.log(p_i / p_i.sum()) * y).sum()

            # We find optimal linear function by minimizing distance (see opt function)
            # This uses BFGS underneath and it's fine for this usage
            coeff = (-minimize(opt, [2]).x)[0]

            # Plot results
            plt.plot(x, y, 'o', label=f"frequencies {case_name}")
            plt.xscale("log")
            plt.yscale("log")

            # Trim axes of graph
            cc = y[0] * (x ** [coeff])
            plt.ylim(1)
            plt.xlim(1)
            plt.plot(x, cc, label=f"log likelihood {case_name}")

            # Plot also linear regression for comparison
            lr_a, lr_b, _, _, _ = scipy.stats.linregress(np.log(x), np.log(y))
            plt.ylim(1)
            plt.xlim(1)
            plt.plot(x, (np.e ** lr_b) * (x ** [lr_a]),
            label=f"linear regression {case_name}")
            plt.legend()

            # Helper to format float to string
            rf = lambda v: str(round(v, 2))
            # Helper function to convert radians to string with degrees
            coeff2angle = lambda coeff: rf(abs(np.arctan(coeff)/(2*np.pi)*360))

            # Find the linear function slope in degrees and print coeff
            l = ListTable()
            l.append(["property", "value"])
            l.append(["case", f"Zipf curve for {case_name}"])
            #l.append(["LR slope", coeff2angle(lr_a)])
            l.append(["Best coeffs from log likelihood ",
            f"{rf(math.log(y[0]))} + x*{rf(coeff)}"])
            l.append(["Curve slope", coeff2angle(coeff)])
            print("\n")
            display(l)
```