APPENDIX

1.2 (b)

In [1]:
```
import numpy as py
```

In [2]:
```
def calc_z(p_a, p_b, p_c, sunny):
    x_a = ((p_a) ** sunny) * (1 - p_a) ** (10 - sunny)
    x_b = ((p_b) ** sunny) * (1 - p_b) ** (10 - sunny)
    x_c = ((p_c) ** sunny) * (1 - p_c) ** (10 - sunny)
    sum_abc = x_a + x_b + x_c
    return (x_a/sum_abc, x_b/sum_abc, x_c/sum_abc)
```

In [14]:
```
calc_z(0.75,0.25,0.55,8)
```

Out[14]:
```
(0.7859387940650213, 0.0010781053416529785, 0.21298310059332584)
```

In [15]:
```
calc_z(0.75,0.25,0.55,3)
```

Out[15]:
```
(0.009421150914142334, 0.7631132240455292, 0.2274656250403285)
```

In [16]:
```
calc_z(0.75,0.25,0.55,5)
```

Out[16]:
```
(0.1664595841967641, 0.1664595841967641, 0.6670808316064718)
```

In [17]:
```
calc_z(0.75,0.25,0.55,2)
```

Out[17]:
```
(0.0012670049499898517, 0.9236466085426018, 0.07508638650740826)
```

In [18]:
```
calc_z(0.75,0.25,0.55,6)
```

Out[18]:
```
(0.36446047786379454, 0.04049560865153273, 0.5950439134846727)
```

In [19]:
```
calc_z(0.75,0.25,0.55,7)
```

Out[19]:
```
(0.5961224206794469, 0.007359536057770949, 0.39651804326278217)
```

3 (b)

```
for iter in range(0,num_iter):
    """
        E-Step
        In the first step, we find the expected log-likelihood of the data which is equivalent to:
        finding cluster assignments for each point probabilistically
        In this section, you will calculate the values of zk(n,k) for all n and k according to current valu
    """
    # TODO

    for i in range(0, N):
        for j in range(0, num_K):
            zk[i, j] = np.log(pk[j]) + calc_logpdf(trainX[i], mu[j], si2)
        zk[i] = zk[i] - logsumexp(zk[i])


    """
        M-step
        Compute the GMM parameters from the expressions which you have in your writeup
    """

    # Estimate new value of pk
    # TODO
    for j in range(0, num_K): pk[j] = np.exp(logsumexp(zk[:,j])) / N

    # Estimate new value for means
    # TODO
    for j in range(0, num_K): mu[j] = np.matmul([np.exp(zk[:,j])], trainX) / np.exp(logsumexp(zk[:,j]))

    # Estimate new value for sigma^2
    # TODO
    transform = np.zeros([N, num_K])
    for i in range(0, N):
        for j in range(0, num_K):
            transform[i, j] = (trainX[i] - mu[j]).dot(trainX[i] - mu[j])
    si2 = np.exp(logsumexp(zk, b = transform)) / (N * D)

# Computing the expected likelihood of data for the optimal parameters computed
# TODO
for i in range(0, N):
    for j in range(0, num_K):
        zk[i, j] = np.log(pk[j]) + calc_logpdf(trainX[i], mu[j], si2)

log_like = np.sum(logsumexp(zk, 1))

# Compute the BIC for the current cluster
# TODO
BIC = (D + 1) * num_K * np.log(N) - 2 * log_like
```

3 (c) (d)

```
print("We'll try different numbers of clusters with GMM, using multiple runs for each to identify the 'best' results")
trainX = get_data()
num_K = range(2, 9) # List of cluster sizes
BIC_K = np.zeros(len(num_K))
means = {} # Dictionary mapping cluster size to corresponding matrix of means
cluster_proportions = {} # Dictionary mapping cluster size to corresponding mixture proportions vector
z_K = {}
sigma2 = {} # Dictionary mapping cluster size to the learned variance value
for idx in range(len(num_K)):
    # Running
    k = num_K[idx]
    print("%d clusters..." % k)
    bestBIC = float("inf")
    for i in range(1, 11):
        # TODO: Run gmm function 10 times and get the best
        # set of parameters for this particular value of k
        log_like = gmm(trainX, k)[4]
        if log_like < bestBIC: bestBIC = log_like
    BIC_K[idx] = bestBIC

# TODO: Part d: Make a plot to show BIC as function of clusters K
plt.plot(num_K, BIC_K)
plt.xlabel("num_cluster")
plt.ylabel("BestBic_Value")
plt.show()
```