Statistics

Lecture 5

Centre for Data Science, ITER
Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India.

Contents

Introduction

- Statistics refers to the mathematics and techniques with which we understand data.
- Using statistics, we will try to understand the distribution of data which in turn can be applied to form good machine learning models.

Describing a Single Set of Data

- One obvious description of any dataset is simply the data itself:
  Eg: `num_friends = [100, 49, 41, 40, 25,...]`
- For a small enough dataset, this might even be the best description.
- But for a larger dataset, this is unwidely and probably opaque.

Histogram

- As a first approach, we can create a histogram using `Counter` and `plt.bar`
- As an example we put the friend counts as taken above, into a histogram using the following code:

```
from collections import Counter
import matplotlib.pyplot as plt
friend_counts = Counter(num_friends)
xs = range(101)   #largest value is 100
ys = [friend_counts[x] for x in xs]
plt.bar(xs, ys)
plt.axis([0, 101, 0, 25])   #x-axis and y-axis limits
plt.title("Histogram of Friend Counts")
plt.xlabel("# of friends")
plt.ylabel("# of people")
plt.show()
```

Some statistics on data

- Probably the simplest statistic is the number of data points:

```
num_points = len(num_friends)
```

- We might probably also be interested in the largest and smallest values:

```
largest_value = max(num_friends)
smallest_value = min(num_friends)
```

- To know the values in specific positions, we'll do the following:

```
sorted_values = sorted(num_friends)
smallest_value = sorted_values[0]
second_smallest_value = sorted_values[1]
second_largest_value = sorted_values[-2]
```

Central Tendencies

- Usually, we'll want some notion of where our data is centered.
- **Mean**
  Most commonly we'll use the mean (or average), which is just the sum of the data divided by its count:

$$Mean = \frac{x_1 + \ldots + x_n}{n}$$

  We can implement it as follows:

```
def mean(xs:  List[float]) -> float:
    return sum(xs) / len(xs)
```

Central Tendencies (Contd.)

- **Median**
  We'll also sometimes be interested in the median, which is the *middle-most* value (if the number of data points is odd) or the *average of the two middle- most values* (if the number of data points is even).

Note

The data points should be sorted.

Note

Notice that, unlike the mean, the median doesn't fully depend on every value in your data. For example, if you make the largest point larger (or the smallest point smaller), the middle points remain unchanged, which means so does the median.

Central Tendencies

- Defining median for odd number of elements :

```
def _median_odd(xs:  List[float]) -> float:
    return sorted(xs)[len(xs) // 2]
```

- Defining median for even number of elements :

```
def _median_even(xs:  List[float]) -> float:
    sorted_xs = sorted(xs)
    hi_midpoint = len(xs) // 2
    return (sorted_xs[hi_midpoint - 1] + sorted_xs[hi_midpoint]) / 2
```

- Combining both to form a common function:

```
def median(v:  List[float]) -> float:
    return _median_even(v) if len(v) % 2 == 0 else _median_odd(v)
```

Quantile

- A generalization of the median is the quantile, which represents the value under which a certain percentile of the data lies (the median represents the value under which 50% of the data lies):

```
def quantile(xs:  List[float], p:  float) -> float:
    p_index = int(p * len(xs))
    return sorted(xs)[p_index]
```

Mode

- Mode is the most commonly occurring value in a dataset:

```
def mode(x:  List[float]) -> List[float]:
      counts = Counter(x)
      max_count = max(counts.values())
      return [x_i for x_i, count in counts.items()
            if count == max_count]
```

Dispersion

- Dispersion refers to measures of how spread out our data is.
- Typically they're statistics for which values near zero signify not spread out at all and for which large values (whatever that means) signify very spread out.
- **Range**
  A very simple measure is the **range**, which is just the difference between the largest and smallest elements:

```
def data_range(xs:  List[float]) -> float:
    return max(xs) - min(xs)
```

Variance

- **Variance**
  A more complex measure of dispersion is the variance, which is computed as:

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n - 1}$$

  (Here, $\bar{x}$ is the mean of the data set.)
  This can be implemented as:

```
from scratch.linear_algebra import sum_of_squares
def de_mean(xs:  List[float]) -> List[float]:
     x_bar = mean(xs)
     return [x - x_bar for x in xs]
def variance(xs:  List[float]) -> float:
     assert len(xs) >= 2
     n = len(xs)
     deviations = de_mean(xs)
     return sum_of_squares(deviations) / (n - 1)
```

Standard Deviation

- The range will similarly be in that same unit in which the data is.
- The variance, on the other hand, has units that are the square of the original units.
- **Standard Deviation**
  We often look instead at the standard deviation, which is simply the square root of the variance:

```
import math
def standard_deviation(xs:  List[float]) -> float:
     return math.sqrt(variance(xs))
```

Inter-quartile Range

- A more robust alternative computes the difference between the 75th percentile value and the 25th percentile value:

```
def interquartile_range(xs:  List[float]) -> float:
     return quantile(xs, 0.75) - quantile(xs, 0.25)
```

Covariance

- **Covariance**
  Variance measures how a single variable deviates from its mean whereas *covariance* measures how two variables vary alongside each other from their means:

```
from scratch.linear_algebra import dot
def covariance(xs:  List[float], ys:  List[float]) -> float:
      assert len(xs) == len(ys)
      return dot(de_mean(xs), de_mean(ys)) / (len(xs) - 1)
```

Covariance (Contd.)

- A “large” positive covariance means that x tends to be large when y is large and small when y is small.
- A “large” negative covariance means the opposite—that x tends to be small when y is large and vice versa.

Correlation

- **Correlation**
  It's more common to look at the *correlation*, which divides out the standard deviations of both variables:

```
def correlation(xs:  List[float], ys:  List[float]) -> float:
      stdev_x = standard_deviation(xs)
      stdev_y = standard_deviation(ys)
      if stdev_x > 0 and stdev_y > 0:
            return covariance(xs, ys) / stdev_x / stdev_y
      else:
            return 0
```

Note

The correlation is unitless and always lies between –1 (perfect anti-correlation) and 1 (perfect correlation).

More about Correlation

- A correlation of zero indicates that there is no linear relationship between the two variables.
- However, there may be other sorts of relationships. For example, if:
  x = [-2, -1, 0, 1, 2]
  y = [2, 1, 0, 1, 2]
  then x and y have zero correlation.
  But they certainly have a relationship— each element of y equals the absolute value of the corresponding element of x.
- Correlation looks for a relationship in which knowing how x_i compares to mean(x) gives us information about how y_i compares to mean(y).

More about correlation

- Correlation tells you nothing about how large the relationship is.
  For example:
  The variables:
  x = [-2, -1, 0, 1, 2]
  y = [99.98, 99.99, 100, 100.01, 100.02]
  are perfectly correlated, but (depending on what you're measuring) it's quite possible that this relationship isn't all that interesting.

Correlation and Causation

- If x and y are strongly correlated, that might mean that x causes y, that y causes x, that each causes the other, that some third factor causes both, or nothing at all.
- One way to feel more confident about causality is by conducting randomized trials.

References

[1] Data Science from Scratch: First Principles with Python by Joel Grus

Thank You
Any Questions?