bayesian-bandit

December 29, 2015

1 Bayesian Bandit

Based on David's implementation of the Epsilon-Greedy algorithm and Probabilistic Programming and Bayesian Methods for Hackers, Chapter 6

In [11]:
```
# TODO: do this with a normal approximation rather than
# sampling since sampling is *incredibly* slow, see:
# Appendix of Chapter 4

import numpy as np
import pymc3 as pm
import scipy as sp
import matplotlib.pyplot as plt

# Display plots in Jupyter
%matplotlib inline

# For reproducibility
np.random.seed(123)

# The arm simulation model, where you can pull an arm and see how much
# money you get
class Arm:
    # Constructor: Arm(1,1)
    def __init__(self, mean, stdev):
        self.mean = mean
        self.stdev = stdev

    # Calling: Arm(1,1)()
    def __call__(self):
        return np.random.normal(self.mean, self.stdev)

    # Printing: print(Arm(1,1))
    def __repr__(self):
        return '<%s.%s object at %s with mean=%s, stdev=%s>'% (
            self.__class__.__module__,
            self.__class__.__name__,
            hex(id(self)), self.mean, self.stdev)

# Using the same model for each subsequent pull of a lever drastically
# speeds up the sampling
class ArmModel:
    def __init__(self):
```

```
        self.model = pm.Model()

        with self.model:
            # We don't know where the mean is
            self.mu = pm.Uniform('mu', 0, 200)

            # Probably a small standard deviation
            #   lambda = 0.01 doesn't start out with too small of a stdev
            #   testval makes it so we don't get an error
            self.sd = pm.Exponential('sd', 0.01, testval=5)

            # The distribution with our observations
            self.arm = pm.Normal('arm', mu=self.mu, sd=self.sd)

class Simulation:
    def __init__(self, bandits, mean_low, mean_high,
                 stdev_low, stdev_high, figsize=(15, 4)):

        assert bandits>0, "# of bandits must be > 0"

        # Create the arms for the simulation
        self.arms = np.zeros(bandits, dtype=object)

        for i in range(0,bandits):
            mean = np.random.randint(mean_low, mean_high)
            stdev = np.random.randint(stdev_low, stdev_high)
            self.arms[i] = Arm(mean, stdev)

        # For consistently-sized plots
        self.figsize = figsize

    # Example: plotArms() or plotArms(bandits=[0,1,5,10])
    def plotArms(self, bandits=None, stdevs=5, bins=75, points=10000):
        plt.figure(figsize=self.figsize)
        plt.suptitle('Bandit Monetary Distributions')
        plt.xlabel('Monetary value on pulling lever')
        plt.ylabel('Probability of this value')
        plt.grid()

        # If list of bandits to display not specified, just use all of them
        if bandits == None:
            bandits = range(0, len(self.arms))

        for i in range(0,len(bandits)):
            samples = [self.arms[bandits[i]]() for j in range(0, points)]
            x = np.linspace(self.arms[bandits[i]].mean
                            - stdevs*self.arms[bandits[i]].stdev,
                            self.arms[bandits[i]].mean
                            + stdevs*self.arms[bandits[i]].stdev)
            plt.hist(samples, bins=bins, normed=True, histtype="stepfilled",
                     alpha=0.2, label='Bandit #'+str(bandits[i]))

        plt.legend()
```

```
    # Randomly pull levers and see how well we do
    def plotRand(self, trials=50, pullsPerTrial=300):
        plt.figure(figsize=self.figsize)
        plt.suptitle('Online "Learning" Algorithm: Random (a baseline)')
        plt.xlabel('Iteration')
        plt.ylabel('Reward')
        plt.grid()

        rewards = np.zeros((trials, pullsPerTrial))

        for trial in range(0, trials):
            for pull in range(0, pullsPerTrial):
                # Randomly select an arm to look at
                selected_arm = np.random.randint(0, len(self.arms))

                # Pull the arm and measure the reward
                rewards[trial, pull] = self.arms[selected_arm]()

        # Average the first pull of each trial, the second pull, etc.
        pullRewards = rewards.mean(axis=0)

        plt.plot(pullRewards)

    # Use the epsilon-greedy algorithm for determining when to exploit vs. explore
    def plotEpsGreedy(self, epsilons=[0.25,0.5,0.75,0.9], trials=500, pullsPerTrial=300):
        plt.figure(figsize=self.figsize)
        plt.suptitle('Online Learning Algorithm: Epsilon-Greedy')
        plt.xlabel('Iteration')
        plt.ylabel('Reward')
        plt.grid()

        for epsilon in epsilons:
            rewards = np.zeros((trials, pullsPerTrial))

            for trial in range(0, trials):
                # Which lever did we pull each time?
                selected_arm = np.zeros(pullsPerTrial, dtype=int)

                for pull in range(0, pullsPerTrial):
                    curiosity = np.random.random()

                    # Be greedy
                    if curiosity >= epsilon:
                        # Pull the lever with the most past reward
                        selected_arm[pull] = selected_arm[rewards[trial].argmax()]

                    # Explore, choose a random lever
                    else:
                        # Randomly select an arm to look at
                        selected_arm[pull] = np.random.randint(0, len(self.arms))

                    # Pull the arm and measure the reward
                    rewards[trial, pull] = self.arms[selected_arm[pull]]()
```

```
                #if trial==0:
                #    print(curiosity>=epsilon, selected_arm[pull],'of',
                #          len(self.arms),'reward =',rewards[trial,pull])

            # Average the first pull of each trial, the second pull, etc.
            pullRewards = rewards.mean(axis=0)

            plt.plot(pullRewards, label='Epsilon='+str(epsilon))

        plt.legend()

    # The Bayesian Bandit solution
    def plotBayes(self, trials=10, pullsPerTrial=300):
        plt.figure(figsize=self.figsize)
        plt.suptitle('Online Learning Algorithm: Bayesian')
        plt.xlabel('Iteration')
        plt.ylabel('Reward')
        plt.grid()

        rewards = np.zeros((trials, pullsPerTrial))

        for trial in range(0, trials):
            # Which lever did we pull each time?
            selected_arm = np.empty(pullsPerTrial, dtype=int)
            selected_arm.fill(-1); # can't be a valid arm, e.g. can't be zero

            # Start with a uniform prior for each lever:
            #    (mu, stdev, 95% least plausible value)
            arm_beliefs = np.zeros([len(self.arms), 3], dtype=float)
            models = np.zeros(len(self.arms), dtype=object)

            for lever in range(0, len(self.arms)):
                arm_beliefs[lever] = (np.nan, np.nan, np.nan)
                models[lever] = ArmModel()

            # Start pulling levers!
            for pull in range(0, pullsPerTrial):
                nans = [np.isnan(mu) or np.isnan(sd) or np.isnan(lpv)
                        for mu, sd, lpv in arm_beliefs]

                # Start off by pulling each lever at least once so we have some data
                # to operate on
                if True in nans:
                    lever = nans.index(True)

                else:
                    # Select by highest 95% least plausible value (see Chapter 4)
                    lpv_min = arm_beliefs[:,2].min()
                    sq_diffs = np.array([(lpv - lpv_min)**2
                                         for mu, sd, lpv in arm_beliefs])
                    diff_sum = sq_diffs.sum()
                    p_by_lpv = [s / diff_sum for s in sq_diffs]

                    # Sample
```

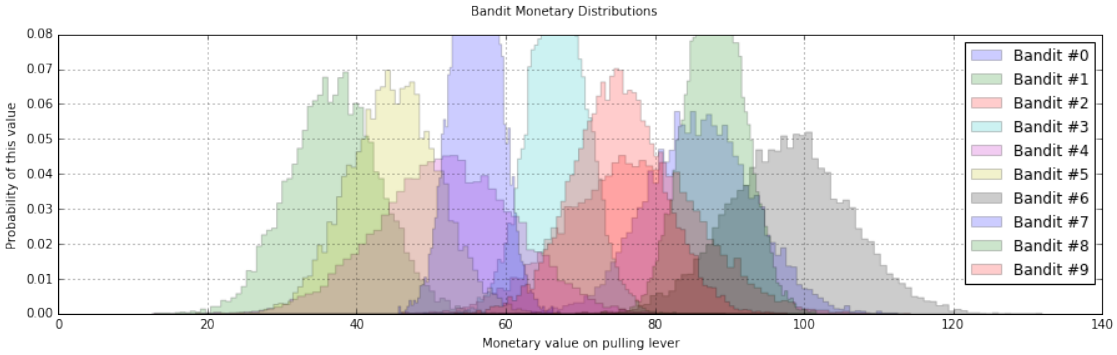
```
            lever  = np.random.choice(len(self.arms), 1,
                                       replace=False, p=p_by_lpv)[0]

        reward = self.arms[lever]()

        # Save results (must save before loading from rewards 2D array)
        selected_arm[pull] = lever
        rewards[trial, pull] = reward

        # What money have we gotten when we pulled this lever this time
        # and before?
        lever_observations = [rewards[trial, j]
                              for j, arm in enumerate(selected_arm)
                              if arm == lever]

        # Update information about this lever
        with models[lever].model:
            # Don't update on every pull, too slow
            if (len(lever_observations)-1)%5 == 0:
                # Create normal distribution with observed values
                models[lever].arm = pm.Normal('arm', mu=models[lever].mu,
                                              sd=models[lever].sd,
                                              observed=lever_observations)

                # Sample this to find the posterior
                step = pm.Metropolis(vars=[models[lever].mu,
                                           models[lever].sd,
                                           models[lever].arm])
                trace = pm.sample(100, step=step, progressbar=False)

                # Mean
                arm_beliefs[lever][0] = trace[-1]['mu']
                # Stdev
                arm_beliefs[lever][1] = trace[-1]['sd']
                # 95% least-plausible value
                arm_beliefs[lever][2] = np.sort(trace['mu'])[int(
                        0.05 * len(trace['mu']))]

            # Debugging at the end
            if pull == pullsPerTrial-1:
                print("Beliefs:")
                for i, (mu, sd, lpv) in enumerate(arm_beliefs):
                    count = len([0 for j, arm in enumerate(selected_arm)
                                 if arm == i])
                    print("Count:", count, "Estimate: mu =", mu,
                          "sd =", sd, "lpv =", lpv,
                          "Actual: mu =", self.arms[i].mean,
                          "sd =", self.arms[i].stdev)

# Average the first pull of each trial, the second pull, etc.
pullRewards = rewards.mean(axis=0)
plt.plot(pullRewards, label='Bayes')
plt.legend()
```

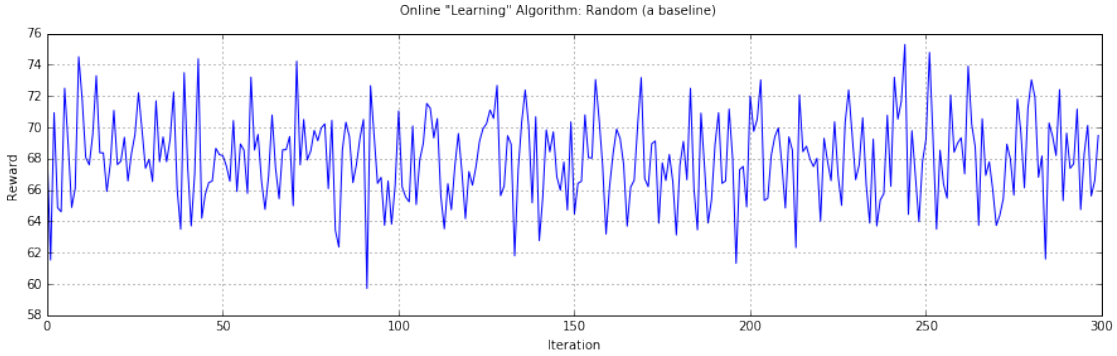
```
        # bandits - How many bandits (or machines, or arms we'll pull) do we want?
        # iterations - How many times will we pull an arm?
        def MultiArmedBandit(bandits):
            # Create all the arms for simulation
            sim = Simulation(bandits=bandits, mean_low=20, mean_high=100,
                             stdev_low=3, stdev_high=10)

            # Plot 10 of the arms for illustrative purposes
            sim.plotArms(range(0,bandits,int(bandits/10)))

            # For reference, just randomly choose arms
            sim.plotRand()

            # The epsilon-greedy algorithm
            sim.plotEpsGreedy()

            # Bayesian Bandit
            sim.plotBayes()

        MultiArmedBandit(bandits=10)
```

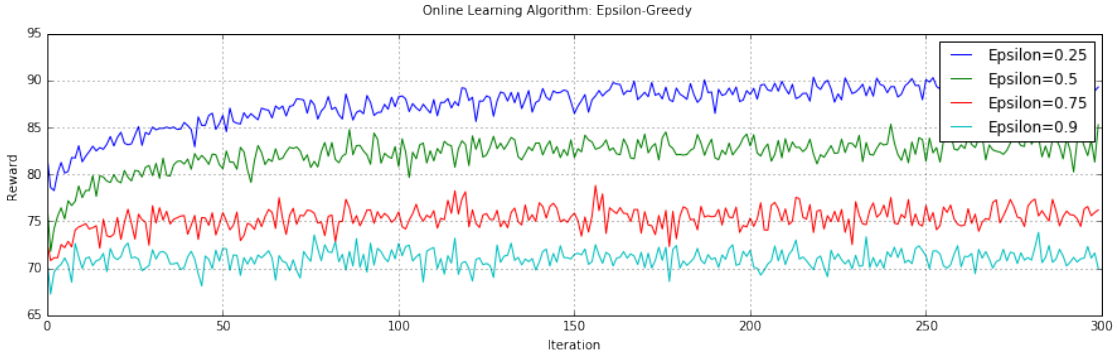
```
INFO (theano.gof.compilelock): Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-w
INFO:theano.gof.compilelock:Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-with

Beliefs:
Count: 12 Estimate: mu = 88.4943869665 sd = 27.3644675206 lpv = 55.9344013566 Actual: mu = 86 sd = 7
Count: 9 Estimate: mu = 5.17191494039 sd = 115.859885946 lpv = 16.2390137927 Actual: mu = 37 sd = 6
Count: 28 Estimate: mu = 115.680548545 sd = 108.024929142 lpv = 56.4572212098 Actual: mu = 77 sd = 9
Count: 17 Estimate: mu = 82.5206141692 sd = 34.6592539541 lpv = 16.1784736061 Actual: mu = 67 sd = 4
Count: 31 Estimate: mu = 78.3680050328 sd = 58.1656811133 lpv = 40.8584968277 Actual: mu = 52 sd = 9
Count: 1 Estimate: mu = 71.1851280678 sd = 23.3671851295 lpv = 2.63126959451 Actual: mu = 45 sd = 6
Count: 30 Estimate: mu = 83.9191178114 sd = 13.9886847139 lpv = 49.8914982162 Actual: mu = 98 sd = 8
Count: 23 Estimate: mu = 131.582096701 sd = 117.451338822 lpv = 16.0044187686 Actual: mu = 56 sd = 3
Count: 78 Estimate: mu = 99.336412045 sd = 6.38790901782 lpv = 74.6003102256 Actual: mu = 88 sd = 4
Count: 71 Estimate: mu = 120.45807967 sd = 53.4569084175 lpv = 7.83043363546 Actual: mu = 75 sd = 6
Beliefs:

INFO (theano.gof.compilelock): Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-w
INFO:theano.gof.compilelock:Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-with

Count: 52 Estimate: mu = 105.544144845 sd = 8.0341844925 lpv = 52.7002293887 Actual: mu = 86 sd = 7
Count: 3 Estimate: mu = 50.8571567338 sd = 89.6565428986 lpv = 10.9979407193 Actual: mu = 37 sd = 6
Count: 41 Estimate: mu = 122.186839919 sd = 115.62310382 lpv = 10.6430918206 Actual: mu = 77 sd = 9
Count: 10 Estimate: mu = 61.7230808675 sd = 57.5270865316 lpv = 15.6822577147 Actual: mu = 67 sd = 4
Count: 10 Estimate: mu = 27.1210942828 sd = 50.9451390131 lpv = 26.0241435893 Actual: mu = 52 sd = 9
Count: 15 Estimate: mu = 76.6946262645 sd = 53.1876446582 lpv = 16.6009640445 Actual: mu = 45 sd = 6
Count: 83 Estimate: mu = 104.437524757 sd = 6.90102005628 lpv = 91.0032371274 Actual: mu = 98 sd = 8
Count: 16 Estimate: mu = 109.572525202 sd = 46.7530831196 lpv = 5.02959917264 Actual: mu = 56 sd = 3
Count: 53 Estimate: mu = 72.8404320676 sd = 31.6862070075 lpv = 65.0554580835 Actual: mu = 88 sd = 4
Count: 17 Estimate: mu = 115.320200759 sd = 17.8495568829 lpv = 9.14866152949 Actual: mu = 75 sd = 6
Beliefs:
Count: 56 Estimate: mu = 109.533564576 sd = 44.4350612762 lpv = 0.401702837851 Actual: mu = 86 sd = 7
Count: 21 Estimate: mu = 24.4053938099 sd = 220.169653487 lpv = 6.48511045912 Actual: mu = 37 sd = 6
Count: 59 Estimate: mu = 81.8792987376 sd = 24.8573143983 lpv = 23.150909003 Actual: mu = 77 sd = 9
Count: 4 Estimate: mu = 9.75310962533 sd = 93.4524097385 lpv = 12.0195693755 Actual: mu = 67 sd = 4
```

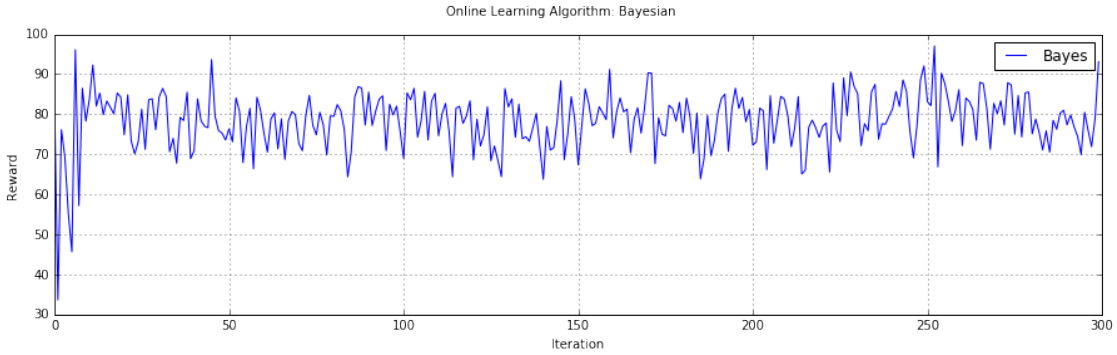
```
Count: 30 Estimate: mu = 83.9758451183 sd = 9.33013337255 lpv = 42.6003709344 Actual: mu = 52 sd = 9
Count: 6 Estimate: mu = 37.155626678 sd = 80.7741098987 lpv = 5.16541287588 Actual: mu = 45 sd = 6
Count: 90 Estimate: mu = 89.4946176891 sd = 10.6549643522 lpv = 66.7592026238 Actual: mu = 98 sd = 8
Count: 25 Estimate: mu = 45.85514929 sd = 184.680692104 lpv = 31.3344500975 Actual: mu = 56 sd = 3
Count: 3 Estimate: mu = 10.1629823326 sd = 124.58109963 lpv = 7.52666450387 Actual: mu = 88 sd = 4
Count: 6 Estimate: mu = 50.6765173903 sd = 90.9728683136 lpv = 3.64361225974 Actual: mu = 75 sd = 6
Beliefs:

INFO (theano.gof.compilelock): Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-w
INFO:theano.gof.compilelock:Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-with

Count: 12 Estimate: mu = 91.6109546523 sd = 7.66331557702 lpv = 15.5927176387 Actual: mu = 86 sd = 7
Count: 24 Estimate: mu = 63.0855297011 sd = 31.2796268227 lpv = 26.8700943313 Actual: mu = 37 sd = 6
Count: 36 Estimate: mu = 199.501471296 sd = 183.57193102 lpv = 13.6110881994 Actual: mu = 77 sd = 9
Count: 14 Estimate: mu = 36.5371237381 sd = 100.277595239 lpv = 15.8140053522 Actual: mu = 67 sd = 4
Count: 12 Estimate: mu = 20.7704397075 sd = 50.8029924291 lpv = 16.8865388703 Actual: mu = 52 sd = 9
Count: 6 Estimate: mu = 56.8570070756 sd = 26.62877891 lpv = 12.653611212 Actual: mu = 45 sd = 6
Count: 16 Estimate: mu = 60.4853465411 sd = 45.2359048246 lpv = 10.9426021222 Actual: mu = 98 sd = 8
Count: 17 Estimate: mu = 131.809210159 sd = 118.543627349 lpv = 13.8201338036 Actual: mu = 56 sd = 3
Count: 112 Estimate: mu = 61.9585927543 sd = 59.2727325891 lpv = 61.9585927543 Actual: mu = 88 sd = 4
Count: 51 Estimate: mu = 93.3610795465 sd = 43.4649142503 lpv = 10.036385322 Actual: mu = 75 sd = 6
Beliefs:
Count: 18 Estimate: mu = 93.5543592562 sd = 31.4373595257 lpv = 7.29104557238 Actual: mu = 86 sd = 7
Count: 3 Estimate: mu = 8.62591105079 sd = 73.6597236196 lpv = 11.6594305053 Actual: mu = 37 sd = 6
Count: 51 Estimate: mu = 104.841095807 sd = 27.0609028563 lpv = 3.85588811205 Actual: mu = 77 sd = 9
Count: 13 Estimate: mu = 68.2036439509 sd = 49.1743103481 lpv = 11.1003366366 Actual: mu = 67 sd = 4
Count: 2 Estimate: mu = 25.0627110648 sd = 141.929747691 lpv = 11.8360700221 Actual: mu = 52 sd = 9
Count: 23 Estimate: mu = 21.5524417998 sd = 162.751292248 lpv = 11.9115259959 Actual: mu = 45 sd = 6
Count: 41 Estimate: mu = 115.675162789 sd = 17.8711226274 lpv = 3.83095544419 Actual: mu = 98 sd = 8
Count: 48 Estimate: mu = 42.3733000167 sd = 81.2013999052 lpv = 30.4983720424 Actual: mu = 56 sd = 3
Count: 78 Estimate: mu = 103.073011131 sd = 11.2761687532 lpv = 67.211351812 Actual: mu = 88 sd = 4
Count: 23 Estimate: mu = 148.236304984 sd = 50.0423007332 lpv = 13.0338886169 Actual: mu = 75 sd = 6
Beliefs:

INFO (theano.gof.compilelock): Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-w
INFO:theano.gof.compilelock:Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-with

Count: 1 Estimate: mu = 92.3241855812 sd = 8.03332874573 lpv = 12.6908100331 Actual: mu = 86 sd = 7
Count: 11 Estimate: mu = 63.9609830052 sd = 49.4133210215 lpv = 13.9403504298 Actual: mu = 37 sd = 6
Count: 11 Estimate: mu = 80.5545426464 sd = 31.2721644653 lpv = 11.9915318456 Actual: mu = 77 sd = 9
Count: 2 Estimate: mu = 16.1630916337 sd = 89.9212006222 lpv = 15.9249063975 Actual: mu = 67 sd = 4
Count: 2 Estimate: mu = 169.963373578 sd = 80.829857846 lpv = 15.4521649885 Actual: mu = 52 sd = 9
Count: 6 Estimate: mu = 65.0296227818 sd = 119.059234304 lpv = 12.5254766717 Actual: mu = 45 sd = 6
Count: 158 Estimate: mu = 95.2961788997 sd = 3.91302333142 lpv = 21.5260008134 Actual: mu = 98 sd = 8
Count: 31 Estimate: mu = 58.1340925175 sd = 19.1126962403 lpv = 10.8290304926 Actual: mu = 56 sd = 3
Count: 67 Estimate: mu = 104.409501282 sd = 14.5177030456 lpv = 46.6624353176 Actual: mu = 88 sd = 4
Count: 11 Estimate: mu = 44.5436517678 sd = 60.4057686852 lpv = 35.0675874083 Actual: mu = 75 sd = 6
Beliefs:

INFO (theano.gof.compilelock): Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-w
INFO:theano.gof.compilelock:Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-with

Count: 28 Estimate: mu = 72.6692469802 sd = 23.4093221409 lpv = 6.09960347596 Actual: mu = 86 sd = 7
Count: 30 Estimate: mu = 78.0580780987 sd = 162.121994346 lpv = 23.091553801 Actual: mu = 37 sd = 6
Count: 6 Estimate: mu = 20.763108445 sd = 104.520646778 lpv = 0.922271110226 Actual: mu = 77 sd = 9
```

```
Count: 59 Estimate: mu = 105.772662136 sd = 33.6142644108 lpv = 44.1112111008 Actual: mu = 67 sd = 4
Count: 27 Estimate: mu = 66.0228339328 sd = 21.7581750305 lpv = 33.3134768751 Actual: mu = 52 sd = 9
Count: 11 Estimate: mu = 90.9084591547 sd = 84.903768982 lpv = 4.94512064995 Actual: mu = 45 sd = 6
Count: 78 Estimate: mu = 103.579932962 sd = 2.74983050714 lpv = 67.7785215015 Actual: mu = 98 sd = 8
Count: 31 Estimate: mu = 51.0221759162 sd = 43.1951397327 lpv = 13.2789959446 Actual: mu = 56 sd = 3
Count: 1 Estimate: mu = 67.7049072121 sd = 24.5443982313 lpv = 9.11046245012 Actual: mu = 88 sd = 4
Count: 29 Estimate: mu = 107.079621309 sd = 70.7784786496 lpv = 19.5512465568 Actual: mu = 75 sd = 6
Beliefs:

INFO (theano.gof.compilelock): Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-w
INFO:theano.gof.compilelock:Refreshing lock /home/garrett/.theano/compiledir_Linux-4.2--ARCH-x86_64-with

Count: 16 Estimate: mu = 91.3590999597 sd = 11.3303580624 lpv = 55.2244964104 Actual: mu = 86 sd = 7
Count: 6 Estimate: mu = 77.0442187985 sd = 33.035725174 lpv = 1.32039662281 Actual: mu = 37 sd = 6
Count: 28 Estimate: mu = 49.1221406024 sd = 36.1566841283 lpv = 11.8216415251 Actual: mu = 77 sd = 9
Count: 16 Estimate: mu = 118.056812269 sd = 38.210575618 lpv = 7.70697777786 Actual: mu = 67 sd = 4
Count: 6 Estimate: mu = 57.5303908954 sd = 49.8355022807 lpv = 5.20371059948 Actual: mu = 52 sd = 9
Count: 6 Estimate: mu = 8.13253641003 sd = 44.1067060419 lpv = 6.17202994583 Actual: mu = 45 sd = 6
Count: 110 Estimate: mu = 98.8691699653 sd = 4.41388243676 lpv = 23.2538960231 Actual: mu = 98 sd = 8
Count: 3 Estimate: mu = 116.796501253 sd = 48.42618187 lpv = 13.8285105602 Actual: mu = 56 sd = 3
Count: 59 Estimate: mu = 83.8216838398 sd = 18.6312795873 lpv = 24.3414602208 Actual: mu = 88 sd = 4
Count: 50 Estimate: mu = 17.499354839 sd = 84.3098815405 lpv = 30.0494371939 Actual: mu = 75 sd = 6
Beliefs:
Count: 37 Estimate: mu = 94.2016277311 sd = 10.432996147 lpv = 14.5920695432 Actual: mu = 86 sd = 7
Count: 7 Estimate: mu = 15.7298651124 sd = 51.1095740919 lpv = 17.9145440236 Actual: mu = 37 sd = 6
Count: 7 Estimate: mu = 98.7534689762 sd = 46.1950836977 lpv = 16.8597353565 Actual: mu = 77 sd = 9
Count: 11 Estimate: mu = 1.62072798145 sd = 79.7983314966 lpv = 2.42720644622 Actual: mu = 67 sd = 4
Count: 26 Estimate: mu = 106.723652798 sd = 68.2625406937 lpv = 43.7565194102 Actual: mu = 52 sd = 9
Count: 3 Estimate: mu = 30.8411106653 sd = 117.737495828 lpv = 13.5902359387 Actual: mu = 45 sd = 6
Count: 68 Estimate: mu = 35.1557944491 sd = 30.8746885506 lpv = 42.2887607264 Actual: mu = 98 sd = 8
Count: 11 Estimate: mu = 93.8664036598 sd = 53.2978289948 lpv = 2.31266602646 Actual: mu = 56 sd = 3
Count: 32 Estimate: mu = 90.2742385758 sd = 21.2384939871 lpv = 17.2094996803 Actual: mu = 88 sd = 4
Count: 98 Estimate: mu = 147.616845712 sd = 143.61296056 lpv = 74.2602870331 Actual: mu = 75 sd = 6
Beliefs:
Count: 9 Estimate: mu = 29.5450953026 sd = 45.7239007735 lpv = 21.6236197202 Actual: mu = 86 sd = 7
Count: 18 Estimate: mu = 27.5128693707 sd = 51.9049105024 lpv = 14.2209265408 Actual: mu = 37 sd = 6
Count: 1 Estimate: mu = 108.146482496 sd = 13.3215796599 lpv = 7.58247313185 Actual: mu = 77 sd = 9
Count: 57 Estimate: mu = 103.295687914 sd = 44.7558835456 lpv = 35.1632548443 Actual: mu = 67 sd = 4
Count: 20 Estimate: mu = 46.7491892369 sd = 82.6618237163 lpv = 39.141781197 Actual: mu = 52 sd = 9
Count: 7 Estimate: mu = 26.1804177663 sd = 112.202670796 lpv = 10.6216730696 Actual: mu = 45 sd = 6
Count: 104 Estimate: mu = 99.7426286136 sd = 2.41918215066 lpv = 34.7393867116 Actual: mu = 98 sd = 8
Count: 11 Estimate: mu = 68.7087473043 sd = 35.6447827264 lpv = 1.07088071852 Actual: mu = 56 sd = 3
Count: 42 Estimate: mu = 51.0183400255 sd = 75.8646409023 lpv = 32.0979430461 Actual: mu = 88 sd = 4
Count: 31 Estimate: mu = 7.27070332311 sd = 53.4106892541 lpv = 27.5004942545 Actual: mu = 75 sd = 6
```


Bandit Monetary Distributions
Probability of this value
Monetary value on pulling lever
Bandit #0
Bandit #1
Bandit #2
Bandit #3
Bandit #4
Bandit #5
Bandit #6
Bandit #7
Bandit #8
Bandit #9


Online "Learning" Algorithm: Random (a baseline)
Reward
Iteration


Online Learning Algorithm: Epsilon-Greedy
Reward
Iteration
Epsilon=0.25
Epsilon=0.5
Epsilon=0.75
Epsilon=0.9


Online Learning Algorithm: Bayesian
Bayes
Reward
Iteration
100
90
80
70
60
50
40
30
0
50
100
150
200
250
300