

# Bayesian A/B Testing

Beta Distribution and Multi-Arm Bandit

# A/B Testing

## (frequentist)

- define a metric
- determine parameters of interest for study (number of observations, power, significance threshold, and so on)
- run test, without checking results, until number of observations has been achieved
- calculate p-value associated with hypothesis test
- report p-value and suggestion for action

# A/B Testing

(frequentist)

- can you say “it is 95% likely that site A is better than site B”?
- can you stop test early based on surprising data?
- can you update the test while it is running?

# A/B Testing

## (Bayesian)

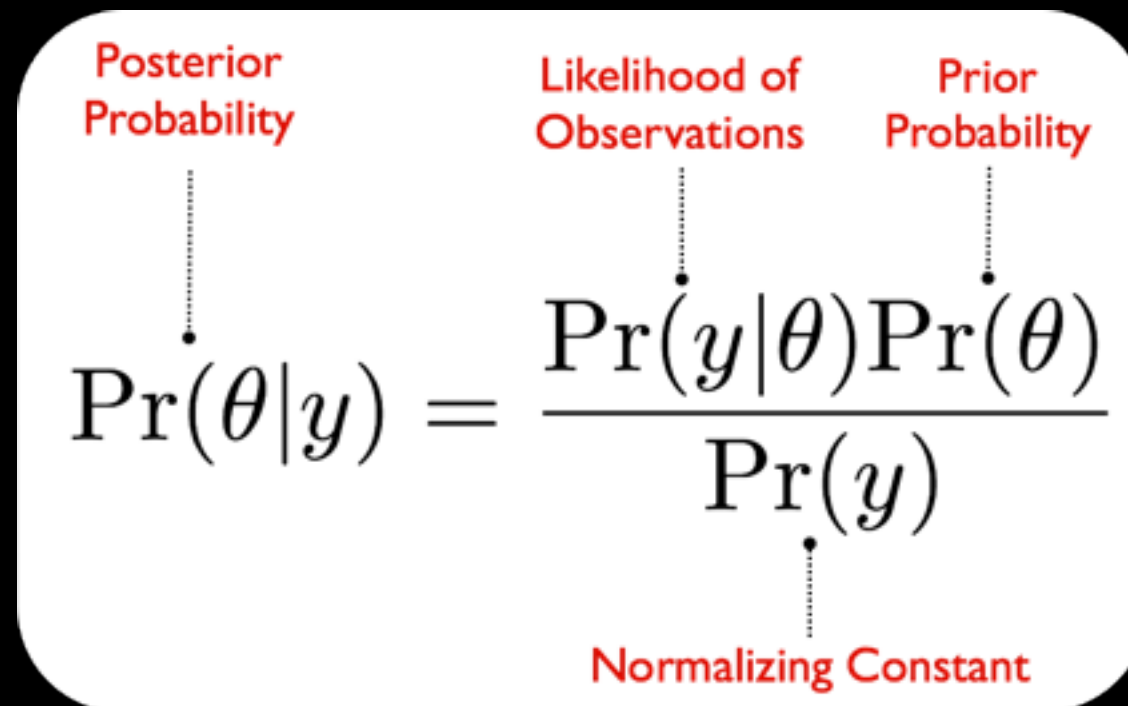
- define a metric
- run test, continually monitor results
- at any time calculate probability that  $A > B$  or vice versa
- suggest course of action based on probabilities calculated

# A/B Testing

## (Bayesian)

- can you say “it is 95% likely that site A is better than site B”?
- can you stop test early based on surprising data?
- can you update the test while it is running?

# Bayes Theorem



The diagram shows the Bayes Theorem equation with labels and arrows indicating the components:

$$\text{Posterior Probability } \Pr(\theta|y) = \frac{\text{Likelihood of Observations } \Pr(y|\theta) \text{ Prior Probability } \Pr(\theta)}{\text{Normalizing Constant } \Pr(y)}$$

Labels and arrows:

- Posterior Probability** (red text) with a dashed arrow pointing to  $\Pr(\theta|y)$ .
- Likelihood of Observations** (red text) with a dashed arrow pointing to  $\Pr(y|\theta)$ .
- Prior Probability** (red text) with a dashed arrow pointing to  $\Pr(\theta)$ .
- Normalizing Constant** (red text) with a dashed arrow pointing to  $\Pr(y)$ .

- **prior:** initial belief
- **likelihood:** likelihood of data given outcome
- **posterior:** updated belief

# Bayes Theorem

$$\text{posterior} \propto \text{prior} \times \text{likelihood}$$

# Binomial (likelihood)

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

- $p$ : conversion rate (between 0 and 1)
- $n$ : number of visitors
- $k$ : number of conversions



# Conjugate Priors

posterior  $\propto$  prior  $\times$  likelihood

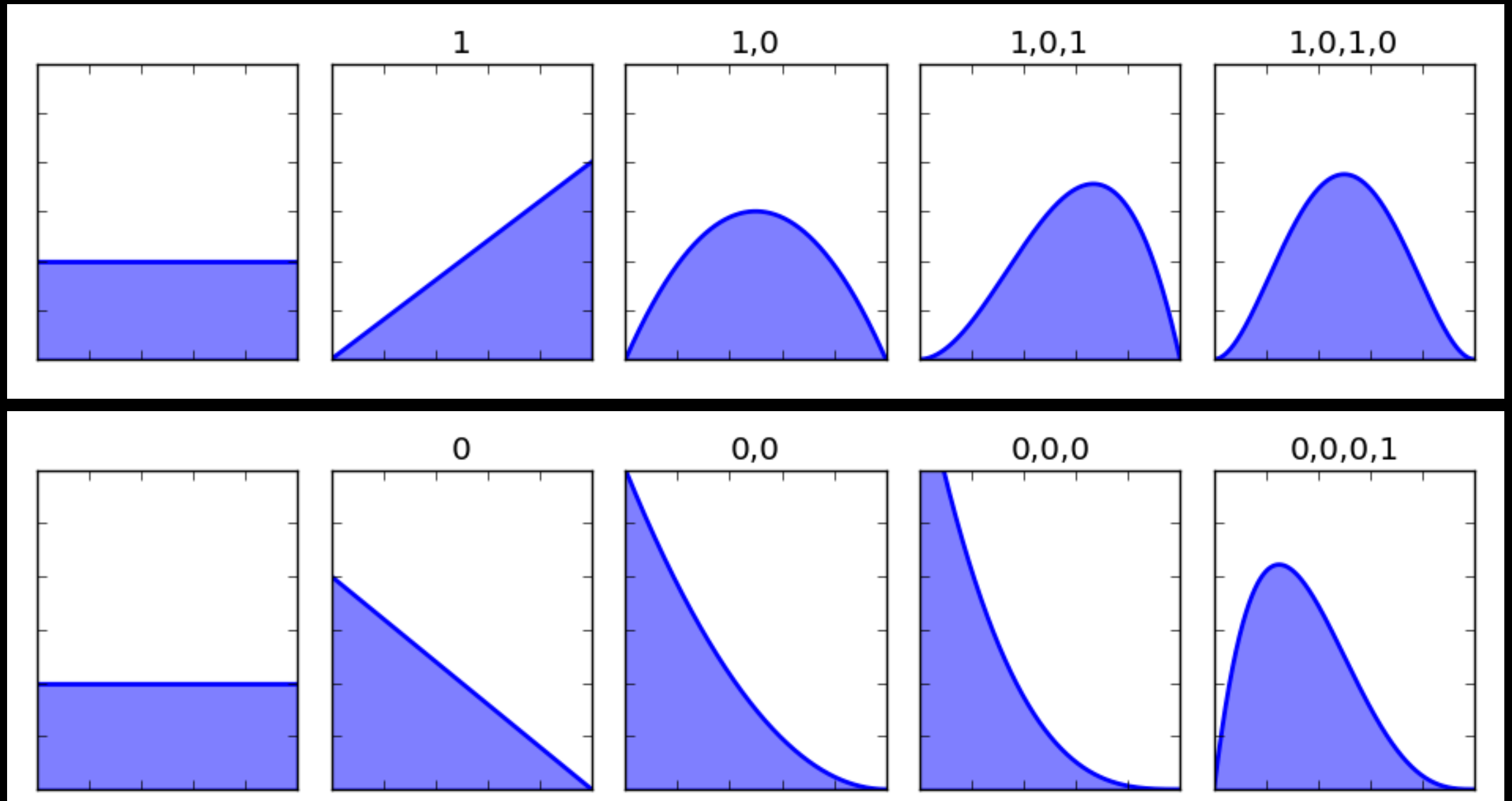
beta  $\propto$  beta  $\times$  binomial

# Beta Distribution

$$\frac{p^{\alpha-1} (1-p)^{\beta-1}}{B(\alpha, \beta)}$$

- $p$ : conversion rate (between 0 and 1)
- $\alpha, \beta$ : shape parameters
  - $\alpha = 1 + \text{number of conversions}$
  - $\beta = 1 + \text{number of non conversions}$
- beta function (B) is for normalization
- $\alpha = \beta = 1$  gives the uniform distribution

# Beta Distribution



1 = conversion

0 = non conversion

# Conjugate Priors

posterior  $\propto$  prior  $\times$  likelihood

beta  $\propto$  beta  $\times$  binomial

THE MATH:

posterior  $\propto$  prior  $\times$  likelihood

$$= \frac{p^{\alpha-1}(1-p)^{\beta-1}}{B(a, b)} \times \binom{n}{k} p^k (1-p)^{n-k}$$

$$\propto p^{\alpha-1}(1-p)^{\beta-1} \times p^k (1-p)^{n-k}$$

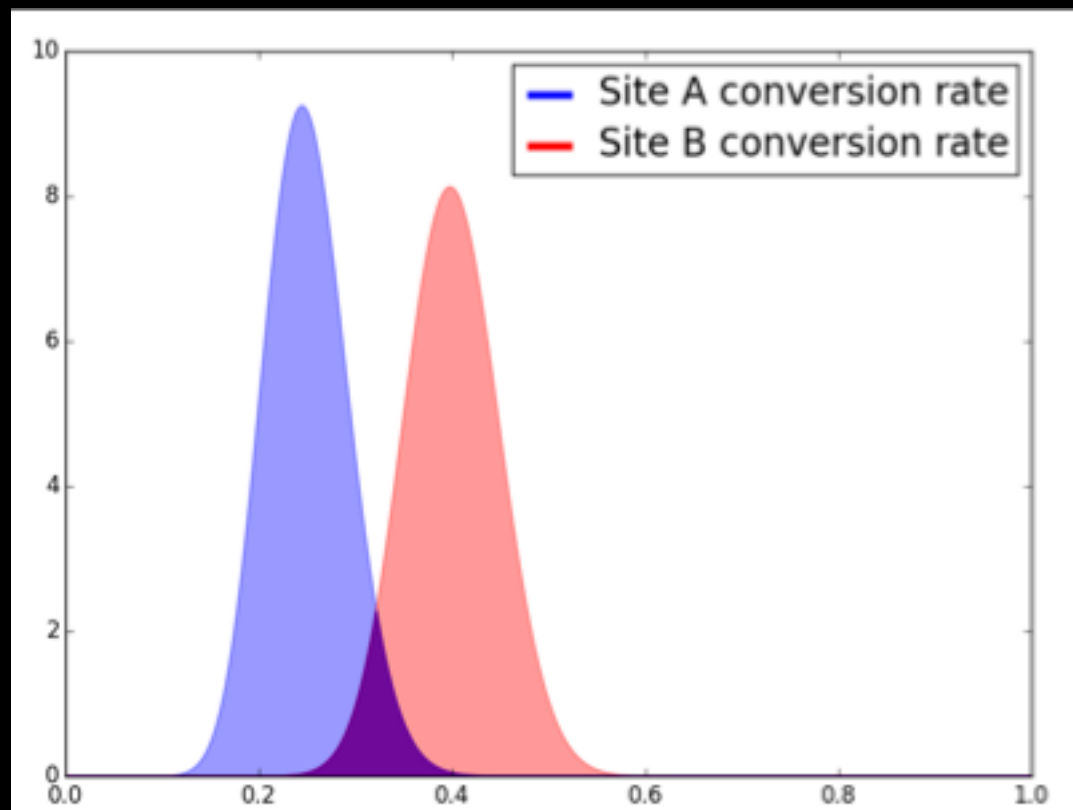
$$\propto p^{\alpha+k-1}(1-p)^{\beta+n-k-1}$$

result is a beta distribution with shape parameters:

$$\alpha + k \text{ and } \beta + n - k$$

# A/B Testing

- we want to know if this is true:  
**conversion rate of site A > conversion rate of site B**
- we can also answer if this is true:  
**conversion rate of site A > conversion rate of site B + 5%**



Method:

- sample repeatedly from each distribution
- count how often site A has a high rate than site B

# code

```
num_samples = 10000
A = np.random.beta(1 + num_clicks_A,
                   1 + num_views_A - num_clicks_A,
                   size=num_samples)
B = np.random.beta(1 + num_clicks_B,
                   1 + num_views_B - num_clicks_B,
                   size=num_samples)
### The probability that A wins:
print np.sum(A > B) / float(num_samples)
### The probability that A > B + 0.5%:
print np.sum(A > (B + 0.05)) / float(num_samples)
```

# to sum up

- the bayesian framework has significant advantages over the frequentist in the context of A/B testing
  - can answer the questions we want to ask
  - allows for more flexibility in the testing framework
- beta distributions are a natural fit for A/B testing
  - conjugate to both Bernoulli and binomial distributions
  - maps naturally onto a probability

# Multi-Arm Bandit

Smarter A/B Testing



# boring A/B testing

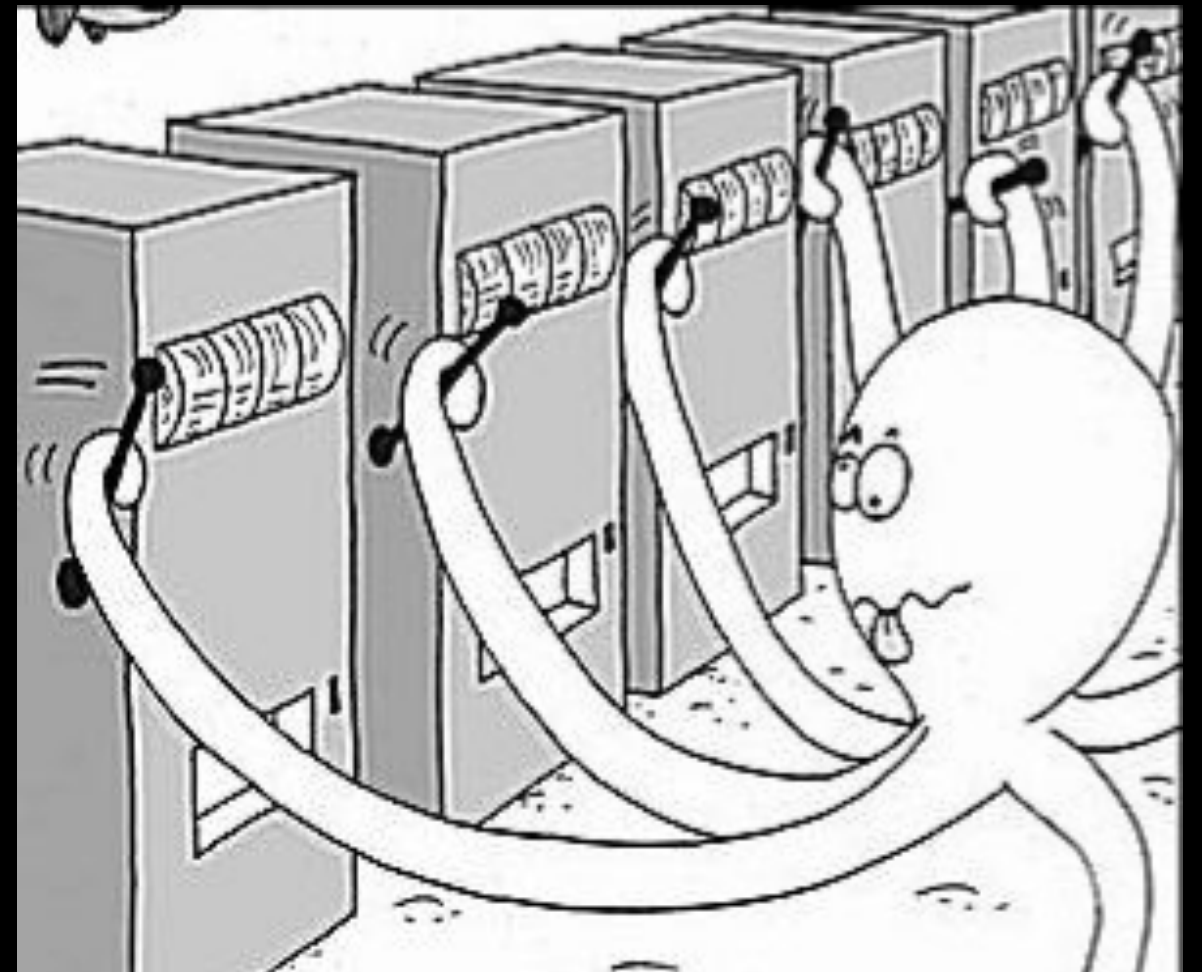
- to start, use pure exploration, assign equal numbers to each group
- then: pure exploitation, stop the experiment and send all users too more successful version of the site

# boring A/B testing

- but, every user you send to the worse site during the test is lost money
- can we minimize our loses during the exploration phase?

# enter the bandits

- start exploiting the likely best solution before ending the exploration phase
- each version of the site is a bandit
- how do we pick which bandit to play



# regret

- we can evaluate the multi-armed bandit by thinking about **regret**
- we want to minimize the time we spend on non-optimum payouts

$$\begin{aligned}\text{regret} &= \sum_{i=1}^k (p_{\text{opt}} - p_i) \\ &= k \cdot p_{\text{opt}} - \sum_{i=1}^k p_i\end{aligned}$$

# regret

- regret lets us evaluate in the abstract the behavior of bandit algorithms
- it is not generally something you calculate as it requires knowledge of the true probability

# epsilon-greedy

- explore with a fixed probability
  - epsilon
  - often 10% or less
- if you do not explore, play the bandit with the best performance seen so far

# ucb1

- calculate a probability for each bandit
- use the bandit with the maximum of the following equation

$$p_A + \sqrt{\frac{2 \log N}{n_A}}$$

where  $n_A$  = number of times bandit A has been played  
and  $N$  = total number of times any bandit has been played

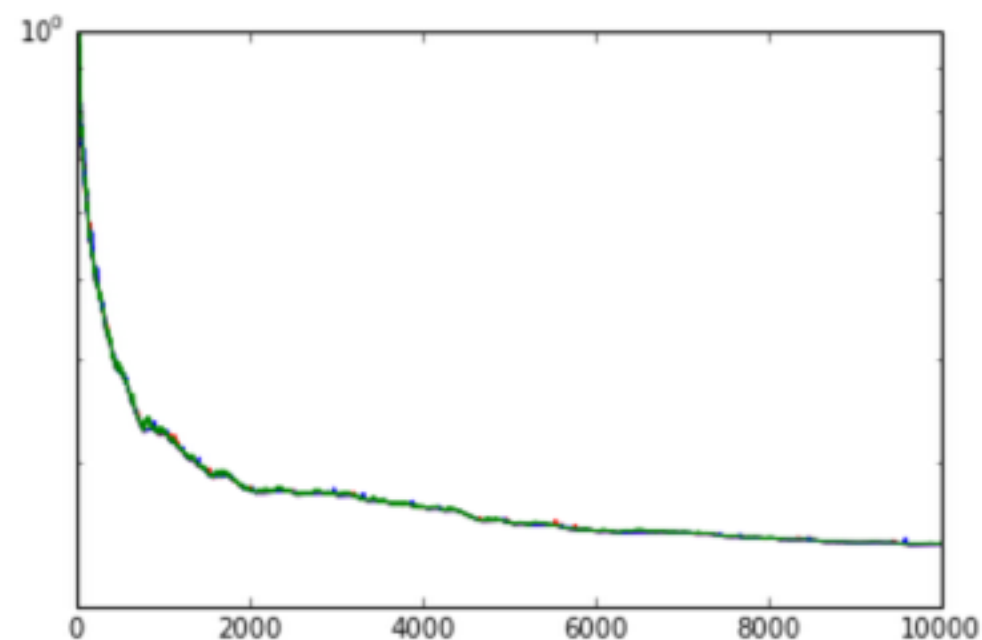
# simulation of ucb1 bandits

payouts are [0.05, 0.1, 0.2]

```
In [62]: probs = [0.05,0.1,0.2]
played_bandits = play_bandits(bandits,probs)
colors = ['r','b','g']
for i,bandit in enumerate(played_bandits):
    plt.plot(bandit['ucb'],color=colors[i])
    print i,len(bandit['plays']),max(bandit['plays'])
plt.yscale('log')
plt.ylim(.2,1)
```

```
0 528 9633
1 707 9639
2 8765 9999
```

Out[62]: (0.2, 1)





# softmax

- choose the bandit random in proportion to its estimated value:

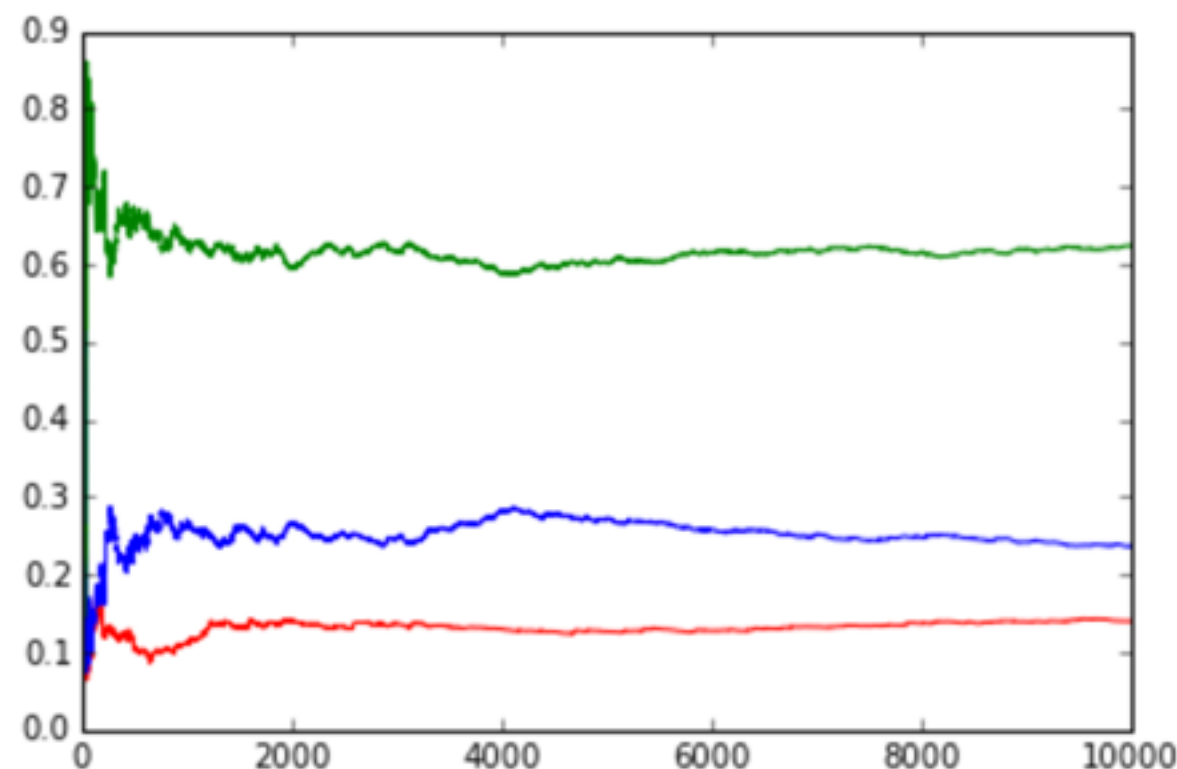
$$\frac{e^{p_A/\tau}}{e^{p_A/\tau} + e^{p_B/\tau} + e^{p_C/\tau}}$$

# simulation of softmax bandits

payouts are [0.05, 0.1, 0.2]

```
In [101]: probs = [0.05,0.1,0.2]
played_bandits = play_bandits(bandits,probs)
colors = ['r','b','g']
for i,bandit in enumerate(played_bandits):
    print i, len(bandit['plays']), max(bandit['plays'])
    plt.plot(bandit['softmax'],color=colors[i])
```

```
0 1342 9999
1 2460 9996
2 6198 9998
```

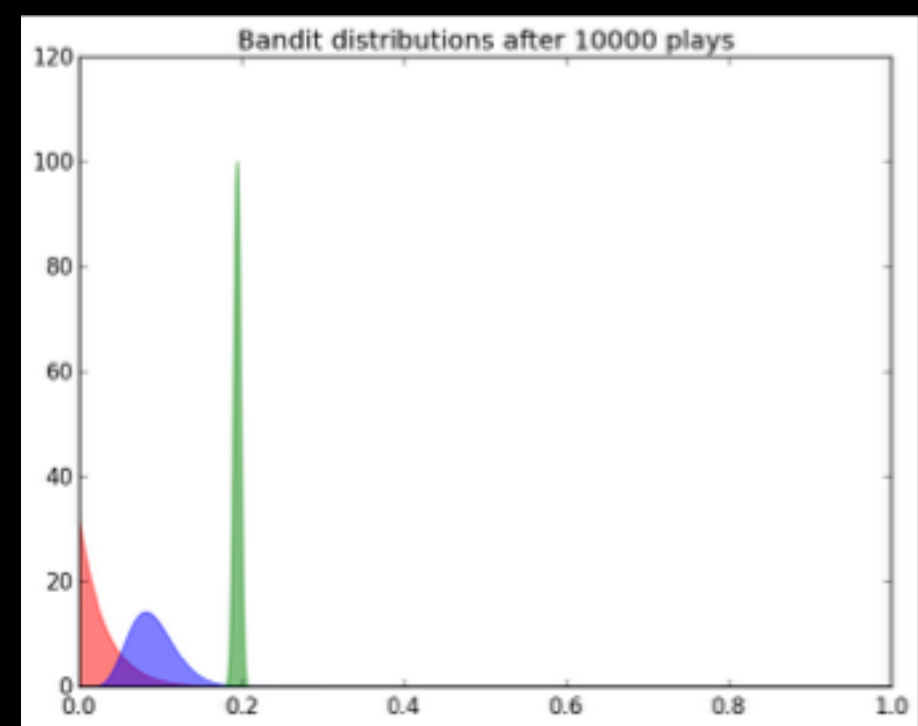
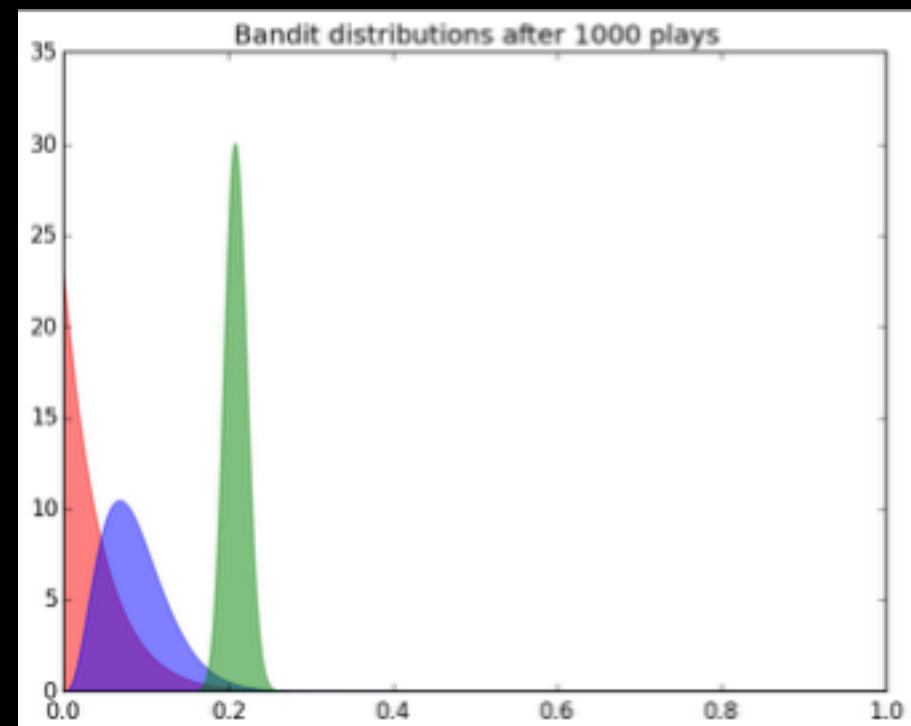
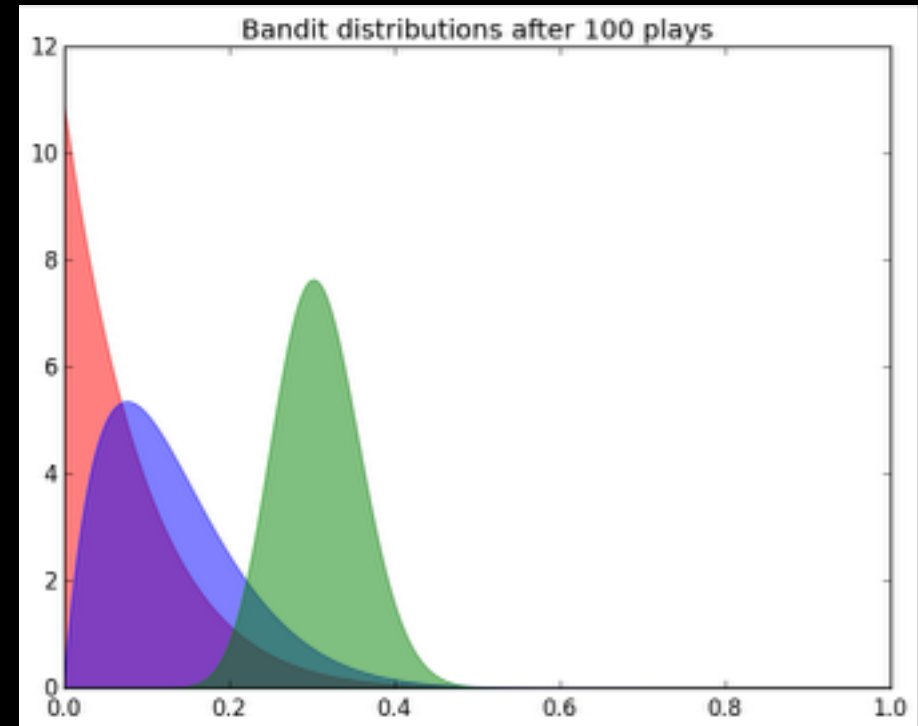
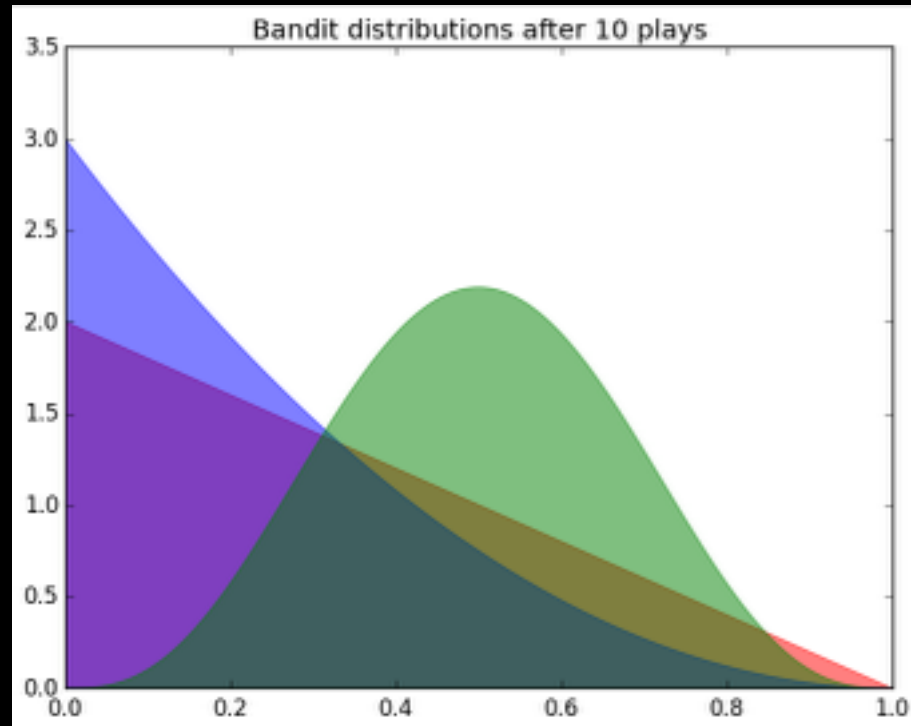


# Bayesian bandit

- use bayesian updates to model the bandits
- each bandit has an associated beta distribution with parameters:  
 $\alpha = 1 + \text{number of times bandit has won}$   
 $\beta = 1 + \text{number of times bandit has lost}$
- sample from each bandit's distribution and play the bandit with the highest value
- will naturally converge on the bandit with the best payout

# simulations of bayesian bandits

payouts are  $[0.05, 0.1, 0.2]$



# to sum up

- multi-armed bandit problem maps onto A/B testing naturally
  - can easily expand to testing many things simultaneously
  - can tune exploitation versus exploration
  - can combine with bayesian methods in a natural way
- there are many solutions to the multi-armed bandit problem
  - depend on how you want to make the trade offs
  - if payoffs change over time interpretation of bandits regret changes
  - will play around with different simulations during the sprint