1. **Part II, Question 1**

```
iteratorVal = {0:2, 1:2} #number of columns of second matrix, number of rows for first

db = [ (0, ((0, 1), -1)), (0, ((0, 2), 2)), (0, ((1, 0), 4)), (0, ((1, 1), 11)), (0, ((1, 2), 2)), (1, ((0, 0), 3)),
(1, ((0, 1), -1)), (1, ((1, 0), 1)), (1, ((1, 1), 2)), (1, ((2, 0), 6)), (1, ((2, 1), 1))]  # Example matrix
matrix = sc.parallelize(db)      # Can also read from the disk/ db/ table.

# Checking for matrix 0/1 and creating (i, k) intermediate key value pairs accordingly
matrix = matrix.flatMap(lambda x: [((x[1][0][0], iter), (x[0], x[1][0][1], x[1][1])) if (x[0]==0) else
((iter, x[1][0][1]), (x[0],  x[1][0][0], x[1][1])) for iter in range(iteratorVal[x[0]])])

matrix = matrix.groupByKey()
matrix = matrix.map(reducer).filter(lambda x: x[1]!=0).collect() #Filter to produce a sparse
matrix at the end
# Technically, GroupByKey + Map = Reduce operation. Hence the single pass is not violated!

def reducer(x):
	x1 = list(x[1])
	m1, m2 = [], []
	for i in x1:
		if(list(i)[0] == 0):
			m1.append(list(i))
		else:
			m2.append(list(i))

	m1 = sorted(m1, key=lambda x: x[1] )
	m2 = sorted(m2, key=lambda x: x[1] )
	maxIndex = max(m1[-1][1], m2[-1][1])
	for i in range(maxIndex):
	# Handling sparse matrix resulting in different intermediate key value pairs.
		if(m1[i][1] != i):
			m1.insert(i, None)
		if(m2[i][1] != i):
			m2.insert(i, None)

	val = 0
	for i in range(maxIndex+1):
		if(m1[i] == None or m2[i] == None ):
			val  = val
		elif(m1[i][1] == i and m2[i][1] == i):     #Multiplying the j-th value alone
			val += (m1[i][2]*m2[i][2] )
	return (x[0], val)
```

2. **Part II, Question 2**

```
#Certain variables like input features are assumed to be defined already.
import tensorflow as tf
X = tf.constant(features, dtype=tf.float32, name="X")
y = tf.constant(y.reshape(-1,1), dtype=tf.float32, name="y")
n_epochs = 10
learning_rate = 0.0001
penalty = tf.constant(1.0, dtype=tf.float32, name="penalty")
beta = tf.Variable(tf.random_uniform([features.shape[1], 1], -1., 1.), name = "beta")
m = tf.Variable(tf.zeros([features.shape[1], 1]), name="momentum_weights")
momentum = 0.9
y_pred = tf.matmul(X, beta, name="predictions")
penalizedCost = tf.reduce_sum(tf.square(y - y_pred)) + penalty * tf.reduce_sum(tf.square(beta))
grads = tf.gradients(penalizedCost, [beta])[0]
momentum_op = tf.assign(m, momentum*m + learning_rate*grads)
training_op = tf.assign(beta, beta-momentum_op)

init = tf.global_variables_initializer()
with tf.Session() as sess:
 sess.run(init)
 for epoch in range(n_epochs):
  if epoch %1 == 0: #print debugging output
   print("Epoch", epoch, "; penalizedCost =", int(penalizedCost.eval()))
  sess.run(training_op)
 best_beta = beta.eval()
```

RESULT ANALYSIS

Momentum: 0.9

('Epoch', 0, '; penalizedCost =', 248912)
('Epoch', 1, '; penalizedCost =', 208998)
('Epoch', 2, '; penalizedCost =', 146552)
('Epoch', 3, '; penalizedCost =', 83685)
('Epoch', 4, '; penalizedCost =', 38788)
('Epoch', 5, '; penalizedCost =', 20954)
('Epoch', 6, '; penalizedCost =', 28629)
('Epoch', 7, '; penalizedCost =', 52131)
('Epoch', 8, '; penalizedCost =', 78465)
('Epoch', 9, '; penalizedCost =', 96344)
('Mean Absolute Error:', 13.02788132525499)
('Pearson Correlation:', (0.6373451939992985, 4.927043496055146e-07))

Momentum: 0.0
('Epoch', 0, '; penalizedCost =', 289662)
('Epoch', 1, '; penalizedCost =', 242613)
('Epoch', 2, '; penalizedCost =', 203794)
('Epoch', 3, '; penalizedCost =', 171763)
('Epoch', 4, '; penalizedCost =', 145335)
('Epoch', 5, '; penalizedCost =', 123528)
('Epoch', 6, '; penalizedCost =', 105535)
('Epoch', 7, '; penalizedCost =', 90689)
('Epoch', 8, '; penalizedCost =', 78439)
('Epoch', 9, '; penalizedCost =', 68331)
('Mean Absolute Error:', 9.376570549976392)
('Pearson Correlation:', (0.6373451939992985, 4.927043496055146e-07))

We can clearly see that the momentum helps in stable reduction of loss and better performance sooner.

Link to code

3. **Part II, Question 3**

A. If the Jaccard similarity between two sets (say A, B) = 0, then there is no element that lies in common between them.

This means that no element (∊ union(A, B)) would have result in a tuple of [1, 1] in the characteristic table together.

This implies no hash function can produce the same value [of first row with a 1] for both these sets. Hence the minhashing estimate would also be 0.

B.

```
import numpy as np
import random

rdd = sc.textFile(hdfs://*.json)          #Reading all the json files
rdd = rdd.map(json.loads)

#To club all the elements across different partition belonging to the same set
rdd = rdd.reduceByKey( lambda x,y: x.union(y))
# Op:  {'set Name 1': set(element1, element2, ...), 'set Name 2': set(element1, element2, ...)
...}

#Collects all the elements from all the sets
rows = rdd.reduce( lambda x, y: x.union(y))

numSets = 0            # Can also be computed as count of keys

def getCharacteristicMatrix(x):
# Returns the characteristic Matrix for a given set
        numSets += 1
        row = list(rows[0])      # converting set to list
        elements = list(x[1])   # converting set to list
        element_flag = []
        for i in row:
                element_flag.append(elements.count(i)>0)
        return (x[0], np.array(element_flag))

charMatrix = rdd.map(getCharacteristicMatrix).reduce(lambda x,y:
np.concatenate((x.reshape(-1,1), y.reshape(-1,1)), axis=1) )
#Turns it to (num_elements , numsets): Characteristic matrix. Easier to fetch a row now.

hashes = [getHfunc(randint(1,10000)) for i in range(500)]  #500 hash functions, r seeded
```

```
signatureMat = np.zeros(size=(500, numSets))        # Signature Matrix init
signatureMat[:, :] = np.inf

def minHashing(x):
    rowNum = getRowNum(charMatrix, x)        #Function that searches the row num
    row = list(rows[0])
    h = [hashes[i](row[rowNum]) for i in range(500)]      #Hash results for that row
    for s in range(numSets):
        If x[s] == 1:
            for i in range(500):
                if (h[i] < signatureMat[i][s]):
                    signatureMat[i][s] = h[i]
                else:
                    pass


#foreach on 2d np matrix reads line by line
charMatrix = charMatrix.foreach(minHashing)

print (signatureMat)
```

4. **Part II, Question 4**

A. Band size r=5; implies b = 100, jaccard = 0.75
Probability that set1 and set2 matches in 1 band: $(0.75)^5 = 0.237$
Probability that set1 and set2 didn't match in 1 band: $1 - (0.75)^5 = 0.763$

Probability that set1 and set2 matches in at least 1 band
= 1 - Probability that set1 and set2 matched in no band
= $1 - (0.763)^{100}$ = **0.9999999999982817** ~= 0.999

B. Probability that set1 and set2 matches in a band = s^r
Probability that set1 and set2 don't match in a band = $1 - s^r$

Probability that they match in at least 1 band = $1 - (1 - s^r)^b$
$1 - (1 - s^r)^b >= 0.99$
$0.01 >= (1 - s^r)^b$
s = 0.9
$0.01 >= (1 - 0.9^r)^b$
$\log(0.01) >= b \log(1 - 0.9^r)$

b * r = n [Taking n to be 500 from q3(b)]

b = 27.9, r =17.8
Since they need to be whole numbers **(27, 18)**, since higher b would imply higher false positive rate.

C. Since some false positive rate is allowed to be a bit more than the usual implementation, I would follow the following approach.

I would pick first 'p' rows from the characteristic matrix [Having p smaller than total number of rows, for large speedup]
And I would build the minHashing on this, so that it has fewer comparisons to make.
If both sets have a minhash for a row it is dealt the normal way based on the equality/inequality.

If a set doesn't have its minHash value within the first p rows and the other set has, then it is counted as unequal minhash.

If both the sets being compared don't have a minHash from the first p rows, then it is not considered a valid example.

Due to the last condition the denominator tends to decrease, which would increase the false positive rate. However the speedup would be large.