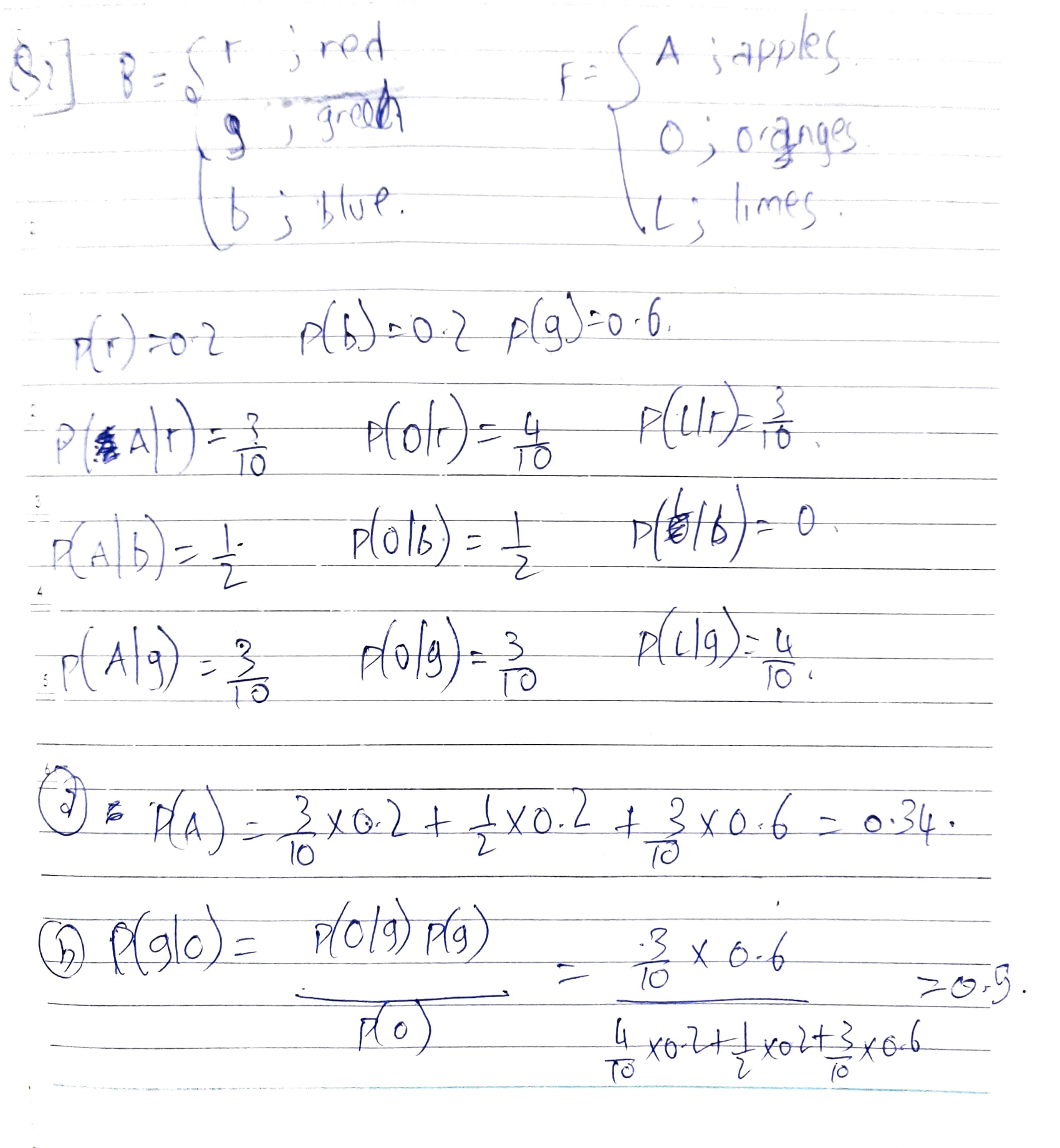
Machine Leaning Submission 1

Team: Jan Kruska, Sagar Kalburgi, Pawan, Kashyap Melur Mahesh

Q2)



Q5)

a)

pos = np.arange(-5, 5.0, 0.1)

factor = 1 / (np.power(2 \* np.pi, 1 / 2) \* h)

val = [factor \* np.mean(np.exp((-1/(2\*h\*\*2)\*(samples-i)\*\*2))) for i in pos]

# Compute the number of samples created

return np.stack([pos,val]).transpose()

b)

pos = np.arange(-5, 5.0, 0.1)

n = samples.shape[0]

val = [k/(2\*n\*(np.linalg.norm(np.sort(np.abs(samples-i))[k-1]))) for i in pos]

# Compute the number of samples created

return np.stack([pos,val]).transpose()

Q6)

a)

return np.sum([np.log(gaussian\_mix(x, means, weights, covariances)) for x in X])

def gaussian\_mix(x, means, weights, covariances):

return np.sum([weights[k]\*gaussian(x,means[k],covariances[:,:,k]) for k in range(len(weights))])

def gaussian(x,mu,sigma):

d = sigma.shape[0]

factor = 1 / np.sqrt((np.power(2 \* np.pi, d) \* np.linalg.det(sigma)))

return factor \* np.exp((-1 / 2 \* np.dot((x-mu),np.dot(np.linalg.inv(sigma),(x-mu)))))

b)

n = X.shape[0]

k = len(weights)

gamma = np.empty([n,k])

for i in range(n):

for j in range(k):

gamma[i,j] = (weights[j]\*gaussian(X[i,:],means[j],covariances[:,:,j]))/gaussian\_mix(X[i,:],means,weights,covariances)

return [getLogLikelihood(means,weights,covariances,X), gamma]

c)

n\_k = np.sum(gamma,0)

covariances = np.empty([X.shape[1], X.shape[1], gamma.shape[1]])

means = np.array([np.dot(gamma[:,k],X)/n\_k[k] for k in range(gamma.shape[1])])

for k in range(gamma.shape[1]):

# elementwise product of (DxN)(N) => DxN dot-product NxD => DxD

covariances[:, :, k] = np.dot(np.transpose(X - means[k]) \* gamma[:, k], X - means[k]) / n\_k[k]

# Regularize to avoid numerical anomalies

#covariances[:, :, k] = regularize\_cov(covariances[:, :, k], 0.0001)

weights = n\_k/X.shape[0]

logLikelihood = getLogLikelihood(means, weights, covariances, X)

return weights, means, covariances, logLikelihood

d)

regularized\_cov = covariance + epsilon\*np.identity(len(covariance))

return regularized\_cov

e)

n\_dim = data.shape[1]

weights = np.ones(K) / K

kmeans = KMeans(n\_clusters=K,n\_init=10).fit(data)

cluster\_idx = kmeans.labels\_

means = kmeans.cluster\_centers\_

covariances = np.empty([n\_dim,n\_dim,K])

for j in range(K):

data\_cluster = data[cluster\_idx==j]

min\_dist = np.inf

for i in range(K):

#compute sum of distances

dist = np.mean(euclidean\_distances(data\_cluster,[means[j]],squared=True))

if dist < min\_dist:

min\_dist = dist

covariances[:,:,j] = np.eye(n\_dim) \* min\_dist

for i in range(n\_iters):

\_, gamma = EStep(means, covariances, weights, data)

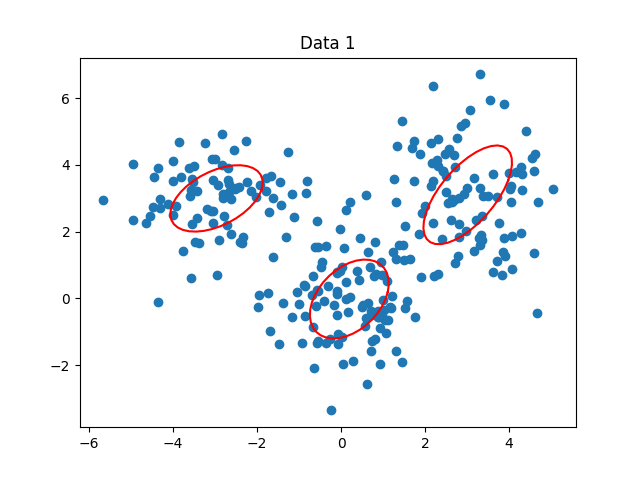
weights, means, covariances, \_ = MStep(gamma, data)

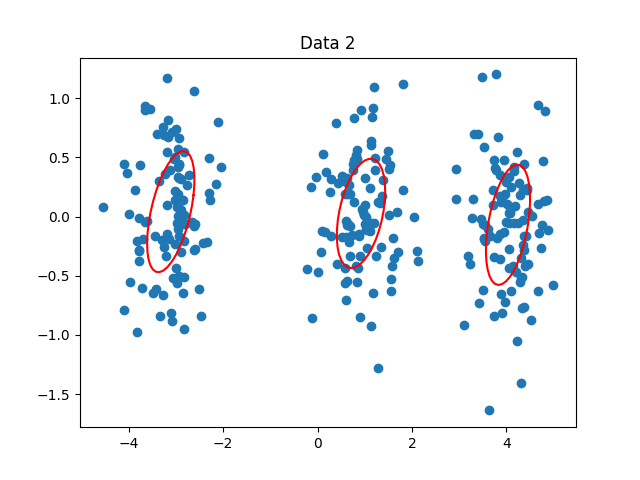
for k in range(K):

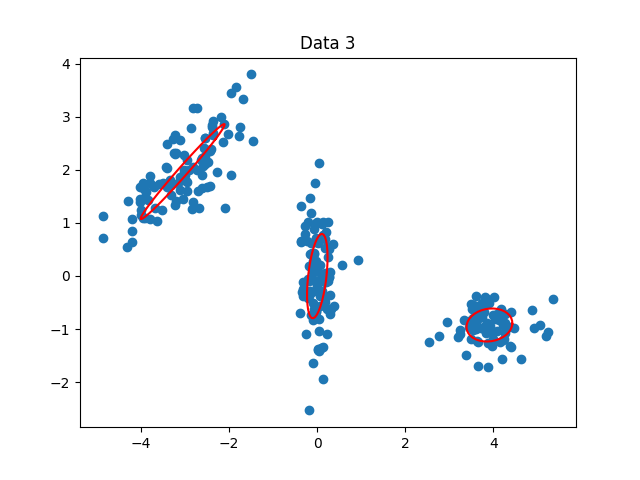
covariances[:,:,k] = regularize\_cov(covariances[:,:,k], epsilon)

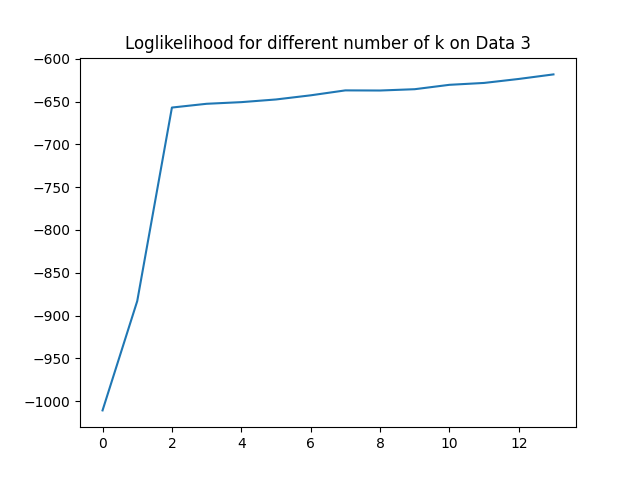
return [weights, means, covariances]

f)









We can clearly see that anything beyond K=2 has no significant improvement. K=2 is optimum.