tutorial_entropy_decisions

January 26, 2023

1 Entropy and Decisions tutorial

1.1 1. Entropy

[ ]:
```
%matplotlib inline
from collections import Counter
import numpy as np
import scipy
import sklearn
import statsmodels.api
```

```
/Users/zhuzi/miniforge3/envs/transformers4/lib/python3.9/site-
packages/statsmodels/compat/pandas.py:65: FutureWarning: pandas.Int64Index is
deprecated and will be removed from pandas in a future version. Use pandas.Index
with the appropriate dtype instead.
  from pandas import Int64Index as NumericIndex
```

1.1.1 Compute entropy from a list of probabilities

[ ]:
```
def entropy(probs, base=2):
    assert np.sum(probs) == 1, \
        "ALl probability values should sum up to 1. Got {} instead.".format(np.
 ↪sum(probs))
    res = 0
    for p in probs:
        res -= p * np.log(p) / np.log(base)
    return res

#entropy([1/2, 1/2])
entropy([.25, .5, .25])
#entropy([1])
```

[ ]:
```
1.5
```

1.1.2 Entropy for a random variable with known probability distribution

[ ]:
```
def estimate_probability_by_occurrence(Y_values):
    """
    Note that we are just using the "plain counter" and are approximating
    probabilities with frequencies. The lecture about corpus counting describes
    some methods to address the shortcomings of this approach.
    """
    N = len(Y_values)
    # Count the frequencies
    Y_freq = {}
    for y in Y_values:
        if y not in Y_freq:
            Y_freq[y] = 1
        else:
            Y_freq[y] += 1
    Y_probs = [Y_freq[y]/N for y in Y_freq]
    return Y_probs

def theoretical_vs_estimated_entropy_example_1():
    """
    X_1, X_2 ~ Bern(0.5) independent results of fair coin toss
    Y = X_1 + X_2
    Theoretical value for H(Y) = H([0.25, 0.5, 0.25]) = 1.67 bits
    To estimate by Monte Carlo simulation: we can run N=100000 trials,
    compute the probabilities by simulation.
    """
    print("Theoretical H(Y)={:.4f}".format(entropy([.25, .5, .25])))
    Y_values = []
    N = 100000
    for i in range(N):
        X_1 = np.random.binomial(1, 0.5)
        X_2 = np.random.binomial(1, 0.5)
        Y_values.append(X_1 + X_2)
    Y_probs = estimate_probability_by_occurrence(Y_values)
    print ("Estimated H(Y) from N={} trials is {:.4f}".format(N,␣
 ↪entropy(Y_probs)))

theoretical_vs_estimated_entropy_example_1()
```

```
Theoretical H(Y)=1.5000
Estimated H(Y) from N=100000 trials is 1.5036
```

[ ]:
```
def theoretical_vs_estimated_entropy_example_2():
    """
    X ~ draw from {-1, 0, 1} with equal probability
    Y = X^2
    This is an interesting case because X and Y have 0 correlation but
```

```
        obviously they are dependent. You can verify by computing the␣
↪correlation
        following the definitions.
    """
    #print("H(X)={:.4f}".format(entropy([1/3, 1/3, 1/3])))  # No need to␣
↪simulate here
    print("Theoretical H(Y)={:.4f}".format(entropy([1/3, 2/3])))

    X_values, Y_values = [], []
    N = 100000
    for i in range(N):
        x = np.random.choice([-1, 0, 1])
        X_values.append(x)
        Y_values.append(x**2)
    corr, p = scipy.stats.pearsonr(X_values, Y_values)
    print ("Correlation between X and Y samples are: {:.2f}".format(corr))
    Y_probs = estimate_probability_by_occurrence(Y_values)
    print ("Estimated H(Y) from N={} trials is {:.4f}".format(N,␣
↪entropy(Y_probs)))

theoretical_vs_estimated_entropy_example_2()
```

```
Theoretical H(Y)=0.9183
Correlation between X and Y samples are: 0.00
Estimated H(Y) from N=100000 trials is 0.9174
```

1.2 2. Decisions

Generate UofT students and McGill students' tweet lengths as examples.

[ ]:
```
uoft_students_tweet_lengths = np.random.normal(115, 10, size=(100)).astype(int)
mcgill_students_tweet_lengths = np.random.normal(105, 10, size=(100)).
↪astype(int)
```

1.2.1 One-sample t test

Check if the mean of UofT students' tweets is longer than 100: use one-sample t test: `scipy.stats.ttest_1samp`
Alternative hypothesis: $X > 100$
Null hypothesis: otherwise

[ ]:
```
scipy.stats.ttest_1samp(uoft_students_tweet_lengths, popmean=100,
                        alternative="greater")
```

[ ]:
```
Ttest_1sampResult(statistic=14.438100529082739, pvalue=2.103066011562505e-26)
```

1.2.2 Two-sample t-test

Check if the mean of UofT students' tweets is longer than the mean of McGill students' tweets: use two-sample t test: `scipy.stats.ttest_ind`
Alternative hypothesis: $X_{ut} > X_{mcgill}$
Null hypothesis: otherwise

[ ]:
```
scipy.stats.ttest_ind(uoft_students_tweet_lengths,␣
 ↪mcgill_students_tweet_lengths,
                      equal_var=False,
                      alternative="greater")
# Note 1: equal_var=False does not assume the two group have the same␣
 ↪population variances
# Note 2: alternative argument is only supported after scipy>=1.6.0
# Note 3: Don't change the settings to get smaller pvalues (i.e., p-hacking)
#    You should decide the settings before the experiment, and then
#    report the results as-is.
```

[ ]:
```
Ttest_indResult(statistic=8.453820763898607, pvalue=3.3161145753868867e-15)
```

1.2.3 Check for normality

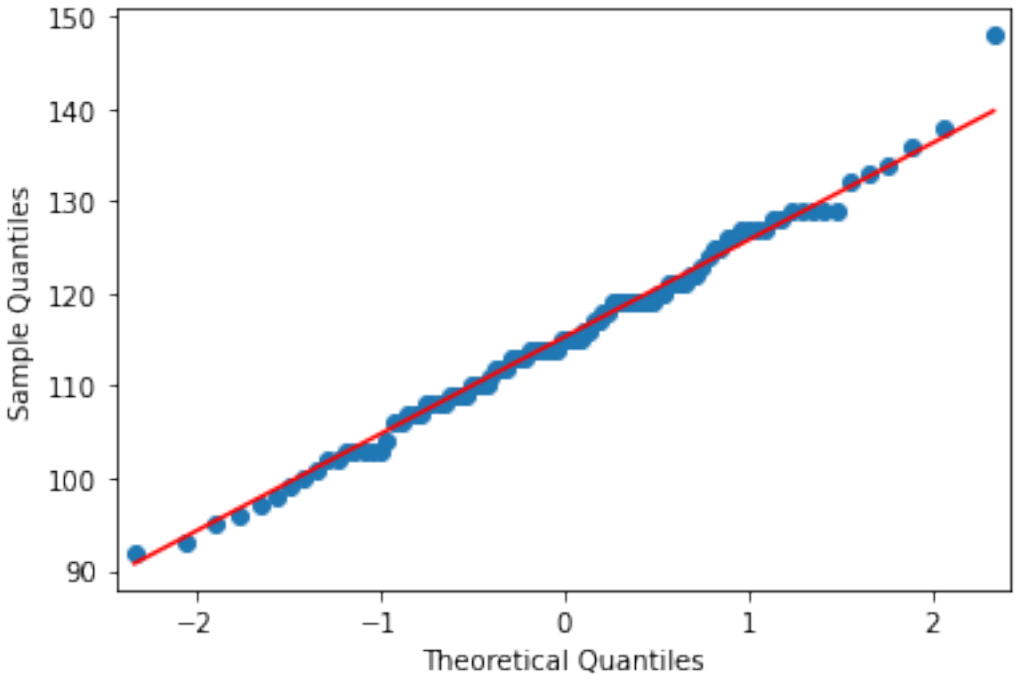
Alternative hypothesis: The distribution is not Gaussian
Null hypothesis: otherwise

[ ]:
```
#result = scipy.stats.shapiro([0,1,1,1,1])  # This array is obviously not␣
 ↪Gaussian
result = scipy.stats.shapiro(uoft_students_tweet_lengths)  # How about this?
if result.pvalue < 0.05:
    print ("p={:.4f} rejects the normality assumption".format(result.pvalue))
else:
    print ("p={:.4f} does not provide sufficient evidence to reject the null.".
 ↪format(result.pvalue))
    print ("i.e., the distribution passes the normality check")
```

```
p=0.8158 does not provide sufficient evidence to reject the null.
i.e., the distribution passes the normality check
```

[ ]:
```
# Visual check with Q-Q plot
statsmodels.api.qqplot(np.array(uoft_students_tweet_lengths), line="s")
print("If the line matches the points, then the distribution is approximately␣
 ↪Gaussian")
```

```
If the line matches the points, then the distribution is approximately Gaussian
```



1.2.4 *t*-test replacements for non-Gaussian data

Here is an example of Mann-Whitney U test, for two-sample t tests.

[ ]:
```
scipy.stats.mannwhitneyu(uoft_students_tweet_lengths,␣
↪mcgill_students_tweet_lengths,
                         alternative="greater")
```

[ ]:
```
MannwhitneyuResult(statistic=8018.5, pvalue=8.034744935596453e-14)
```

[ ]:
```

```