non-personal-recommenders

March 15, 2020

1 Non-Personal Recommendation Techniques

- We don't always know our user's preferences, e.g. new users
- We can user patterns from the populations behavior to make suggestions
- In many examples, no machine learning is required

1.1 Popularity Ranking

- other people really liked it, so you probably will too

1.1.1 Challenges

- McDonald's is popular, should we recommend that? Probably not
- top 40 music is popular, but many people won't like it
- age can be an important factor to consider, we should usually consider this
- news from last week is popular, but we don't want to see old news

1.2 Affinity Analysis

- also known as Association Rule mining, identifying Frequent Item Sets, or Market Basket analysis
- technique is used for making "context-based" recommendations
- if you're buying an iPhone, you might also want an iPhone case

1.2.1 Conditional Probability

- compute the probability of purchasing item A given context of item B
- this will lead to many false positives, we need to be smarter

$$P(A \mid B) = \frac{count(A, B)}{count(B)}$$

1.2.2 Lift

- in association rule mining, a Lift score is the performance ratio of a target model divided by a random choice or default model
- Lift can also be used for context-based recommendations

- the Lift score increases (> 1) when buying one item (B) makes buying another item (A) more likely

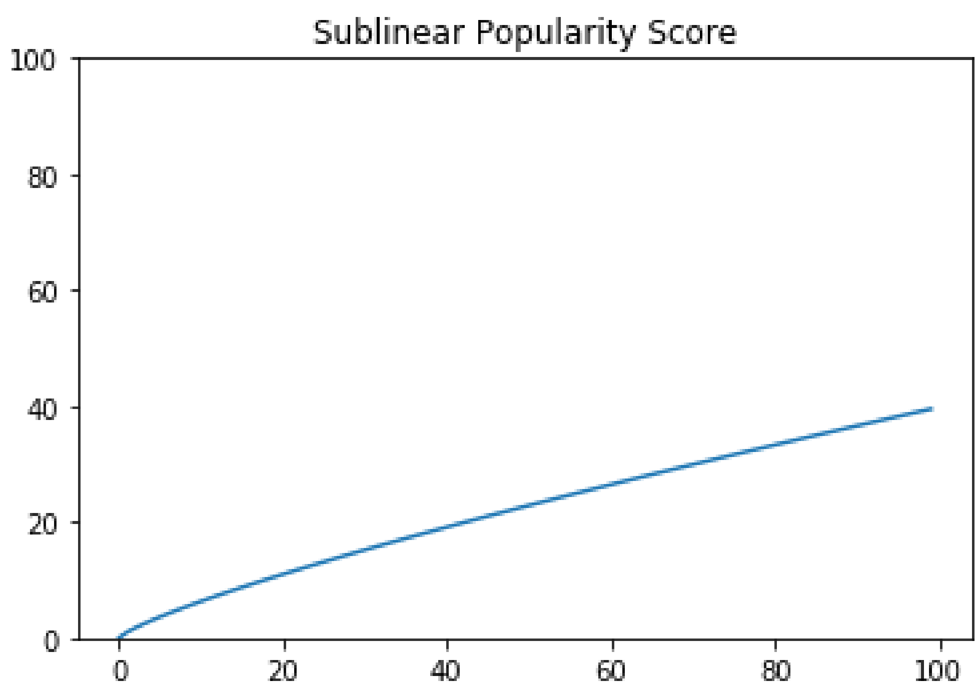
$$Lift = \frac{p(A, B)}{p(A)p(B)} = \frac{p(A \mid B)}{p(A)} = \frac{p(B \mid A)}{p(B)}$$

1.3 Hacker News - Popularity Over Time

$$score = \frac{(upvotes - downvotes - 1)^{0.8}}{(age + 2)^{gravity}} \times penalty$$

- Hacker News ranking considers up votes, down votes, age, and penalties
- numerator is a measure of article popularity
- sublinear numerator (exponent < 1) because
  - age must overpower popularity over time
  - first 100 votes should carry more meaning than 1000-1100 votes
  - very few articles make up most of the votes, and many articles have just a few votes
- penalty terms include self-posts, controversial posts, etc

[20]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 100, 1)
y = x ** 0.8
plt.plot(x, y)
plt.ylim(0, 100)
plt.title('Sublinear Popularity Score');
```



1.4 Average Rating Ranking

- average item ratings is the easiest way to score items

1.4.1 Challenges

- not always as simple as upvotes and downvotes, e.g. 5 star systems
- some items have very few ratings, i.e. confidence of average is low

1.4.2 Using Confidence Intervals

$$95\%CI = (\bar{X} + -z_{score}\frac{s}{\sqrt{N}})$$

- as total number of ratings increases, estimated averaged approaches the expected rating
- compute the confidence interval for an item's rating and use the lower bound
- popularity will increase score by creating tighter confidence intervals, i.e. higher lower bounds

1.4.3 Problems with Average Ratings

- 5 star ratings can leverage Wilson's interval
- we can convert each possible rating to a upvote and downvote percentage, e.g. 0 star is 1 downvote 0 upvote, 3 star is 0.5 downvote and 0.5 upvote, and 5 star is 0 downvotes and 1 upvote
- what if there are 0 ratings? We need to use smoothing to prevent divide by 0
- Laplace smoothing is common solution, also used in NLP
- this allows us to obtain smooth transition as number of voters increases

1.4.4 Explore-Exploit Dilemma

- if you're at casino and there is row of slot machines, you can't tell which one is the best, you must play them to see which one has best rewards
- you need to calculate the win rate for each slot machine to determine which one to play (exploit)
- how many times should you play each slot machine (explore)?
  - if you play too few, your estimate will have large confidence interval
  - if you play too many, you are missing opportunity to explore other machines
- explore-exploit is faced when recommending items to users, and exploration is needed to encourage new, novel items
- exploring too much runs the risk of bad recommendations, but not exploring can be a bad user experience

1.5 Bayesian Methods for the Explore-Exploit Challenge

AKA Bayesian Bandits

- we want to know the probability that a user will click on a recommendation
- can be applied to AB testing, Ad clicks, selecting stocks, etc
- we can draw recommendations from beta distributions and update these distributions with the customer feedback
- this allows for easy online learning

1.5.1 Beta Distribution

A continuous probability distribution defined on [0,1] with 2 parameters, α and β.

$$\text{Beta PDF} = f(x;\alpha,\beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$$

When $\alpha = 1$ and $\beta = 1$, the beta distribution is equivalent to uniform distribution.

1.5.2 Bayes Theorem

$$p(H \mid D) = \frac{p(H)p(D \mid H)}{p(D)}$$

$$p(H \mid D) = posterior$$

$$p(H) = \text{prior, our belief before observing evidence}$$

$$p(D \mid H) = \text{likelihood of seeing evidence D if hypothesis H is correct}$$

$$p(D) = \text{likelihood of evidence under any circumstance, normalizing factor}$$

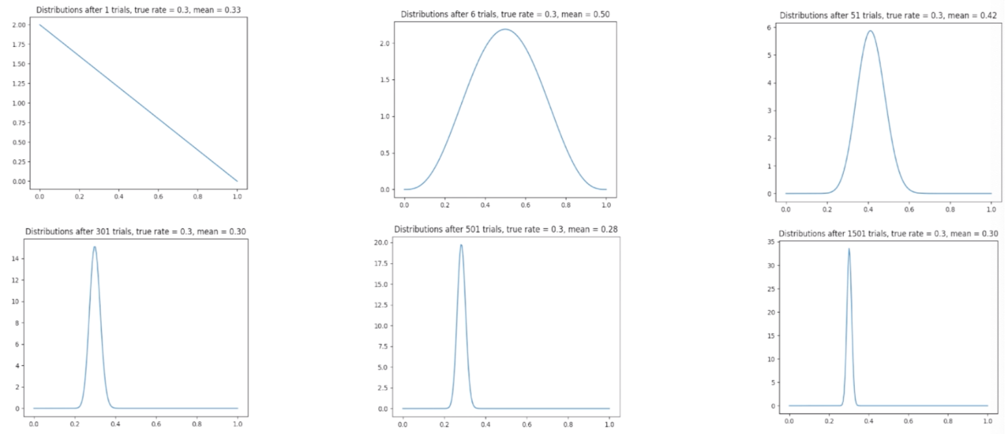
1.5.3 Strategy

In the beginning, we don't have prior beliefs because we have not observed anything. So we start with $\alpha = 1$ and $\beta = 1$, i.e. all items have uniform, or equal probability.

1. Sample random variable from each of 3 asset's beta distributions.
2. Select the maximum random variable and show it to our user.
3. Determine feedback on item, e.g. user click.
4. Update the prior for selected item using feedback from (3).
5. Repeat.

1.5.4 Posterior Over Time

Posteriors converge on expected value as we receive feedback from users.



alt text

1.5.5 Ranking with Bayesian Methods

- ranking scores is non-deterministic, we must sample from the posterior beta distributions
- by sampling, ranking is intelligently random
- this encourages *exploration*

1.5.6 Bayesian Bandits Example Code

See bayesian-bandits notebook

1.6 Supervised Learning

1.6.1 Demographic-Based Learners

- we can try to predict various targets using simple learning algorithms
  - did user buy product?
  - click on ad?
  - click on article?
  - sign up for newsletter?
  - make an accountt?
  - what rating did they give an item?
- common demographic features include
  - age, gender, religion, location, race, occupation
  - education level, marital status, socio-economic status
- other data from site
  - date/location of sign up
  - device type, mobile?
  - page views
  - credit card history
  - purchase history

- can purchase data
  - Acxiom
  - Intelius

1.6.2 How to Incorporate Product Data

- above list includes only user features, how do we include product features?
- can create a separate model for each item, but will not scale to many products
- can add some product feature flags to the user feature vector and feed to model

1.6.3 Latent Variable Models

- instead of explicit user features like age, gender, etc., we can learn implicit features
- these learnes features are not as interpretable, but they are mathematically optimal and yield better results
- this means we don't have to feature engineer features, saves time

1.7 Page Rank

The Page Rank of a page is the probability that a user would end up on a page if they surfed the Internet randomly for an infinite amount of time.

Page Rank is just a score, and it can be applied to various recommender systems.

1.7.1 Markov Models

- Markov Model finds x_t given x_{t-1}
- Visual explanations of MMs

- similar to bigrams in NLP - building a probabilistic language model that allows prediction of next word given current word
  - what is probability of "cats" given "love" P(cats | love)
- bigrams only consider 2 words at a time, which is limited, and more advanced language models use DNN models with recurrent and attention layers
- instead of thinking about each item as a word, we think of it as a state $x(t)$
- $x(t)$ only depends on x_{t-1}

$$p(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_1) = p(x_t \mid x_{t-1})$$

- the **Transition* Probability Matrix A** defines the probability of transitioning from state j to state i
- valid probabilities - rows of the matrix must sum up to 1
- AKA as stochastic matrix or Markov matrix

$$A(i, j) = p(x_t = j \mid x_{t-1} = i)$$

- how to calculate probabilities in transition matrix?

$$p(rainy \mid sunny) = \frac{count(sunny \rightarrow rainy)}{count(sunny)}$$

- can use this method to calculate the probability of observing a sentence "the quick brown fox jumps over the lazy dog"

$$p(the)p(quick \mid the)p(brown \mid quick)\ldots$$

$$p(x_1, \ldots, x_T) = p(x_1)\prod_{t=2}^{T} p(x_t \mid x_{t-1})$$

- what if the test set contains a bigram that never occurs in the training data?
- this *zero* probability will produce a 0 probability due to multiplication
- therefore, we use add-1 smoothinng to prevent 0s in below equation, where V is equal to the total number of states

$$p(x_t \mid x_{t-1}) = \frac{count(i \rightarrow j) + 1}{count(i) + V}$$

- **state distribution** π is the probability of being in a state at a given time
- Example: if there are 2 possible states, sunny and rainy, $\pi(t)$ will be a vector of size 2 $[p(x_t = sunny), p(x_t = rainy)]$
- we can calculate $\pi(t+1)$ using bayes rule, i.e. we can calculate the next state distribution

$$\pi_{t+1}(j) = \sum_{i=1}^{M} A(i,j)\pi_t(i)$$

$$\pi_{t+1} = \pi_t A$$

- we can predict the state *k* steps into the future

$$\pi_{t+k} = \pi_t A^k$$

- each web page on internet is modelled as a state in a Markov Model
- we model transition probability using links on a page

$$p(x_t = j \mid x_{t-1} = i) = \frac{1}{n(i)}\text{if i links to j, otw 0}$$

$$n(i) = \#linksonpagei$$

- there are billions of web pages, so very sparse and mostly 0, so smoothing must be applied

1.8 Evaluating Rankings

- goal is to return list of items sorted by predicted ranking
- how can we determine if our predicted ranking is good?
- a number of metrics exist, recall, precision, etc, but these are not always the best method because sometimes we need models to explore new items, e.g. novel/surprising items. Users don't always want more of the same
- no particular ranking can be "correct", we have to instead optimize metrics like revenue, impressions, clicks using A/B tests

bayesian-bandits-code

March 15, 2020

[55]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import beta, uniform
import seaborn as sns
```

0.1 Example 1

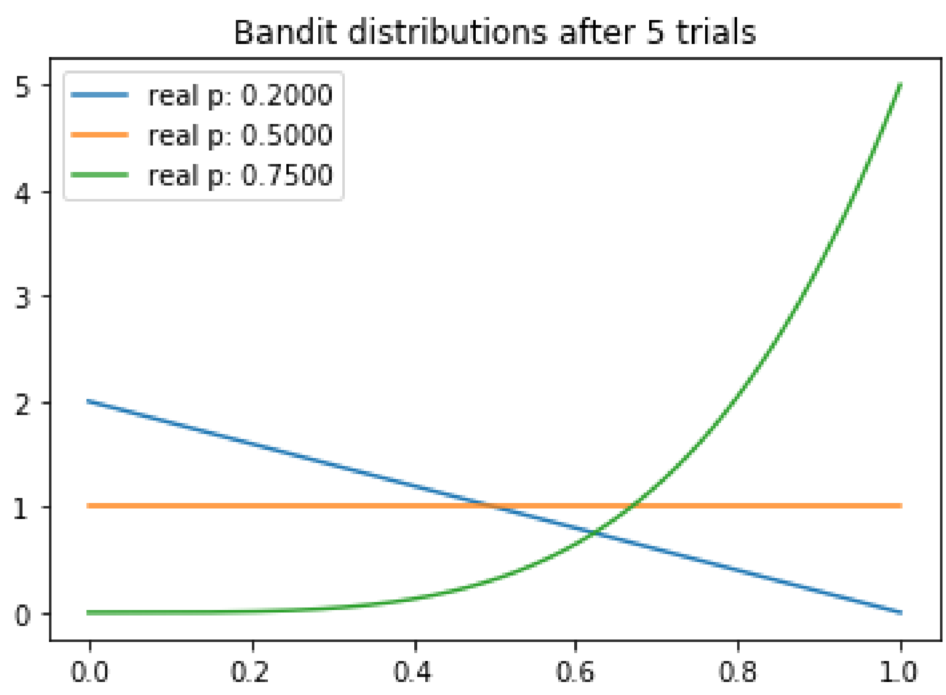
[36]:
```
NUM_TRIALS = 2000
BANDIT_PROBABILITIES = [0.2, 0.5, 0.75]
```

[30]:
```
class Bandit(object):
  def __init__(self, p):
    self.p = p
    self.a = 1
    self.b = 1

  def pull(self):
    return np.random.random() < self.p

  def sample(self):
    return np.random.beta(self.a, self.b)

  def update(self, x):
    self.a += x
    self.b += 1 - x


def plot(bandits, trial):
  x = np.linspace(0, 1, 200)
  for b in bandits:
    y = beta.pdf(x, b.a, b.b)
    plt.plot(x, y, label='real p: %.4f' % b.p)
  plt.title('Bandit distributions after %s trials' % trial)
  plt.legend()
  plt.show()
```

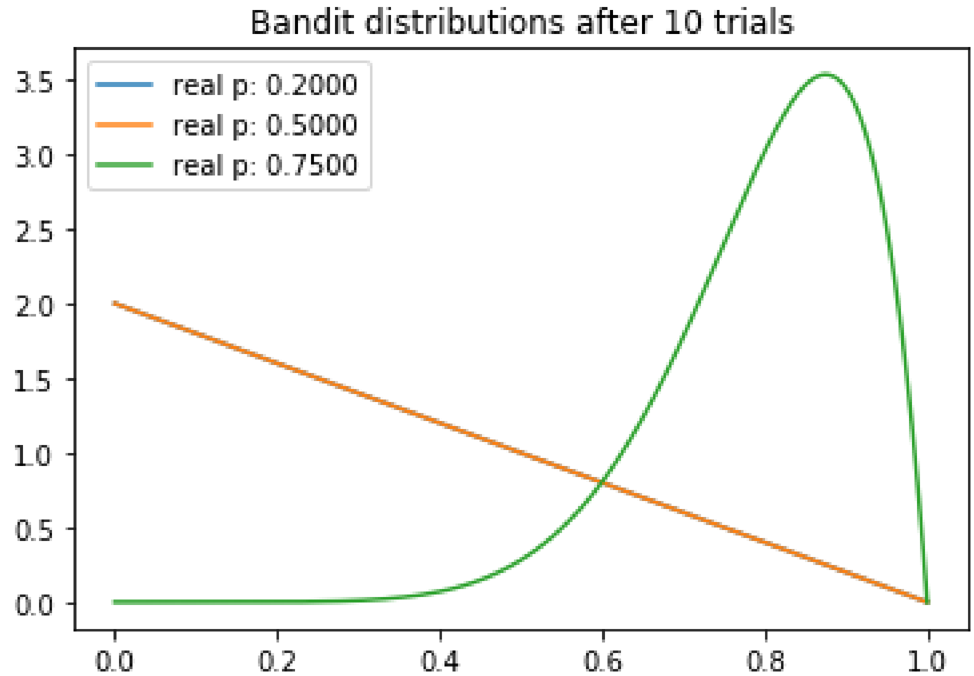
```
def experiment():
  bandits = [Bandit(p) for p in BANDIT_PROBABILITIES]

  sample_points = [5, 10, 20, 50, 100, 200, 500, 1000, 1500, 1999]

  for i in range(NUM_TRIALS):
    bestb = None
    maxsample = -1
    allsamples = []
    for b in bandits:
      sample = b.sample()
      allsamples.append('%.4f' % sample)
      if sample > maxsample:
        maxsample = sample
        bestb = b
    if i in sample_points:
      print('current samples: %s' % allsamples)
      plot(bandits, i)

    x = bestb.pull()
    bestb.update(x)
```

[31]:
```
experiment();
```

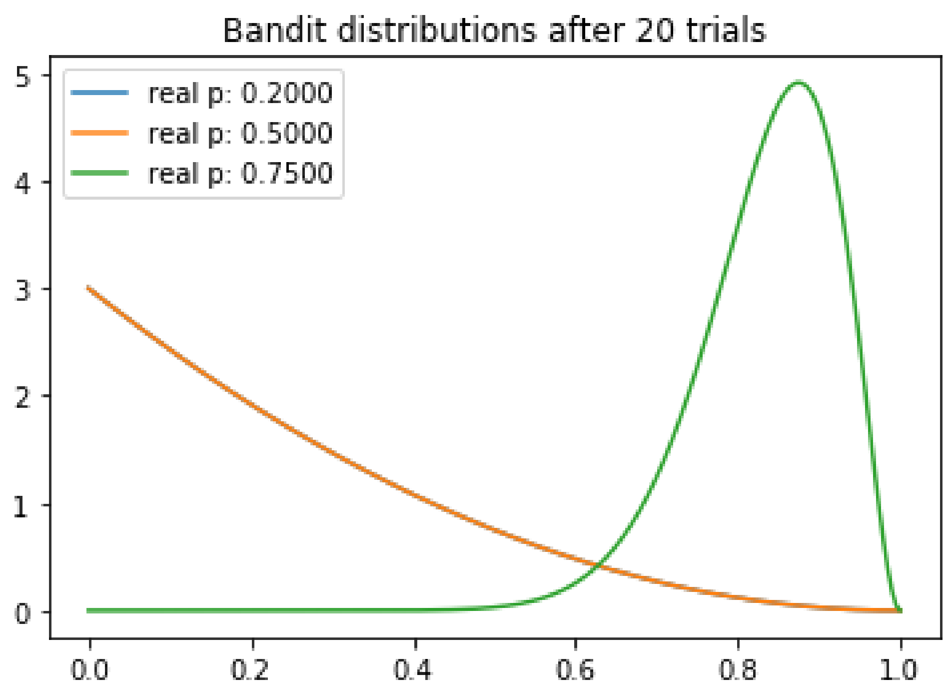
```
current samples: ['0.7618', '0.4567', '0.8549']
```



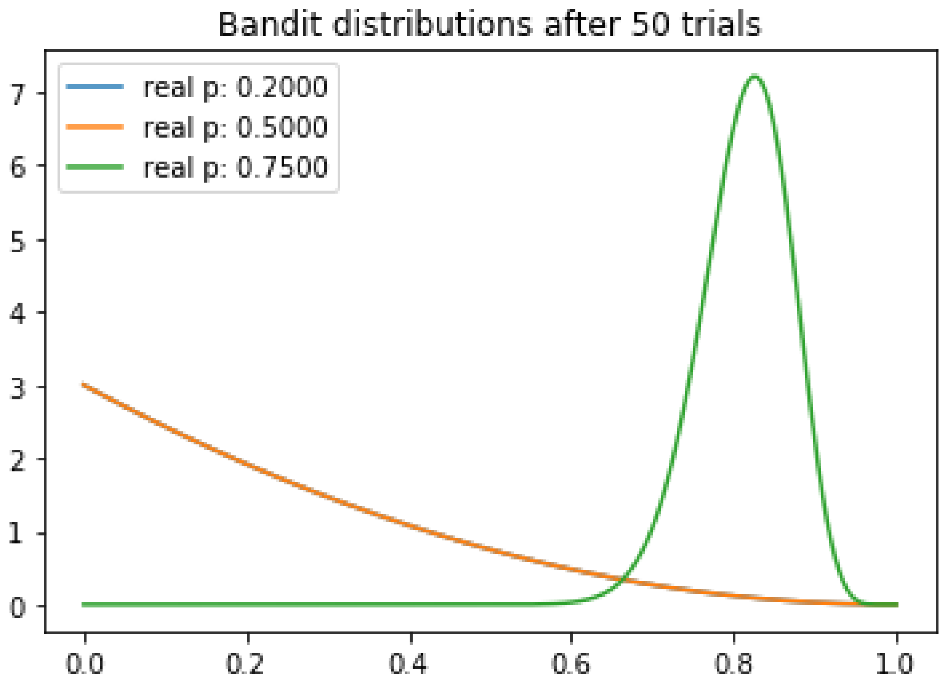
```
current samples: ['0.1935', '0.0163', '0.7698']
```



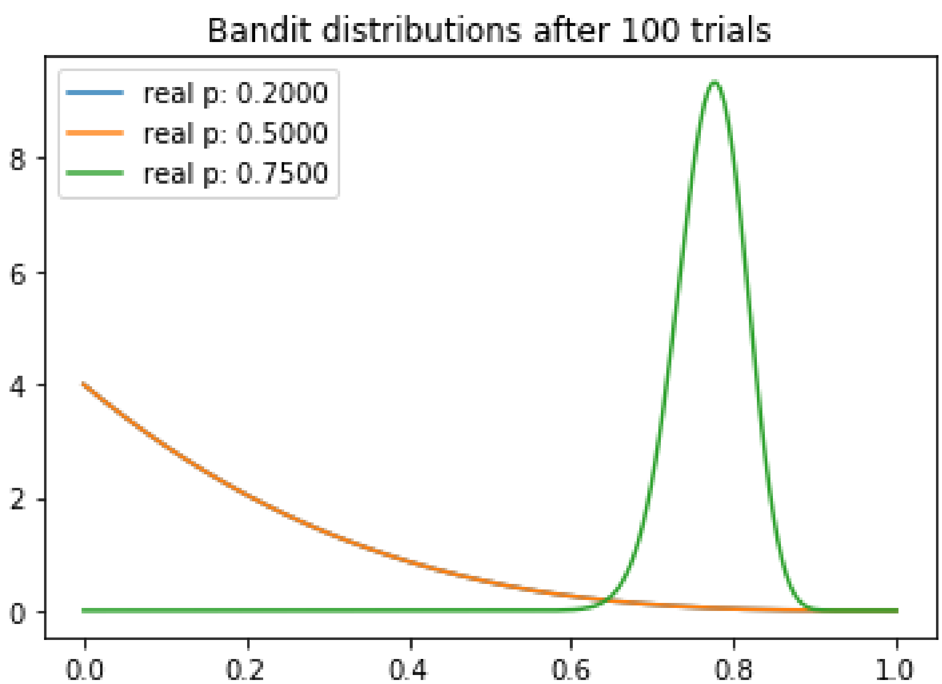

```
current samples: ['0.0806', '0.5004', '0.9161']
```



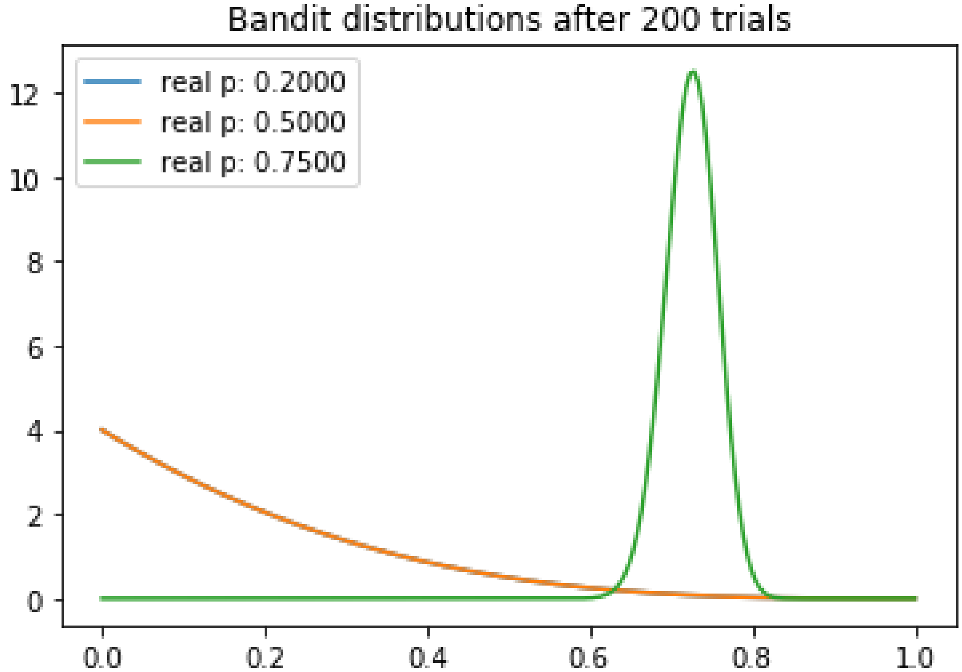
```
current samples: ['0.2050', '0.1388', '0.8971']
```



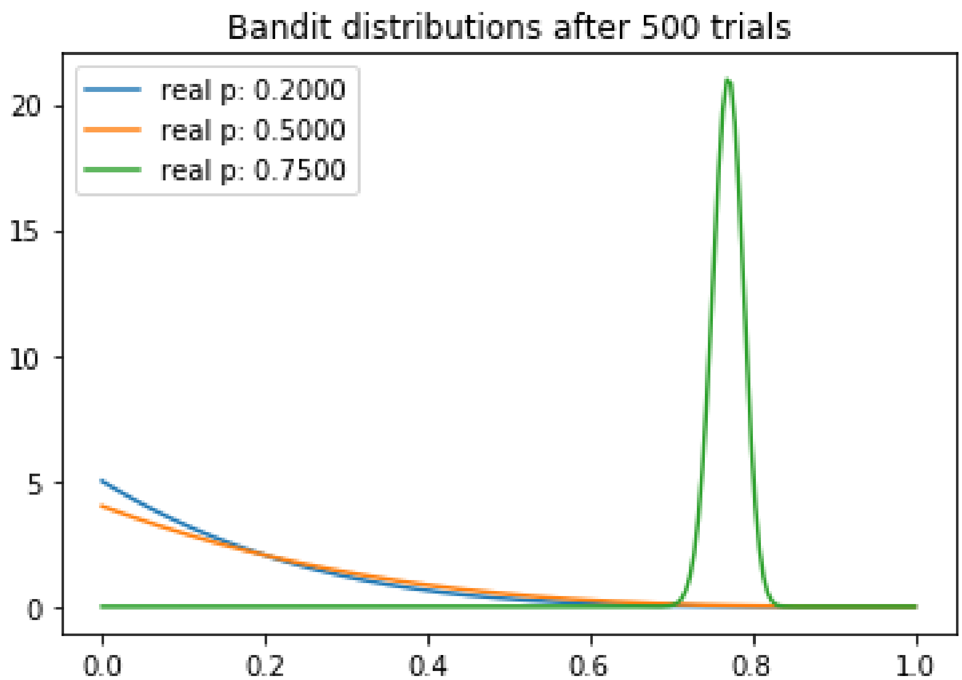

```
current samples: ['0.0555', '0.0838', '0.8012']
```



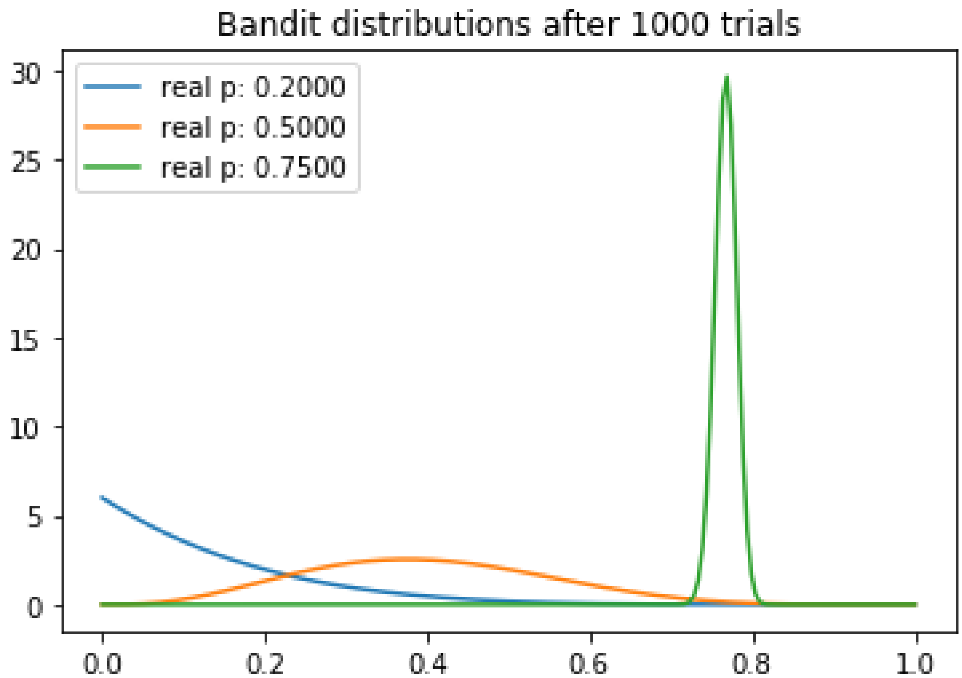
```
current samples: ['0.4525', '0.0752', '0.7951']
```



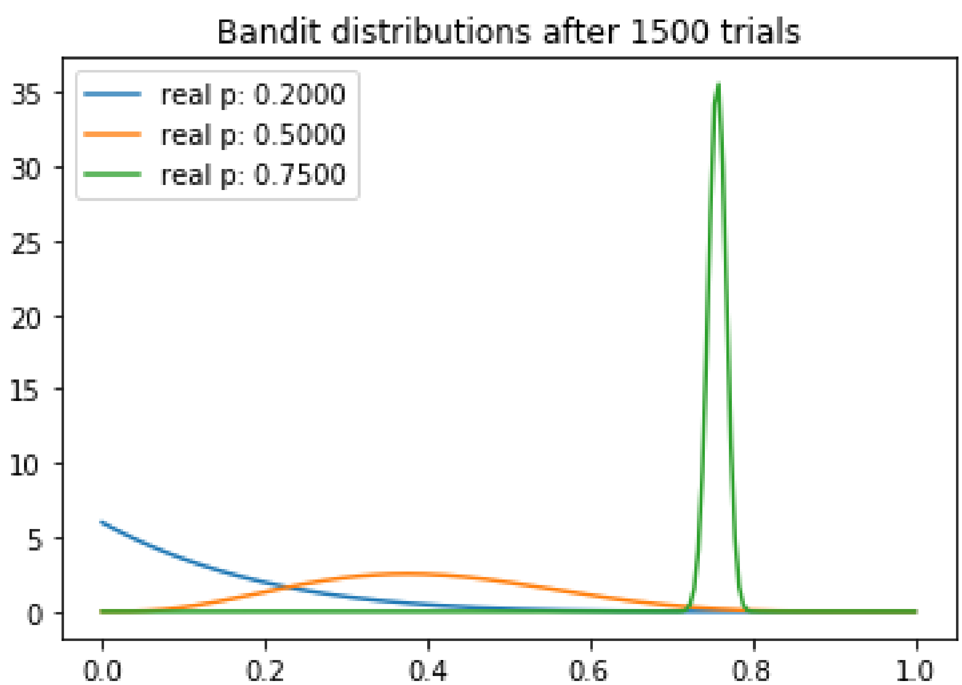

```
current samples: ['0.0559', '0.0381', '0.7579']
```



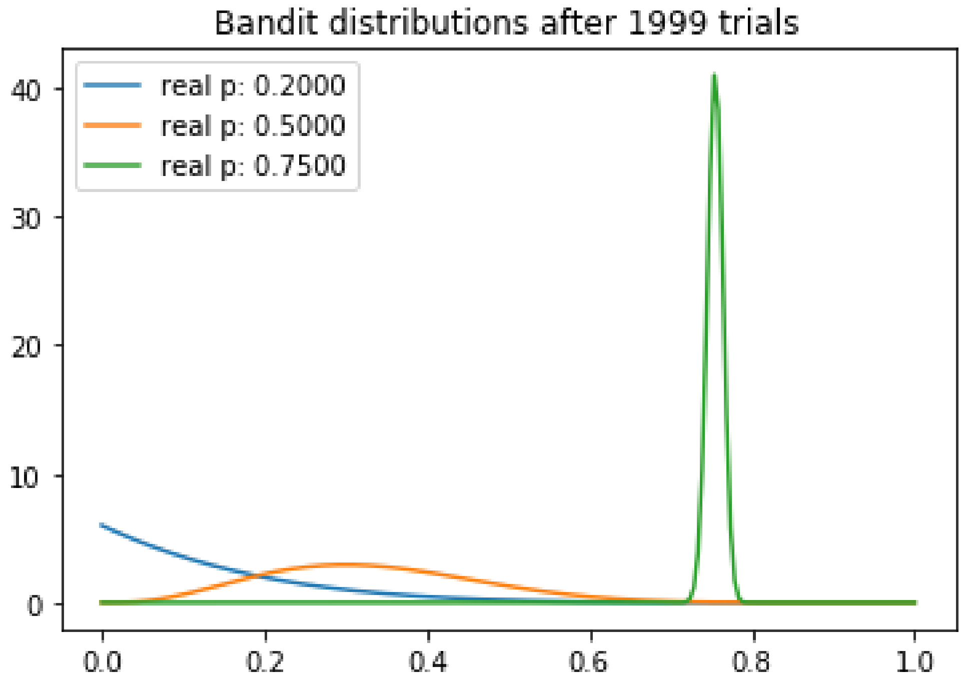
```
current samples: ['0.1711', '0.3887', '0.7536']
```




```
current samples: ['0.1885', '0.3112', '0.7767']
```



```
current samples: ['0.1058', '0.1914', '0.7520']
```



0.2 Example 2

[59]:

```
'''
This function takes as input three tuples for alpha,beta that specify␣
↪priorR,priorG,priorB
And returns R,G,B along with the maximum value sampled from these three␣
↪distributions.
We can sample from a beta distribution using scipy.
'''
def find_asset(priorR,priorG,priorB):
    red_rv = beta.rvs(priorR[0],priorR[1])
    green_rv = beta.rvs(priorG[0],priorG[1])
    blue_rv = beta.rvs(priorB[0],priorB[1])
    return assets[np.argmax([red_rv,green_rv,blue_rv])]


'''
This is a helper function that simulates the real world using the actual␣
↪probability value of the assets.
```

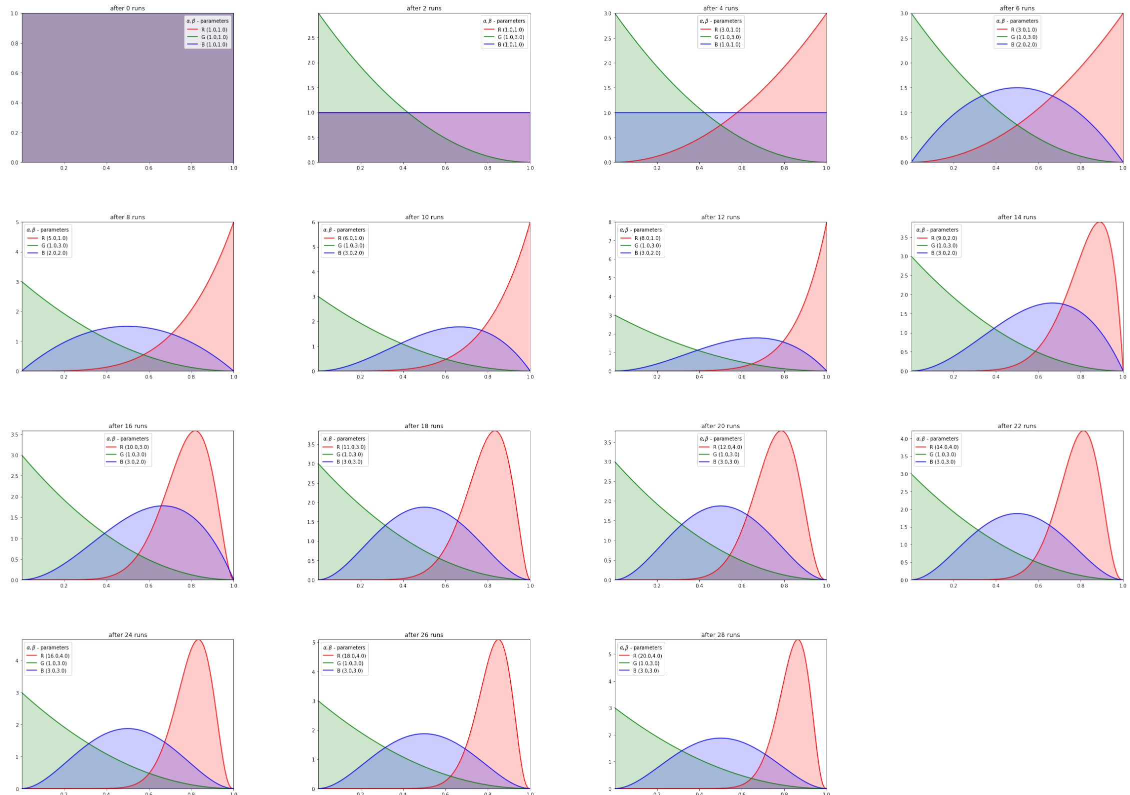
```
In real life we won't have this function and our user click input will be the␣
↪proxy for this function.
'''
def simulate_real_website(asset, real_probs_dict):
    #simulate a coin toss with probability. Asset clicked or not.
    if real_probs_dict[asset]> uniform.rvs(0,1):
        return 1
    else:
        return 0

'''
This function takes as input the selected asset and returns the posteriors for␣
↪the selected asset.
'''
def update_posterior(asset,priorR,priorG,priorB,outcome):
    if asset=='R':
        priorR=(priorR[0]+outcome,priorR[1]+1-outcome)
    elif asset=='G':
        priorG=(priorG[0]+outcome,priorG[1]+1-outcome)
    elif asset=='B':
        priorB=(priorB[0]+outcome,priorB[1]+1-outcome)
    return priorR,priorG,priorB


'''
This function runs the strategy once.
'''
def run_strategy_once(priorR,priorG,priorB):
    # 1. get the asset
    asset = find_asset(priorR,priorG,priorB)
    # 2. get the outcome from the website/users
    outcome = simulate_real_website(asset, real_probs_dict)
    # 3. update prior based on outcome
    priorR,priorG,priorB = update_posterior(asset,priorR,priorG,priorB,outcome)
    return asset,priorR,priorG,priorB

def plot_posteriors(priorR,priorG,priorB,ax=None,title=None):
    #fig = plt.figure(figsize=(12.5, 10))
    parameters = [priorR,priorG,priorB]
    x = np.linspace(0.001, 1, 150)
    for i, (_alpha, _beta) in enumerate(parameters):
        color = assets[i]
        y = beta.pdf(x, _alpha, _beta)
        lines = sns.lineplot(x, y, label="%s (%.1f,%.1f)" % (color, _alpha,␣
↪_beta), color = color,ax=ax)
        plt.fill_between(x, 0, y, alpha=0.2, color=color)
        if title:
```

```
            plt.title(title)
        plt.autoscale(tight=True)
    plt.legend(title=r"$\alpha, \beta$ - parameters")
    return plt
```

[60]:
```
real_probs_dict = {'R':0.8,'G':0.4,'B':0.3}
assets = ['R','G','B']

priorR,priorG,priorB = (1,1),(1,1),(1,1)

data = [("_",priorR,priorG,priorB)]

for i in range(50):
    asset,priorR,priorG,priorB = run_strategy_once(priorR,priorG,priorB)
    data.append((asset,priorR,priorG,priorB))
```

[61]:
```
fig = plt.figure(figsize=(40, 60))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
cnt=1
for i in range(0,30,2):
    ax = fig.add_subplot(8, 4, cnt)
    g = plot_posteriors(*data[i][1:],ax,"after "+str(i)+" runs")
    cnt+=1
plt.show()
```

```
/Users/jujohnson/anaconda3/envs/tf.latest/lib/python3.6/site-
packages/ipykernel_launcher.py:57: MatplotlibDeprecationWarning: Support for
uppercase single-letter colors is deprecated since Matplotlib 3.1 and will be
removed in 3.3; please use lowercase instead.
/Users/jujohnson/anaconda3/envs/tf.latest/lib/python3.6/site-
packages/IPython/core/pylabtools.py:128: MatplotlibDeprecationWarning: Support
for uppercase single-letter colors is deprecated since Matplotlib 3.1 and will
be removed in 3.3; please use lowercase instead.
  fig.canvas.print_figure(bytes_io, **kw)
```


after 0 runs
after 2 runs
after 4 runs
after 6 runs
after 8 runs
after 10 runs
after 12 runs
after 14 runs
after 16 runs
after 18 runs
after 20 runs
after 22 runs
after 24 runs
after 26 runs
after 28 runs

[ ]:

[ ]: