[DAN.IT] EDUCATION

Machine learning

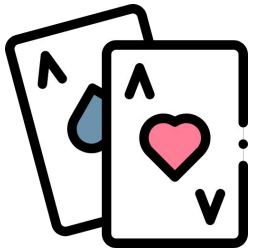
Session 3 - probability

Probability theory

Mathematics of uncertainty

Pierre de Fermat

Blaise Pascal



Probability theory

Primarily concerned with **Random variables.**

A random variable represents possible outcomes of a measurement / experiment / event / etc.

For example - rolling a die, flipping a coin, measuring the amount of rain on a given day.

Random variable - coin flipping

X - result of a coin flip (fair coin)

Value	Heads	Tails
P(X=value)	0.5	0.5

Y - result of a coin flip (loaded coin - 70% heads)

Value	Heads	Tails
P(Y=value)	0.7	0.3

Sampling - “actualizing the event” and obtaining a single value. E.g. concretely flipping a coin.

Random variable - rolling dice

X - result of a die roll (fair die)

Probability distribution

Value	P(X=value)
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

Random variable types

Discrete

Values are categories or predefined set of numbers (finite set)

Examples: coin flip (heads / tails), grade (integer between 0 - 100), roll of dice ([1, 2, 3, 4, 5, 6])

Continuous

Values are real numbers in some range (infinite set)

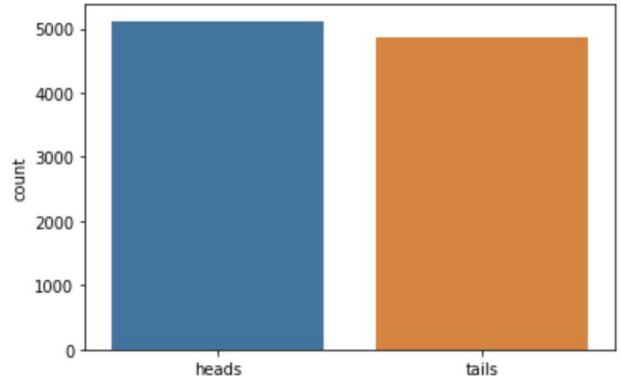
Examples: Temperature measurement, wait time for the bus, child height

Common discrete distributions

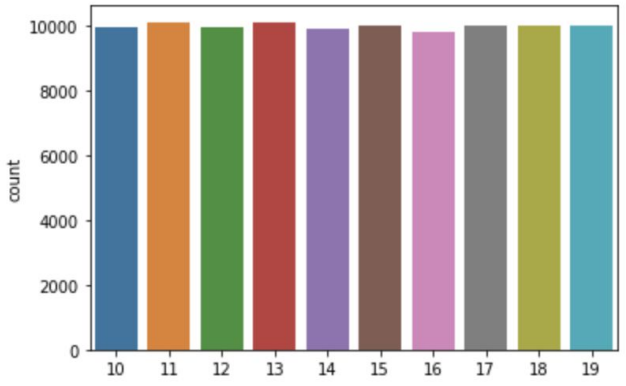
Uniform

Sample size

In [479]:
```
X = np.random.choice(['heads', 'tails'],
                     size=10000, replace=True)
sns.countplot(X)
plt.show()
```



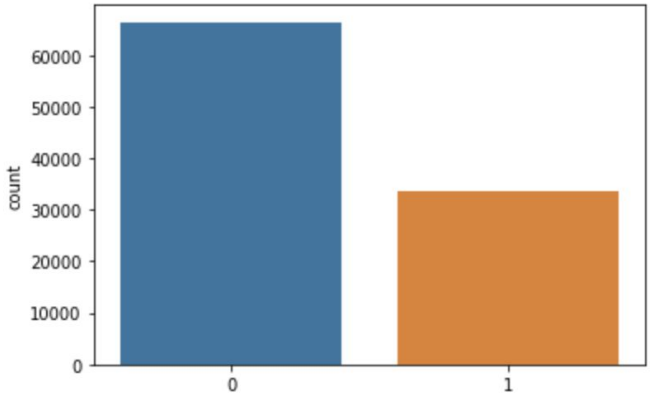
In [484]:
```
X = np.random.randint(10, 20,
                      size=100000)
sns.countplot(X)
plt.show()
```



Common discrete distributions

Bernoulli

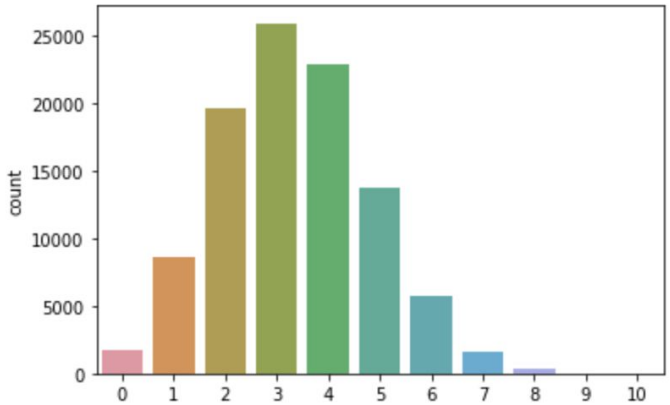
In [488]:
```
X = (np.random.random(size=100000) <= 0.3333).astype(int)
sns.countplot(X)
plt.show()
```



Common discrete distributions

Binomial

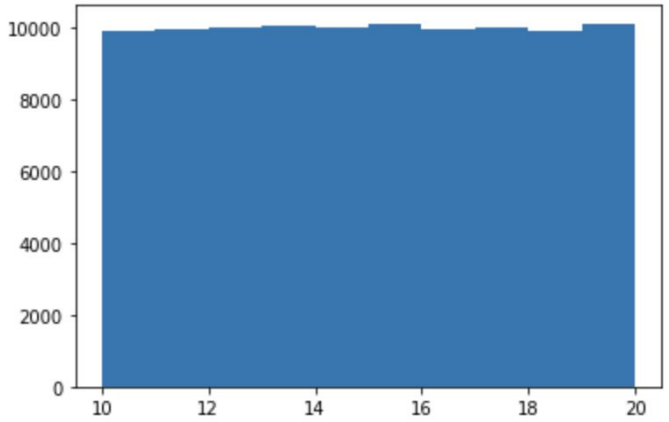
```
In [490]: X = np.random.binomial(p=.3333, n=10, size=100000)
          sns.countplot(X)
          plt.show()
```



Common continuous distributions

Uniform

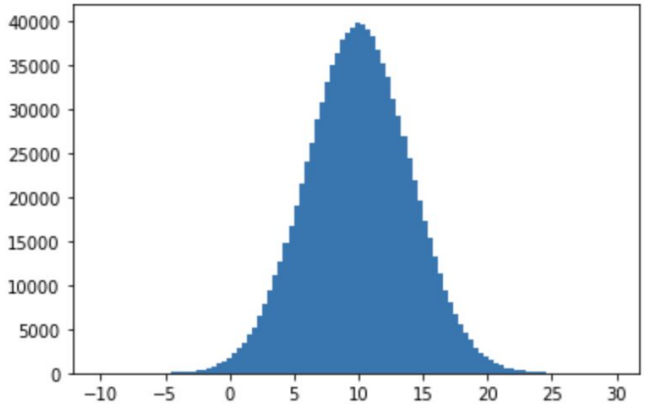
```
In [499]: X = np.random.random(size=100000) * 10 + 10
          plt.hist(X)
          plt.show()
```



Common continuous distributions

Normal

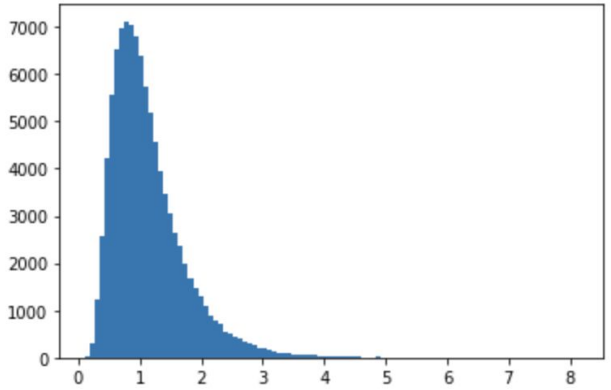
```
In [503]: X = np.random.normal(loc=10, scale=4,
                               size=1000000)
          plt.hist(X, bins=100)
          plt.show()
```



Common continuous distributions

Log-Normal

```
In [532]: X = np.random.lognormal(mean=0, sigma=.5,
                                  size=100000)
          plt.hist(X, bins=100)
          plt.show()
```



Probability mass function

Probability mass function (pmf) - function that defines the distribution of a discrete random variable.

Example: binomial RV with parameters n, p:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Probability axioms (first two)

1. $\forall k,\ P(X = k) \geq 0$

2. $\sum_{k=-\infty}^{\infty} P(X = k) = 1$

Probability density function

Probability density function (pdf) - function that defines the distribution of a continuous random variable.

Example: standard normal:

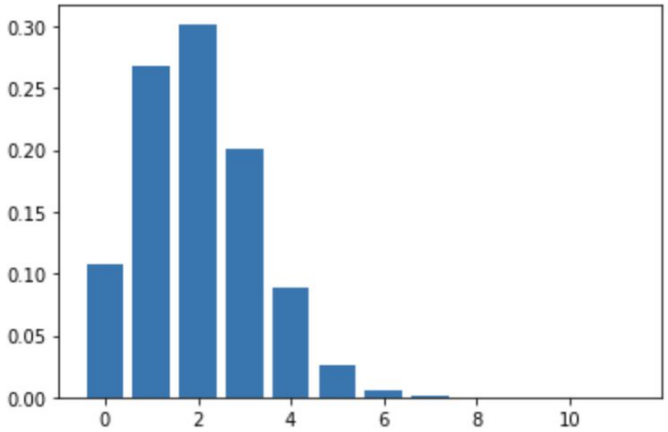
$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Axiom 2 in the continuous case

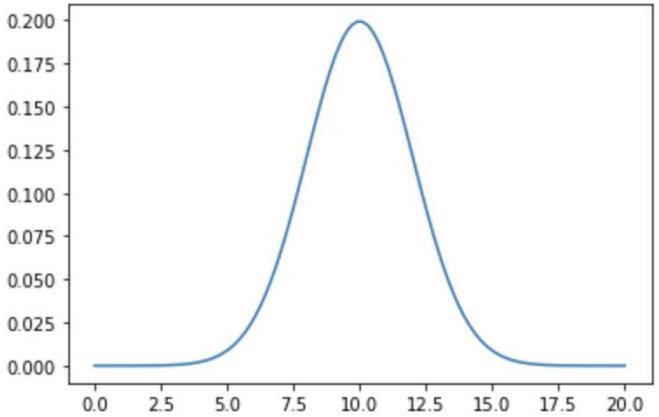
$$\int_{-\infty}^{\infty} p(x)dx = 1$$

scipy.stats

```
x = np.arange(0, 12)
y = scipy.stats.binom(n=10, p=.2).pmf(x)

plt.bar(x, y)
plt.show()
```




```
x = np.linspace(0, 20, 100)
y = scipy.stats.norm(loc=10, scale=2).pdf(x)

plt.plot(x, y)
plt.show()
```



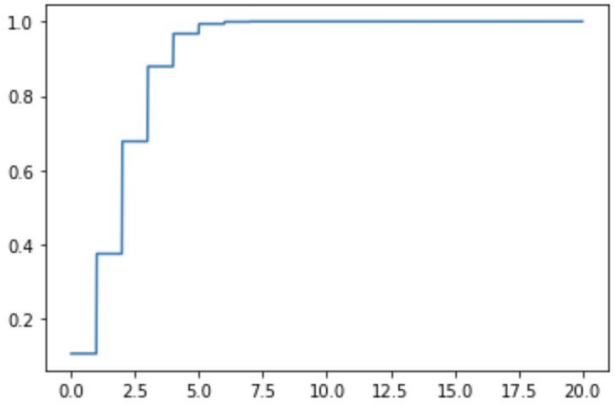
Cumulative distribution function

Cumulative distribution function (cdf) - probability that a sample of a RV will take a value less than or equal to a given value.

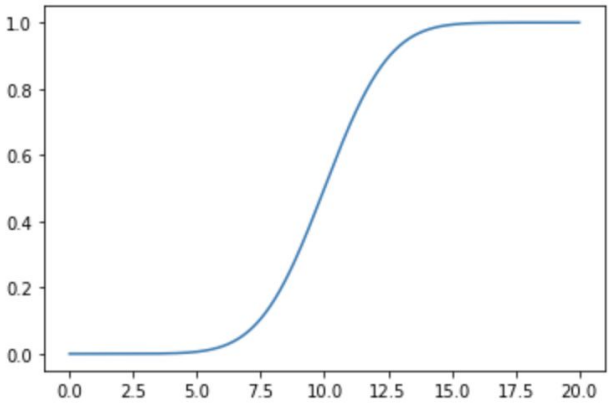
$$cdf(x) = P(X \leq x) = \int_{-\infty}^{x} p(x)dx$$

cdfs in scipy.stats

```
x = np.linspace(0, 20, 1000)
y = scipy.stats.binom(n=10, p=.2).cdf(x)

plt.plot(x, y)
plt.show()
```




```
x = np.linspace(0, 20, 1000)
y = scipy.stats.norm(loc=10, scale=2).cdf(x)

plt.plot(x, y)
plt.show()
```



Multivariate samples & joint distributions

Samples involving more than one random variable

Example: rolling 2 dice

X_1 / X_2	1	2	3	4	5	6
1	1/36	1/36	1/36	1/36	1/36	1/36
2	1/36	1/36	1/36	1/36	1/36	1/36
3	1/36	1/36	1/36	1/36	1/36	1/36
4	1/36	1/36	1/36	1/36	1/36	1/36
5	1/36	1/36	1/36	1/36	1/36	1/36
6	1/36	1/36	1/36	1/36	1/36	1/36

$P(X_1 = 5 \cap X_2 = 3)$

Joint probability distribution

Conditional probability

Probability distribution of one RV given the value of another RV. Denoted P(X|Y).

For example:

P(it will rain tomorrow) = 20%

P(it will rain tomorrow | it rained today) = 70%

Conditional probability

$$P(X|Y) = \frac{P(X \cap Y)}{P(Y)}$$

Independence between 2 RVs

Two RVs are called “independent” if information on one RV doesn’t change the distribution of the other.

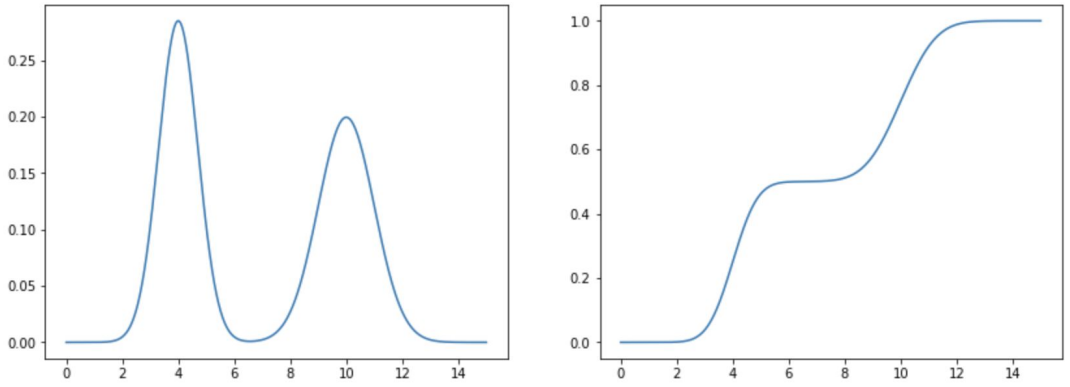
$$P(X|Y) = P(X) \Leftrightarrow P(X \cap Y) = P(X) \cdot P(Y)$$

Inverse transform sampling

A technique for sampling from a distribution given its cdf and a uniform random sample.

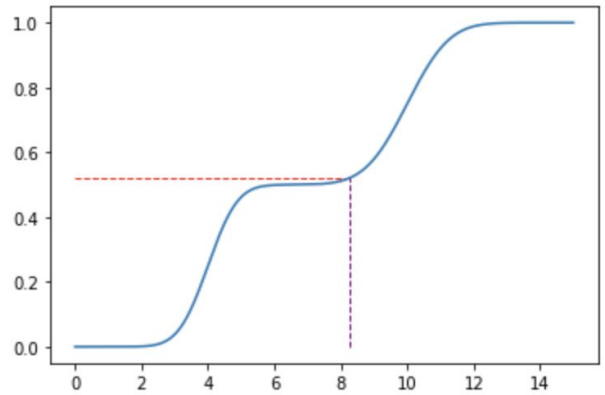
Inverse transform sampling

```
my_pdf = np.vectorize(lambda x: (scipy.stats.norm(loc=4, scale=.7).pdf(x) +
                                 scipy.stats.norm(loc=10, scale=1).pdf(x)) / 2)

my_cdf = np.vectorize(lambda x: (scipy.stats.norm(loc=4, scale=.7).cdf(x) +
                                 scipy.stats.norm(loc=10, scale=1).cdf(x)) / 2)

plt.figure(figsize=(14, 5))

plt.subplot(121)
x = np.linspace(0, 15, 1000)
y = my_pdf(x)
plt.plot(x, y)

plt.subplot(122)
x = np.linspace(0, 15, 1000)
y = my_cdf(x)
plt.plot(x, y)

plt.show()
```



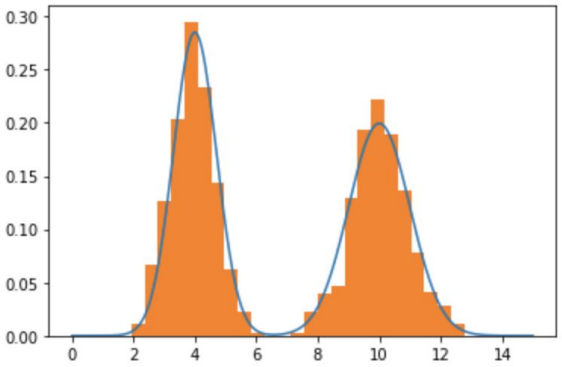
Inverse transform sampling

```
my_cdf = np.vectorize(lambda x: (scipy.stats.norm(loc=4, scale=.7).cdf(x) +
                                 scipy.stats.norm(loc=10, scale=1).cdf(x)) / 2)

u = np.random.random()
v = scipy.optimize.bisect(lambda x: my_cdf(x) - u, 0, 20)

x = np.linspace(0, 15, 1000)
y = my_cdf(x)
plt.plot(x, y)
plt.hlines(y=u, xmin=0, xmax=v, linestyle='--', color='red', linewidth=1)
plt.vlines(x=v, ymin=0, ymax=u, linestyle='--', color='purple', linewidth=1)

plt.show()
```



Inverse transform sampling

```
my_pdf = np.vectorize(lambda x: (scipy.stats.norm(loc=4, scale=.7).pdf(x) +
                                 scipy.stats.norm(loc=10, scale=1).pdf(x)) / 2)
my_cdf = np.vectorize(lambda x: (scipy.stats.norm(loc=4, scale=.7).cdf(x) +
                                 scipy.stats.norm(loc=10, scale=1).cdf(x)) / 2)

samples = []
for i in range(1000):
    u = np.random.random()
    v = scipy.optimize.bisect(lambda x: my_cdf(x) - u, 0, 20)
    samples.append(v)

x = np.linspace(0, 15, 1000)
y = my_pdf(x)
plt.plot(x, y)
plt.hist(samples, bins=25, density=True)

plt.show()
```



Bayes' theorem

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) \cdot P(A \cap B)}{P(A) \cdot P(B)} = \frac{P(A)}{P(B)} \cdot P(B|A)$$

Bayes' theorem

$$P(default) = 1\% \qquad P(predict - default) = 3\%$$

$$P(predict - default \mid default) = 99\%$$

$$P(default \mid predict - default) = ?$$

Bayes' theorem

$$P(default) = 1\% \qquad P(predict - default) = 3\%$$

$$P(predict - default \mid default) = 99\%$$

$$P(default \mid predict - default) =$$

$$= \frac{P(default)}{P(predict - default)} P(predict - default \mid default) =$$

$$= \frac{1\%}{3\%} 99\% = 33\%$$

Likelihood

Suppose we are given data D and want to find parameters of a distribution θ that best fit the data.

$$P(\theta|D) = ?$$

Likelihood

Let’s use Bayes’ theorem!

Prior

$$P(\theta|D) = \frac{P(\theta)}{P(D)} \cdot P(D|\theta)$$

Not interesting

Likelihood

$\mathcal{L}(\theta|D)$

Likelihood under iid

i.i.d - a common assumption about samples:

independent and identically distributed.

Under i.i.d:

$$\mathcal{L}(\theta|D) = \prod_{d \in D} p(d|\theta)$$

Log-likelihood under iid

Under i.i.d, for ease of computation (and numeric stability), we can take the log of the likelihood, called simply log-likelihood:

$$\ell(\theta|D) = \log \prod_{d \in D} p(d|\theta) = \sum_{d \in D} \log p(d|\theta)$$

Maximum likelihood estimates (MLE)

```python
def experiment(p, n, N):
    return np.random.binomial(n=n, p=p, size=N)
```

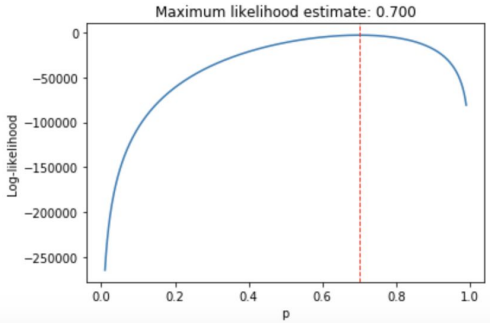
```python
D = experiment(.7, 100, 1000)
```

```python
def single_log_likelihood(p, n_heads, n_tails):
    n = n_heads + n_tails
    return np.log(scipy.special.binom(n, n_heads) * p ** n_heads * (1 - p) ** n_tails)


def log_likelihood(p, n, D):
    return np.sum([single_log_likelihood(p, d, n - d) for d in D])
```

```python
possible_ps = np.linspace(.01, .99, 1000)
```

```python
log_likelihoods = [log_likelihood(p, 100, D) for p in possible_ps]
```

```python
plt.plot(possible_ps, log_likelihoods)
mle_p = possible_ps[np.argmax(log_likelihoods)]
plt.axvline(mle_p, c='red', linewidth=1, linestyle='--')
plt.title(f'Maximum likelihood estimate: {mle_p:.3f}')
plt.xlabel('p')
plt.ylabel('Log-likelihood')
plt.show()
```



Kernel density estimation (KDE)

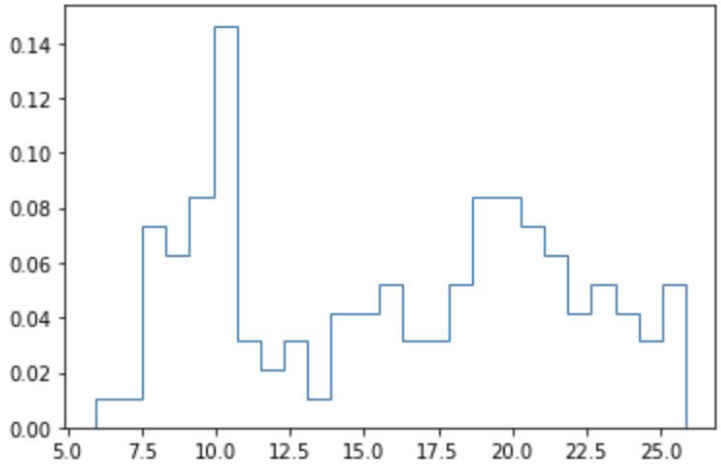
Creating a smooth continuous pdf from samples.

Normalized histograms are examples of non-smooth (stepwise) pdfs from samples.

Kernel density estimation (KDE)

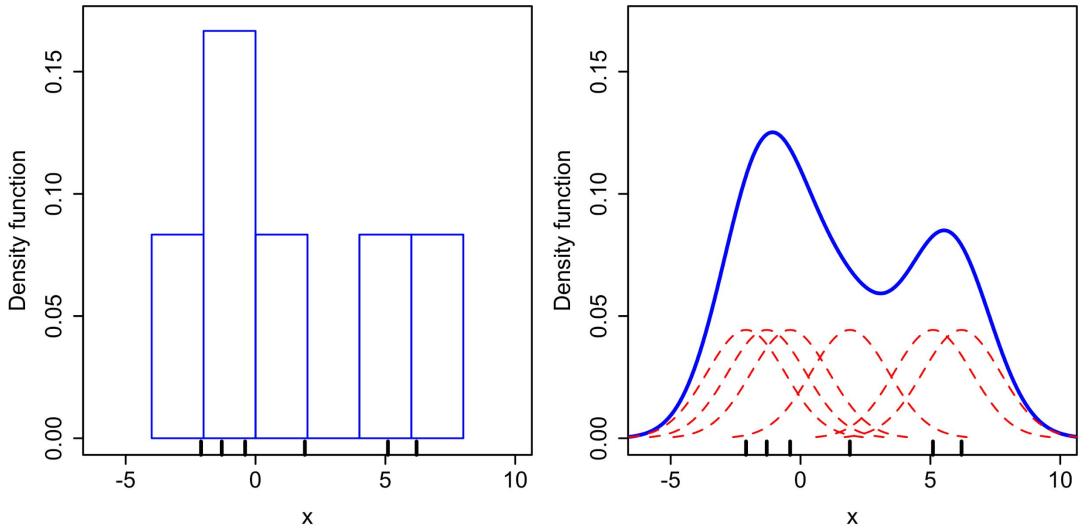
```
X = np.concatenate([np.random.normal(loc=10, scale=2, size=50),
                    np.random.normal(loc=20, scale=3.2, size=70)])
```

```
plt.hist(X, density=True, histtype='step', bins=25)
plt.show()
```



Kernel density estimation (KDE)

$$pdf(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

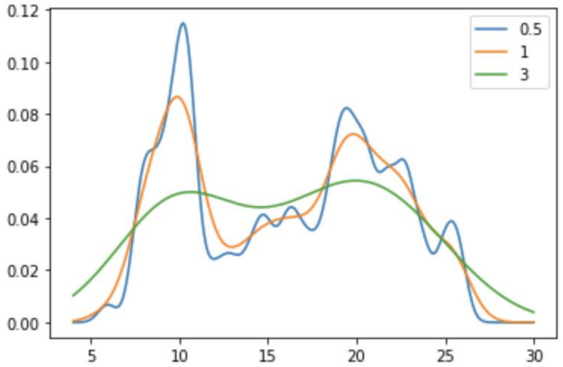


Kernel density estimation (KDE)

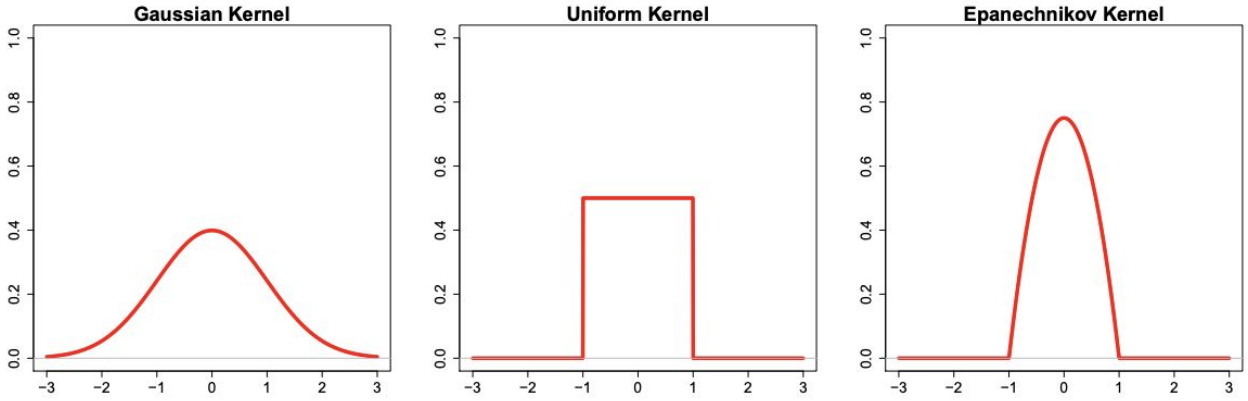
```python
def normal_kernel(x):
    return 1 / np.sqrt(2 * np.pi) * np.exp(-x ** 2 / 2)
```

```python
def KDE(kernel, bandwidth, X):
    return np.vectorize(lambda v: kernel((X - v) / bandwidth).mean() / bandwidth)
```

```python
for h in [.5, 1, 3]:
    kde_pdf = KDE(normal_kernel, h, X)
    xx = np.linspace(4, 30, 1000)
    yy = kde_pdf(xx)
    plt.plot(xx, yy, label=h)

plt.legend()
plt.show()
```



KDE - common kernels

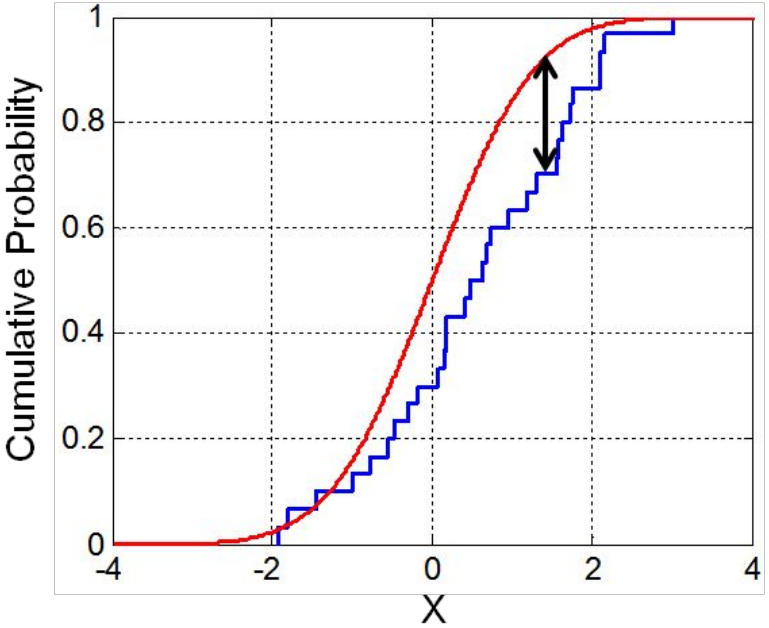


Statistical distance measures

Different ways of quantifying the distance between distributions.

Kolmogorov-Smirnov

Maximum difference between cdfs



Kullback-Leibler divergence

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

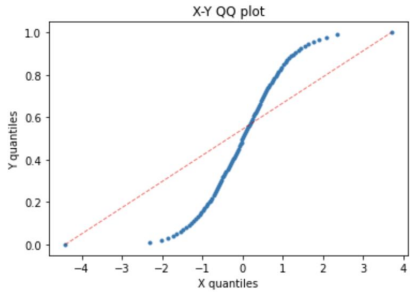
Q-Q plots

A visual comparison of distributions

```
x = np.random.normal(size=10000)
y = np.random.random(size=10000)
```

```
x_q = np.quantile(x, np.linspace(0, 1, 100))
y_q = np.quantile(y, np.linspace(0, 1, 100))
```

```
plt.scatter(x_q, y_q, s=9)
plt.plot([x_q[0], x_q[-1]], [y_q[0], y_q[-1]], linewidth=1, alpha=.6, linestyle='--', c='r')
plt.xlabel('X quantiles')
plt.ylabel('Y quantiles')
plt.title('X-Y QQ plot')
plt.show()
```



References

- http://www.cs.toronto.edu/~urtasun/courses/CSC2515/Tutorial-ReviewProbability.pdf
- http://web.stanford.edu/class/ee269/probability.pdf
- https://towardsdatascience.com/machine-learning-probability-statistics-f830f8c09326
- https://mathisonian.github.io/kde/
- https://machinelearningmastery.com/probability-density-estimation/
- https://en.wikipedia.org/wiki/Kernel_density_estimation
- http://faculty.washington.edu/yenchic/18W_425/Lec6_hist_KDE.pdf
- https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test
- https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence