- I ran it on following test set:
  t1 - t10
- It accurately reported it SPAM/NOT SPAM in 7 of total cases
- It inaccurately reported 4 of non spam as spam
- **Precision: 70%. Recall: 100%**
- I also noticed that if the mail is a bit bigger (t6), then the probability value becomes so less that computer starts treating it as 0, and thus whole calculation for SPAM goes for toss. So, I have put a check that once the probability value gets set to 0, then take the decision based on the previous value of the probability
- I also noticed that even after implementing the above point, a SPAM mail was not reported as SPAM. This is because I had not implemented smoothing and so I was taking the probability of the word not found as 1. So, this was causing issue. To address it, instead of implementing a full fledged smoothing, I implemented a basic smoothing wherein if a particular word was not found in the training data, then I just multiplied it with the 1/(notSpamCnt or spamCnt). This improved the accuracy and thus the number of FP reduced
- I noticed that even if the mail is not spam, still it can be reported as spam. This behavior is highly attributed to the training data
- Following table represents the above data:

Test File Name	SPAM in real?	Reported?	Accuracy	FP
t1	1	1	1	0
t2	1	1	1	0
t3	1	1	1	0
t4	0	0	1	0
t5	1	1	1	0
t6	1	1	1	0
t7	0	1	0	1
t8	1	1	1	0
t9	0	1	0	1
t10	0	1	0	1
Total			7	3

- I can see following ways in which its accuracy can be improved:
  - Smoothing – since I have used hashing for faster searching, the implementation of smoothing becomes very complex. But if someone can use maps (C++ maps) then it can be done easily but then the search time will be O(logn)
  - Not making distinction between similar looking words – For eg, “compare” and “compared” can be taken as similar looking word. It will require some more preprocessing but will give better results