Q1

1a) The SparseVector is like a dictionary expect that it doesn't show any non-value key.

1b) SparseVector inherit all the method in dict but dict doesn't share the methods which are built in SparseVector

The example is shown below:

```
>>> import SparseVector as sv
>>> a= {'a' : 1}
>>> b= {'a' : 2}
>>> c=sv.SparseVector(a)
>>> d=sv.SparseVector(b)
>>> cmp(a,b)
-1
>>> cmp(c,d)
-1
>>> a.__eq__(b)
False
>>> c.__eq__(d)
False
>>> c.dot(d)
2
>>> a.dot(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'dict' object has no attribute 'dot'
>>>
```

1c)

```
>>> a= {'a' : 1}
>>> b= {'a' : 2}
>>> c=sv.SparseVector(a)
>>> d=sv.SparseVector(b)
>>> c + d
Into __add__ function.

{'a': 3}
>>> c.dot(d)
2
>>> c.__mul__(2)
{'a': 2}
>>> c.__rmul__(2)
{'a': 2}
>>>
```

Q2

2a) $l(\beta: x_i, y_i) = \log(1 + e^{-y\beta^T x})$

$$\nabla l = \frac{-yx * e^{-y\beta^T x}}{1+e^{-y\beta^T x}} = \frac{-yx}{1+e^{y\beta^T x}}$$

$$\nabla^2 l = \frac{y^2 x^2 e^{y\beta^T x}}{(1+e^{y\beta^T x})^2}$$

$(1 + e^{y\beta^T x})^2 > 0$ and since $y \in \{-1, 1\}$ so $y^2 > 0$ and $x^2 \geq 0$ and $e^{y\beta^T x} > 0$

Then we have $Cl = \frac{y^2 x^2 e^{y\beta^T x}}{(1+e^{y\beta^T x})^2} \geq 0$

So, the function $l(\beta: x_i, y_i) = \log(1 + e^{-y\beta^T x})$ is convex

2b) as we proved on the above

$$\nabla l = \frac{-yx * e^{-y\beta^T x}}{1 + e^{-y\beta^T x}} = \frac{-yx}{1 + e^{y\beta^T x}}$$

2c) $l(\beta: x_i, y_i) = \log(1 + e^{-y\beta^T x})$

When x = 0, $l(\beta: x_i, y_i) = \log(1) = 0$

So, we know that $l(\beta: x_i, y_i)$ $do\ not\ depend\ on\ \beta_j$ $where\ the\ coordinate$ $x_j = 0$

When x = 0 $\nabla l = \frac{-yx}{1+e^{y\beta^T x}} = \frac{0}{2} = 0$

So, we know that ∇l $do\ not\ depend\ on\ \beta_j$ $where\ the\ coordinate$ $x_j = 0$

2d) $L(\beta) = \sum_{i=1}^{n} l(\beta: x_i, y_i) + \lambda||\beta||^2$

Since $l(\beta: x_i, y_i)$ is convex so $\sum_{i=1}^{n} l(\beta: x_i, y_i)$ is convex.

Let $f(\beta) = \lambda||\beta||^2$. Then $\nabla f = 2\lambda\beta$ and $\nabla^2 f = 2\lambda$ which is a constant

So $L(\beta) = \sum_{i=1}^{n} l(\beta: x_i, y_i) + \lambda||\beta||^2$ is convex

Q3

3a)

```
def gradLogisticLoss(beta,x,y):
    """
        Given a sparse vector beta, a sparse vector x, and
        a binary value y in {-1,+1}, compute the gradient of the logistic loss

              ∇l(B;x,y) = -y / (1.0 + exp(y <β,x> )) * x

    The input is:
        - beta: a sparse vector β
        - x: a sparse vector x
            - y: a binary value in {-1,+1}


    """
    return  x*((-float(y))/(1.0+np.exp(float(y)*beta.dot(x))))
```

```
def gradTotalLoss(data,beta, lam = 0.0):
    """  Given a sparse vector beta and a dataset compute the gradient of regularized total logistic loss :

            ∇L(β) = Σ_{(x,y) in data}  ∇l(β;x,y)  + 2λ β

        Inputs are:
           - data: a python list containing pairs of the form (x,y), where x is a sparse vector and y is a binary value
           - beta: a sparse vector β
           - lam: the regularization parameter λ
    """
    gradTotalLoss= SparseVector({})
    for (x,y) in data:
        gradTotalLoss = gradLogisticLoss(beta,x,y) + gradTotalLoss
    return gradTotalLoss + 2.0 * lam * beta
```

```
def logisticLoss(beta,x,y):
    """
        Given sparse vector beta, a sparse vector x, and a binary value y in {-1,+1}, compute the logistic loss

              l(β;x,y) = log( 1.0 + exp(-y * <β,x>) )

    The input is:
        - beta: a sparse vector β
        - x: a sparse vector x
            - y: a binary value in {-1,+1}

    """
    return np.log(1.0 + np.exp (-float(y)*beta.dot(x)))
```

3b)

```
def test(data,beta):
    """ Output the quantities necessary to compute the accuracy, precision, and recall of the prediction of labels in a dataset und

        The accuracy (ACC), precision (PRE), and recall (REC) are defined in terms of the following sets:

                P = datapoints (x,y) in data for which <β,x> > 0
                N = datapoints (x,y) in data for which <β,x> <= 0

                TP = datapoints in (x,y) in P for which y=+1
                FP = datapoints in (x,y) in P for which y=-1
                TN = datapoints in (x,y) in N for which y=-1
                FN = datapoints in (x,y) in N for which y=+1

        For #XXX the number of elements in set XXX, the accuracy, precision, and recall of parameter vector β over data are defined

                ACC(β,data) = ( #TP+#TN ) / (#P + #N)
                PRE(β,data) = #TP / (#TP + #FP)
                REC(β,data) = #TP/ (#TP + #FN)

        Inputs are:
             - data: an RDD containing pairs of the form (x,y)
             - beta: vector β

        The return values are
             - ACC, PRE, REC

    """
```

```
    totalscore = [(beta.dot(x), y) for (x,y) in data]

    P=N=TP=FP=TN=FN=0
    for (x,y) in totalscore:
        P += x > 0
        N += x < 0
        TP += x > 0 and float(y) == 1.0
        FP += x > 0 and float(y) == -1.0
        TN += x < 0 and float(y) == -1.0
        TP += x < 0 and float(y) == 1.0

    ACC = (1.0 * (TP +TN)) / (1.0 * (P + N))
    PRE = (1.0 * TP) /(1.0*(TP + FP))
    REC = (1.0 * TP) / (1.0 * (TP + FN))
    return ACC, PRE, REC
```

3c)

1. lambda = 0

```
k = 0 t = 3.0693769455 L(β_k) = 5140.37949103 ||∇L(β_k)||_2 = 4273.54823303
    gamma = 0.000470184984576 ACC = 0.909 PRE = 0.855325914149 REC = 1.0
k = 1 t = 6.7778429985 L(β_k) = 2516.99399449 ||∇L(β_k)||_2 = 3275.9864649
    gamma = 0.000169266594447 ACC = 0.92 PRE = 0.892123287671 REC = 0.968401486989
k = 2 t = 9.47433996201 L(β_k) = 1536.62960892 ||∇L(β_k)||_2 = 802.337284031
    gamma = 0.00362797056 ACC = 0.979 PRE = 0.965765765766 REC = 0.996282527881
k = 3 t = 12.833411932 L(β_k) = 772.339148079 ||∇L(β_k)||_2 = 980.549279124
    gamma = 0.000470184984576 ACC = 0.974 PRE = 0.981203007519 REC = 0.970260223048
k = 4 t = 16.1770298481 L(β_k) = 648.968995294 ||∇L(β_k)||_2 = 636.730178332
    gamma = 0.000470184984576 ACC = 0.976 PRE = 0.972426470588 REC = 0.983271375465
k = 5 t = 19.3420748711 L(β_k) = 583.12930033 ||∇L(β_k)||_2 = 390.121504419
    gamma = 0.00078364164096 ACC = 0.975 PRE = 0.981238273921 REC = 0.972118959108
k = 6 t = 22.663271904 L(β_k) = 554.863687101 ||∇L(β_k)||_2 = 474.38957367
    gamma = 0.000470184984576 ACC = 0.981 PRE = 0.977900552486 REC = 0.986988847584
k = 7 t = 25.8196530342 L(β_k) = 511.658985564 ||∇L(β_k)||_2 = 258.887387255
    gamma = 0.00078364164096 ACC = 0.98 PRE = 0.981412639405 REC = 0.981412639405
k = 8 t = 28.9538359642 L(β_k) = 486.942281985 ||∇L(β_k)||_2 = 276.73548299
    gamma = 0.00078364164096 ACC = 0.984 PRE = 0.976277372263 REC = 0.994423791822
k = 9 t = 32.0857009888 L(β_k) = 467.333365045 ||∇L(β_k)||_2 = 297.335681259
    gamma = 0.00078364164096 ACC = 0.979 PRE = 0.981378026071 REC = 0.979553903346
k = 10 t = 35.3914339542 L(β_k) = 449.768322793 ||∇L(β_k)||_2 = 319.530735224
    gamma = 0.000470184984576 ACC = 0.984 PRE = 0.979779411765 REC = 0.990706319703
k = 11 t = 38.1994228363 L(β_k) = 425.924544297 ||∇L(β_k)||_2 = 166.721201409
    gamma = 0.002176782336 ACC = 0.983 PRE = 0.988742964353 REC = 0.979553903346
k = 12 t = 41.4900939465 L(β_k) = 405.994303585 ||∇L(β_k)||_2 = 388.199259096
    gamma = 0.000470184984576 ACC = 0.986 PRE = 0.979853479853 REC = 0.994423791822
k = 13 t = 44.2999830246 L(β_k) = 374.250964742 ||∇L(β_k)||_2 = 147.167375122
    gamma = 0.002176782336 ACC = 0.987 PRE = 0.994350282486 REC = 0.981412639405
k = 14 t = 47.5975329876 L(β_k) = 364.103711306 ||∇L(β_k)||_2 = 357.709734875
    gamma = 0.000470184984576 ACC = 0.989 PRE = 0.985267034991 REC = 0.994423791822
k = 15 t = 50.2412428856 L(β_k) = 335.533358123 ||∇L(β_k)||_2 = 119.375620574
    gamma = 0.00362797056 ACC = 0.989 PRE = 0.994371482176 REC = 0.985130111524
k = 16 t = 53.5357489586 L(β_k) = 321.40210325 ||∇L(β_k)||_2 = 359.393232945
    gamma = 0.000470184984576 ACC = 0.994 PRE = 0.992592592593 REC = 0.996282527881
k = 17 t = 55.707572937 L(β_k) = 290.83776835 ||∇L(β_k)||_2 = 93.2783034072
    gamma = 0.01679616 ACC = 0.99 PRE = 0.994382022472 REC = 0.986988847584
k = 18 t = 58.8442599773 L(β_k) = 238.234800471 ||∇L(β_k)||_2 = 427.837337396
    gamma = 0.00078364164096 ACC = 0.997 PRE = 0.994454713494 REC = 1.0
k = 19 t = 61.8107469082 L(β_k) = 183.512335128 ||∇L(β_k)||_2 = 104.689949815
    gamma = 0.0013060694016 ACC = 0.997 PRE = 0.994454713494 REC = 1.0
```

2. lambda = 5

```
k = 0 t = 3.13104200363 L(β_k) = 5140.37949103 ||∇L(β_k)||_2 = 4273.54823303
    gamma = 0.000470184984576 ACC = 0.909 PRE = 0.855325914149 REC = 1.0
k = 1 t = 6.85882997513 L(β_k) = 2537.18159657 ||∇L(β_k)||_2 = 3277.82790658
    gamma = 0.000169266594447 ACC = 0.92 PRE = 0.892123287671 REC = 0.968401486989
k = 2 t = 9.57383394241 L(β_k) = 1558.60586113 ||∇L(β_k)||_2 = 794.293677188
    gamma = 0.00362797056 ACC = 0.979 PRE = 0.965765765766 REC = 0.996282527881
k = 3 t = 12.9773328304 L(β_k) = 915.720160974 ||∇L(β_k)||_2 = 1143.26672935
    gamma = 0.000470184984576 ACC = 0.97 PRE = 0.981060606061 REC = 0.96282527881
k = 4 t = 16.3566009998 L(β_k) = 781.406954891 ||∇L(β_k)||_2 = 801.415491657
    gamma = 0.000470184984576 ACC = 0.977 PRE = 0.970749542962 REC = 0.986988847584
k = 5 t = 19.7643377781 L(β_k) = 702.27507752 ||∇L(β_k)||_2 = 498.211120758
    gamma = 0.000470184984576 ACC = 0.98 PRE = 0.981412639405 REC = 0.981412639405
k = 6 t = 22.9591138363 L(β_k) = 654.479647213 ||∇L(β_k)||_2 = 291.418683865
    gamma = 0.00078364164096 ACC = 0.98 PRE = 0.976102941176 REC = 0.986988847584
k = 7 t = 26.321792841 L(β_k) = 631.507345274 ||∇L(β_k)||_2 = 336.79128005
    gamma = 0.000470184984576 ACC = 0.98 PRE = 0.981412639405 REC = 0.981412639405
k = 8 t = 29.3564989567 L(β_k) = 605.740256935 ||∇L(β_k)||_2 = 198.81186565
    gamma = 0.0013060694016 ACC = 0.984 PRE = 0.976277372263 REC = 0.994423791822
k = 9 t = 32.7414338589 L(β_k) = 589.009131316 ||∇L(β_k)||_2 = 342.77320503
    gamma = 0.000470184984576 ACC = 0.98 PRE = 0.97962962963 REC = 0.983271375465
k = 10 t = 35.7747569084 L(β_k) = 564.819571171 ||∇L(β_k)||_2 = 169.651898158
    gamma = 0.0013060694016 ACC = 0.984 PRE = 0.976277372263 REC = 0.994423791822
k = 11 t = 39.1480169296 L(β_k) = 555.998065612 ||∇L(β_k)||_2 = 306.79989283
    gamma = 0.000470184984576 ACC = 0.982 PRE = 0.979704797048 REC = 0.986988847584
k = 12 t = 42.2003118992 L(β_k) = 536.412663316 ||∇L(β_k)||_2 = 141.992440299
    gamma = 0.0013060694016 ACC = 0.984 PRE = 0.976277372263 REC = 0.994423791822
k = 13 t = 45.5722129345 L(β_k) = 528.459592179 ||∇L(β_k)||_2 = 244.661438016
    gamma = 0.000470184984576 ACC = 0.985 PRE = 0.981583793738 REC = 0.990706319703
k = 14 t = 48.6270008087 L(β_k) = 515.159794983 ||∇L(β_k)||_2 = 115.957447378
    gamma = 0.0013060694016 ACC = 0.986 PRE = 0.979853479853 REC = 0.994423791822
k = 15 t = 51.8252668381 L(β_k) = 506.701683219 ||∇L(β_k)||_2 = 177.001664477
    gamma = 0.00078364164096 ACC = 0.989 PRE = 0.988868274583 REC = 0.990706319703
k = 16 t = 55.1756467819 L(β_k) = 501.145796447 ||∇L(β_k)||_2 = 186.787072656
    gamma = 0.000470184984576 ACC = 0.988 PRE = 0.983455882353 REC = 0.994423791822
k = 17 t = 58.061757803 L(β_k) = 492.950591036 ||∇L(β_k)||_2 = 93.6448314909
    gamma = 0.002176782336 ACC = 0.991 PRE = 0.992551210428 REC = 0.990706319703
k = 18 t = 61.4283797741 L(β_k) = 485.712728307 ||∇L(β_k)||_2 = 209.942518822
    gamma = 0.000470184984576 ACC = 0.991 PRE = 0.987108655617 REC = 0.996282527881
k = 19 t = 64.3124659061 L(β_k) = 476.11914913 ||∇L(β_k)||_2 = 83.0571978465
    gamma = 0.002176782336 ACC = 0.994 PRE = 0.994423791822 REC = 0.994423791822
```

3.lambda = 10

```
k = 0 t = 3.06997394562 L(β_k) = 5140.37949103 ||∇L(β_k)||_2 = 4273.54823303
    gamma = 0.000470184984576 ACC = 0.909 PRE = 0.855325914149 REC = 1.0
k = 1 t = 6.79761481285 L(β_k) = 2557.36919865 ||∇L(β_k)||_2 = 3279.79141961
    gamma = 0.000169266594447 ACC = 0.921 PRE = 0.893653516295 REC = 0.968401486989
k = 2 t = 9.46369981766 L(β_k) = 1580.46700693 ||∇L(β_k)||_2 = 786.723977579
    gamma = 0.00362797056 ACC = 0.98 PRE = 0.965827338129 REC = 0.998141263941
k = 3 t = 12.9577608109 L(β_k) = 1064.08936255 ||∇L(β_k)||_2 = 1329.31623914
    gamma = 0.000282110990746 ACC = 0.976 PRE = 0.975925925926 REC = 0.979553903346
k = 4 t = 15.7814469337 L(β_k) = 826.628992936 ||∇L(β_k)||_2 = 322.71356109
    gamma = 0.002176782336 ACC = 0.984 PRE = 0.976277372263 REC = 0.994423791822
k = 5 t = 19.1110758781 L(β_k) = 769.911300389 ||∇L(β_k)||_2 = 649.896610641
    gamma = 0.000470184984576 ACC = 0.977 PRE = 0.981308411215 REC = 0.975836431227
k = 6 t = 22.4678769112 L(β_k) = 704.572834024 ||∇L(β_k)||_2 = 295.29440558
    gamma = 0.000470184984576 ACC = 0.982 PRE = 0.977941176471 REC = 0.988847583643
k = 7 t = 25.6199288368 L(β_k) = 687.554757621 ||∇L(β_k)||_2 = 171.568212543
    gamma = 0.00078364164096 ACC = 0.981 PRE = 0.981447124304 REC = 0.983271375465
k = 8 t = 28.9768409729 L(β_k) = 677.553578933 ||∇L(β_k)||_2 = 198.237479477
    gamma = 0.000470184984576 ACC = 0.984 PRE = 0.979779411765 REC = 0.990706319703
k = 9 t = 31.9720058441 L(β_k) = 668.019557923 ||∇L(β_k)||_2 = 130.722159922
    gamma = 0.0013060694016 ACC = 0.983 PRE = 0.981515711645 REC = 0.986988847584
k = 10 t = 35.2807548046 L(β_k) = 659.584652805 ||∇L(β_k)||_2 = 232.290087336
    gamma = 0.000470184984576 ACC = 0.986 PRE = 0.979853479853 REC = 0.994423791822
k = 11 t = 38.4625279903 L(β_k) = 649.522167599 ||∇L(β_k)||_2 = 130.536027197
    gamma = 0.00078364164096 ACC = 0.986 PRE = 0.983394833948 REC = 0.990706319703
k = 12 t = 41.7711257935 L(β_k) = 644.278385475 ||∇L(β_k)||_2 = 152.946794124
    gamma = 0.000470184984576 ACC = 0.986 PRE = 0.979853479853 REC = 0.994423791822
k = 13 t = 44.7525119781 L(β_k) = 638.714924259 ||∇L(β_k)||_2 = 98.4101786566
    gamma = 0.0013060694016 ACC = 0.988 PRE = 0.987037037037 REC = 0.990706319703
k = 14 t = 48.0632238388 L(β_k) = 633.804194941 ||∇L(β_k)||_2 = 173.008101859
    gamma = 0.000470184984576 ACC = 0.986 PRE = 0.979853479853 REC = 0.994423791822
k = 15 t = 51.2369430065 L(β_k) = 627.906060556 ||∇L(β_k)||_2 = 95.9654162555
    gamma = 0.00078364164096 ACC = 0.99 PRE = 0.987084870849 REC = 0.994423791822
k = 16 t = 54.3993918896 L(β_k) = 624.494970313 ||∇L(β_k)||_2 = 108.111406092
    gamma = 0.00078364164096 ACC = 0.988 PRE = 0.981684981685 REC = 0.996282527881
k = 17 t = 57.7104058266 L(β_k) = 621.86525317 ||∇L(β_k)||_2 = 127.023446288
    gamma = 0.000470184984576 ACC = 0.99 PRE = 0.987084870849 REC = 0.994423791822
k = 18 t = 60.6884288788 L(β_k) = 618.17799926 ||∇L(β_k)||_2 = 74.274509088
    gamma = 0.0013060694016 ACC = 0.989 PRE = 0.983486238532 REC = 0.996282527881
k = 19 t = 64.0021719933 L(β_k) = 615.292058287 ||∇L(β_k)||_2 = 126.552208232
    gamma = 0.000470184984576 ACC = 0.991 PRE = 0.988909426987 REC = 0.994423791822
```

3d) Precision is $\frac{TP}{TP+FP}$ and Recall is $\frac{TP}{TP+FN}$

High precision means in the normal mushroom label there's majority normal mushrooms and poor poisoned mushroom. So, in this case we should use high precision.

Q4

4a)

```
def getAllFeaturesRDD(dataRDD):
    """ Get all the features present in grouped dataset dataRDD.

    The input is:
            - dataRDD containing pairs of the form (SparseVector(x),y).

        The return value is an RDD containing the union of all unique features present in sparse vectors inside dataRDD.
    """
    featuresRDD = dataRDD.flatMap(lambda (x,y) : x.keys() )\
                         .distinct()
    return featuresRDD

def totalLossRDD(dataRDD,beta,lam = 0.0):
    """  Given a sparse vector beta and a dataset  compute the regularized total logistic loss :

                L(β) = Σ_{(x,y) in data}  l(β;x,y)  + λ ||β ||_2^2

          Inputs are:
             - data: a python list containing pairs of the form (x,y), where x is a sparse vector and y is a binary value
             - beta: a sparse vector β
             - lam: the regularization parameter λ
    """
    loss = dataRDD.map(lambda (x,y) : logisticLoss(beta,x,y)).reduce(lambda x,y : x + y)
    return loss + 1.0 * lam* beta.dot(beta)


def gradTotalLossRDD(dataRDD,beta,lam = 0.0):
    """  Given a sparse vector beta and a dataset compute the gradient of regularized total logistic loss :

               ∇L(β) = Σ_{(x,y) in data}  ∇l(β;x,y)  + 2λ β

          Inputs are:
             - data: a python list containing pairs of the form (x,y), where x is a sparse vector and y is a binary value
             - beta: a sparse vector β
             - lam: the regularization parameter λ
    """
    gradTotalLoss =dataRDD.map(lambda (x,y) : gradLogisticLoss(beta,x,y))\
                          .reduce(lambda x,y : x + y)
    return gradTotalLoss + 2.0 * lam * beta
```

```
def test(dataRDD,beta):
    """ Output the quantities necessary to compute the accuracy, precision, and recall of the prediction of labels in a dataset under a given β.

        The accuracy (ACC), precision (PRE), and recall (REC) are defined in terms of the following sets:

                P = datapoints (x,y) in data for which <β,x> > 0
                N = datapoints (x,y) in data for which <β,x> <= 0

                TP = datapoints in (x,y) in P for which y=+1
                FP = datapoints in (x,y) in P for which y=-1
                TN = datapoints in (x,y) in N for which y=-1
                FN = datapoints in (x,y) in N for which y=+1

        For #XXX the number of elements in set XXX, the accuracy, precision, and recall of parameter vector β over data are defined as:

                ACC(β,data) = ( #TP+#TN ) / (#P + #N)
                PRE(β,data) = #TP / (#TP + #FP)
                REC(β,data) = #TP/ (#TP + #FN)

        Inputs are:
             - data: an RDD containing pairs of the form (x,y)
             - beta: vector β

        The return values are
             - ACC, PRE, REC

    """
    totalscore = dataRDD.map(lambda (x,y): (beta.dot(x), y))
    PositiveS = totalscore.filter(lambda (x,y) : x > 0 )
    NegetiveS = totalscore.filter(lambda (x,y) : x <= 0 )
    P = PositiveS.count()
    N = NegetiveS.count()
    TP = PositiveS.filter(lambda (x,y) : float(y) == 1.0)\
                  .count()
    FP = PositiveS.filter(lambda (x,y) : float(y) == -1.0)\
                  .count()
    TN = NegetiveS.filter(lambda (x,y) : float(y) == -1.0)\
                  .count()
    FN = NegetiveS.filter(lambda (x,y) : float(y) == 1.0)\
                  .count()
    ACC = (1.0 * (TP +TN)) / (1.0 * (P + N))
    PRE = (1.0 * TP) /(1.0*(TP + FP))
    REC = (1.0 * TP) / (1.0 * (TP + FN))
    return ACC, PRE, REC
```

```
def train(dataRDD,beta_0,lam,max_iter,eps,f,test_data=None):
    """ Train a logistic classifier from deta.

    The function minimizes:

                L(β) = Σ_{(x,y) in data}  l(β;x,y)  + λ ||β||_2^2

        using gradient descent.

        Inputs are:
            - data: a python list containing pairs of the form (x,y), where x is a sparse vector and y is a binary value
            - beta_0: an initial sparse vector β_0
            - lam: the regularization parameter λ
            - max_iter: the maximum number of iterations
            - eps: the tolerance ε
            - test_data (optional): data over which model β is tested in each iteration w.r.t. accuracy, precision, and recall

        The return values are:
            - beta: the trained β, as a sparse vector
            - gradNorm: the norm ||∇L(β)||_2
            - k: the number of iterations

    """
```

```
    k = 0
    gradNorm = 2*eps
    beta = beta_0
    start = time()
    while k<max_iter and gradNorm > eps:
        obj = totalLossRDD(dataRDD,beta,lam)

        grad = gradTotalLossRDD(dataRDD,beta,lam)
        gradNormSq = grad.dot(grad)
        gradNorm = np.sqrt(gradNormSq)

        fun = lambda x: totalLossRDD(dataRDD,x,lam)
        gamma = lineSearch(fun,beta,grad,obj,gradNormSq)

        beta = beta - gamma * grad
        if test_data == None:
            print>>f, 'k = ',k,'\tt = ',time()-start,'\tL(β_k) = ',obj,'\t||∇L(β_k)||_2 = ',gradNorm,'\tgamma = ',gamma
        else:
            acc,pre,rec = test(test_data,beta)
            print>>f, 'k = ',k,'\tt = ',time()-start,'\tL(β_k) = ',obj,'\t||∇L(β_k)||_2 = ',gradNorm,'\tgamma = ',gamma,'\tACC = ',acc,'\tPRE = ',pre,'\tREC = ',rec
        k = k + 1

    return beta,gradNorm,k
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description = 'Logistic Regression.',formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--traindata',default=None, help='Input file containing (x,y) pairs, used to train a logistic model')
    parser.add_argument('--testdata',default=None, help='Input file containing (x,y) pairs, used to test a logistic model')
    parser.add_argument('--beta', default='beta', help='File where beta is stored (when training) and read from (when testing)')
    parser.add_argument('--lam', type=float,default=0.0, help='Regularization parameter λ')
    parser.add_argument('--max_iter', type=int,default=100, help='Maximum number of iterations')
    parser.add_argument('--eps', type=float, default=0.1, help='ε-tolerance. If the l2_norm gradient is smaller than ε, gradient descent terminates.')
    parser.add_argument('--out', default='out', help='The output file which saves the print result in the script')

    verbosity_group = parser.add_mutually_exclusive_group(required=False)
    verbosity_group.add_argument('--verbose', dest='verbose', action='store_true')
    verbosity_group.add_argument('--silent', dest='verbose', action='store_false')
    parser.set_defaults(verbose=True)

    args = parser.parse_args()
    sc = SparkContext(master='local[40]',appName='Parallel Logistic Regression')
    if not args.verbose :
        sc.setLogLevel("ERROR")

    f = open(args.out,'w')
    print>>f, 'Reading training data from',args.traindata
    traindata = readDataRDD(args.traindata,sc)
    print>>f, 'Read',traindata.count(),'data points with',getAllFeaturesRDD(traindata).count(),'features in total'

    if args.testdata is not None:
        print>>f, 'Reading test data from',args.testdata
        testdata = readDataRDD(args.testdata,sc)
        print>>f, 'Read',testdata.count(),'data points with',getAllFeaturesRDD(testdata).count(),'features'
    else:
        testdata = None

    beta0 = SparseVector({})
    print>>f, 'Training on data from',args.traindata,'with λ =',args.lam,', ε =',args.eps,', max iter = ',args.max_iter
    beta, gradNorm, k = train(traindata,beta_0=beta0,lam=args.lam,max_iter=args.max_iter,eps=args.eps,test_data=testdata,f=f)
    print>>f, 'Algorithm ran for',k,'iterations. Converged:',gradNorm<args.eps
    print>>f, 'Saving trained β in',args.beta
    writeBeta(args.beta,beta)
```

4b)

Using Parallel

```
k = 0 t = 8.79644703865 L(β_k) = 5140.37949103 ||∇L(β_k)||_2 = 4273.54823303
gamma = 0.000470184984576 ACC = 0.909 PRE = 0.855325914149 REC = 1.0
k = 1 t = 18.0698699951 L(β_k) = 2516.99399449 ||∇L(β_k)||_2 = 3275.9864649
gamma = 0.000169266594447 ACC = 0.92 PRE = 0.892123287671 REC = 0.968401486989
k = 2 t = 24.737361908 L(β_k) = 1536.62960892 ||∇L(β_k)||_2 = 802.337284031
gamma = 0.00362797056 ACC = 0.979 PRE = 0.965765765766 REC = 0.996282527881
k = 3 t = 32.9485578537 L(β_k) = 772.339148079 ||∇L(β_k)||_2 = 980.549279124
gamma = 0.000470184984576 ACC = 0.974 PRE = 0.981203007519 REC = 0.970260223048
k = 4 t = 41.2144808769 L(β_k) = 648.968995294 ||∇L(β_k)||_2 = 636.730178332
gamma = 0.000470184984576 ACC = 0.976 PRE = 0.972426470588 REC = 0.983271375465
k = 5 t = 49.0069289207 L(β_k) = 583.12930033 ||∇L(β_k)||_2 = 390.121504419
gamma = 0.00078364164096 ACC = 0.975 PRE = 0.981238273921 REC = 0.972118959108
k = 6 t = 57.2262909412 L(β_k) = 554.863687101 ||∇L(β_k)||_2 = 474.38957367
gamma = 0.000470184984576 ACC = 0.981 PRE = 0.977900552486 REC = 0.986988847584
k = 7 t = 65.1177899837 L(β_k) = 511.658985564 ||∇L(β_k)||_2 = 258.887387255
gamma = 0.00078364164096 ACC = 0.98 PRE = 0.981412639405 REC = 0.981412639405
k = 8 t = 73.0094220638 L(β_k) = 486.942281985 ||∇L(β_k)||_2 = 276.73548299
gamma = 0.00078364164096 ACC = 0.984 PRE = 0.976277372263 REC = 0.994423791822
k = 9 t = 80.9365749359 L(β_k) = 467.333365045 ||∇L(β_k)||_2 = 297.335681259
gamma = 0.00078364164096 ACC = 0.979 PRE = 0.981378026071 REC = 0.979553903346
k = 10 t = 89.217705965 L(β_k) = 449.768322793 ||∇L(β_k)||_2 = 319.530735224
gamma = 0.000470184984576 ACC = 0.984 PRE = 0.979779411765 REC = 0.990706319703
k = 11 t = 96.2231118679 L(β_k) = 425.924544297 ||∇L(β_k)||_2 = 166.721201409
gamma = 0.002176782336 ACC = 0.983 PRE = 0.988742964353 REC = 0.979553903346
k = 12 t = 104.442292929 L(β_k) = 405.994303585 ||∇L(β_k)||_2 = 388.199259096
gamma = 0.000470184984576 ACC = 0.986 PRE = 0.979853479853 REC = 0.994423791822
k = 13 t = 111.281156063 L(β_k) = 374.250964742 ||∇L(β_k)||_2 = 147.167375122
gamma = 0.002176782336 ACC = 0.987 PRE = 0.994350282486 REC = 0.981412639405
k = 14 t = 119.487916946 L(β_k) = 364.103711306 ||∇L(β_k)||_2 = 357.709734875
gamma = 0.000470184984576 ACC = 0.989 PRE = 0.985267034991 REC = 0.994423791822
k = 15 t = 125.968127012 L(β_k) = 335.533358123 ||∇L(β_k)||_2 = 119.375620574
gamma = 0.00362797056 ACC = 0.989 PRE = 0.994371482176 REC = 0.985130111524
k = 16 t = 134.196310997 L(β_k) = 321.40210325 ||∇L(β_k)||_2 = 359.393232945
gamma = 0.000470184984576 ACC = 0.994 PRE = 0.992592592593 REC = 0.996282527881
k = 17 t = 139.345375061 L(β_k) = 290.83776835 ||∇L(β_k)||_2 = 93.2783034072
gamma = 0.01679616 ACC = 0.99 PRE = 0.994382022472 REC = 0.986988847584
```

```
gamma = 0.01679616 ACC = 0.99 PRE = 0.994382022472 REC = 0.986988847584
k = 18 t = 147.14546895 L(β_k) = 238.234800471 ||∇L(β_k)||_2 = 427.837337396
gamma = 0.00078364164096 ACC = 0.997 PRE = 0.994454713494 REC = 1.0
k = 19 t = 154.476336002 L(β_k) = 183.512335128 ||∇L(β_k)||_2 = 104.689949815
gamma = 0.0013060694016 ACC = 0.997 PRE = 0.994454713494 REC = 1.0
Algorithm ran for 20 iterations. Converged: False
Saving trained β in beta
```

logisticRegressison use 63s however ParallelLogisticRegression use 154s which is slower. Thus, using parallel in small data set doesn't improve the speed

4c)

LogisticRegression

```
k = 0 →t = 23.08573699 ⟶L(β_k) = 825.538292047 ⟶||∇L(β_k)||_2 = 583.67949253 →
  ⟶⟶gamma = 0.0060466176 →ACC = 0.862547288777 →PRE = 0.786427145709 →REC = 0.994949494949
k = 1 →t = 50.644258976 →L(β_k) = 235.997770698 ⟶||∇L(β_k)||_2 = 298.527990692 →
  ⟶⟶gamma = 0.00362797056 →ACC = 0.923076923077 →PRE = 0.98833819242 ⟶REC = 0.856060606061
k = 2 →t = 78.1819589138 →L(β_k) = 165.313481914 ⟶||∇L(β_k)||_2 = 170.445788056 →
  ⟶⟶gamma = 0.00362797056 →ACC = 0.958385876419 →PRE = 0.939467312349 →REC = 0.979797979798
k = 3 →t = 104.127202988 →L(β_k) = 127.084846636 ⟶||∇L(β_k)||_2 = 70.1305338894 →
  ⟶⟶gamma = 0.0060466176 →ACC = 0.959646910467 →PRE = 0.971502590674 →REC = 0.94696969697
k = 4 →t = 128.61430788 →L(β_k) = 113.21669691 →||∇L(β_k)||_2 = 58.8631689157 →
  ⟶⟶gamma = 0.010077696 ⟶ACC = 0.954602774275 →PRE = 0.928571428571 →REC = 0.984848484848
k = 5 →t = 154.757153988 →L(β_k) = 102.790003817 ⟶||∇L(β_k)||_2 = 73.7283234092 →
  ⟶⟶gamma = 0.0060466176 →ACC = 0.965952080706 →PRE = 0.971867007673 →REC = 0.959595959596
k = 6 →t = 177.578147888 →L(β_k) = 88.4034890022 ⟶||∇L(β_k)||_2 = 39.9420196745 →
  ⟶⟶gamma = 0.01679616 ⟶ACC = 0.959646910467 →PRE = 0.93961352657 ⟶REC = 0.982323232323
k = 7 →t = 201.926411867 →L(β_k) = 79.6646784348 ⟶||∇L(β_k)||_2 = 58.5179490289 →
  ⟶⟶gamma = 0.010077696 ⟶ACC = 0.962168978562 →PRE = 0.98670212766 ⟶REC = 0.936868686869
k = 8 →t = 226.398449898 →L(β_k) = 69.2890442477 ⟶||∇L(β_k)||_2 = 42.2328868211 →
  ⟶⟶gamma = 0.010077696 ⟶ACC = 0.970996216898 →PRE = 0.965087281796 →REC = 0.977272727273
k = 9 →t = 247.73533988 →L(β_k) = 61.5010061918 ⟶||∇L(β_k)||_2 = 27.2594855791 →
  ⟶⟶gamma = 0.0279936 →ACC = 0.960907944515 →PRE = 0.99727520436 ⟶REC = 0.924242424242
k = 10 ⟶t = 272.140666962 →L(β_k) = 57.1027717799 ⟶||∇L(β_k)||_2 = 48.0178834958 →
  ⟶⟶gamma = 0.010077696 ⟶ACC = 0.972257250946 →PRE = 0.9675 →REC = 0.977272727273
k = 11 ⟶t = 291.771583796 →L(β_k) = 46.8764140886 ⟶||∇L(β_k)||_2 = 17.4834056128 →
  ⟶⟶gamma = 0.046656 →ACC = 0.965952080706 →PRE = 0.994638069705 →REC = 0.936868686869
k = 12 ⟶t = 314.727641821 →L(β_k) = 40.3072811261 ⟶||∇L(β_k)||_2 = 28.17195718 ⟶
  ⟶⟶gamma = 0.01679616 ⟶ACC = 0.970996216898 →PRE = 0.967418546366 →REC = 0.974747474747
k = 13 ⟶t = 334.465430975 →L(β_k) = 34.7375225095 ⟶||∇L(β_k)||_2 = 13.9589882034 →
  ⟶⟶gamma = 0.046656 →ACC = 0.974779319042 →PRE = 0.994736842105 →REC = 0.954545454545
k = 14 ⟶t = 355.685072899 →L(β_k) = 31.1045706768 ⟶||∇L(β_k)||_2 = 19.5463166714 →
  ⟶⟶gamma = 0.0279936 →ACC = 0.96973518285 ⟶PRE = 0.965 ⟶REC = 0.974747474747
k = 15 ⟶t = 375.305197001 →L(β_k) = 27.5298505433 ⟶||∇L(β_k)||_2 = 13.6770946229 →
  ⟶⟶gamma = 0.046656 →ACC = 0.974779319042 →PRE = 0.994736842105 →REC = 0.954545454545
k = 16 ⟶t = 396.650461912 →L(β_k) = 25.1736211979 ⟶||∇L(β_k)||_2 = 16.2624151488 →
  ⟶⟶gamma = 0.0279936 →ACC = 0.973518284994 →PRE = 0.974683544304 →REC = 0.972222222222
k = 17 ⟶t = 413.194274902 →L(β_k) = 21.8901100291 ⟶||∇L(β_k)||_2 = 8.24527473729 →
  ⟶⟶gamma = 0.1296 ⟶ACC = 0.965952080706 →PRE = 0.994638069705 →REC = 0.936868686869
k = 18 ⟶t = 434.416885853 →L(β_k) = 20.0810908656 ⟶||∇L(β_k)||_2 = 19.118647409 →
  ⟶⟶gamma = 0.0279936 →ACC = 0.973518284994 →PRE = 0.982005141388 →REC = 0.964646464646
k = 19 ⟶t = 444.548135996 →L(β_k) = 15.4223747302 ⟶||∇L(β_k)||_2 = 4.96431570345 →
  ⟶⟶gamma = 1.0 ⟶ACC = 0.947036569987 →PRE = 0.997191011236 →REC = 0.896464646465
```

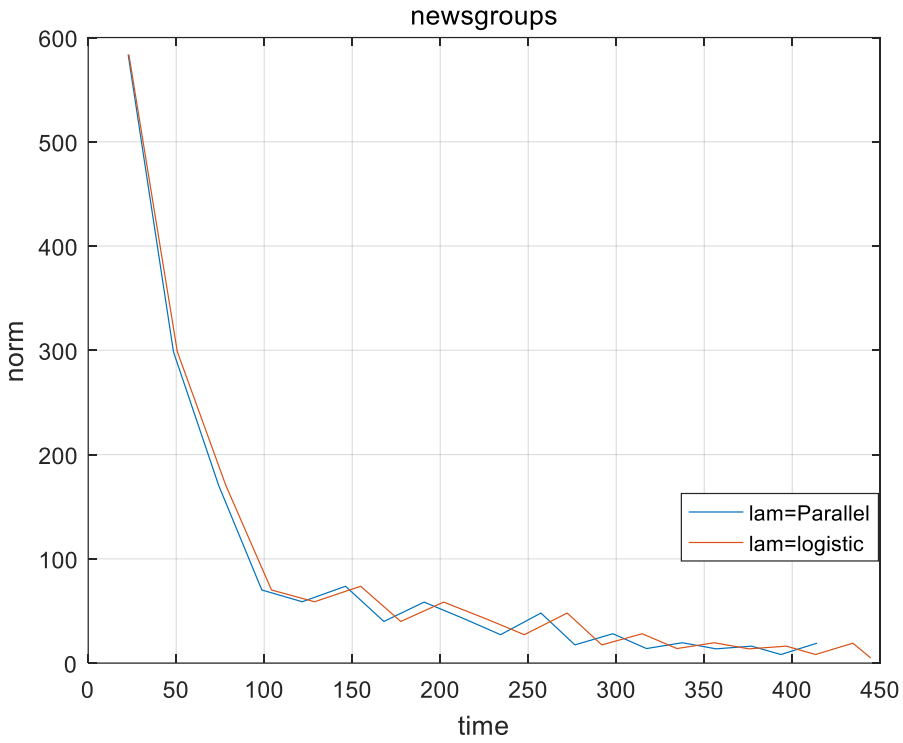
ParallelLogisticRegression

```
k = 0 →t = 22.574409008 →L(β_k) = 825.538292047 ⟶||∇L(β_k)||_2 = 583.67949253 →
 ⟶⟶gamma = 0.0060466176 →ACC = 0.863808322825 →PRE = 0.788 ⟶REC = 0.994949494949
k = 1 →t = 48.6189448833 →L(β_k) = 235.997770698 ⟶||∇L(β_k)||_2 = 298.527990692 →
 ⟶⟶gamma = 0.00362797056 →ACC = 0.923076923077 →PRE = 0.98833819242 ⟶REC = 0.856060606061
k = 2 →t = 74.6104660034 →L(β_k) = 165.313481914 ⟶||∇L(β_k)||_2 = 170.445788056 →
 ⟶⟶gamma = 0.00362797056 →ACC = 0.958385876419 →PRE = 0.939467312349 →REC = 0.979797979798
k = 3 →t = 99.3507430553 →L(β_k) = 127.084846636 ⟶||∇L(β_k)||_2 = 70.1305338894 →
 ⟶⟶gamma = 0.0060466176 →ACC = 0.959646910467 →PRE = 0.971502590674 →REC = 0.94696969697
k = 4 →t = 122.550463915 →L(β_k) = 113.21669691 →||∇L(β_k)||_2 = 58.8631689157 →
 ⟶⟶gamma = 0.010077696 ⟶ACC = 0.954602774275 →PRE = 0.928571428571 →REC = 0.984848484848
k = 5 →t = 146.998682976 →L(β_k) = 102.790003817 ⟶||∇L(β_k)||_2 = 73.7283234092 →
 ⟶⟶gamma = 0.0060466176 →ACC = 0.965952080706 →PRE = 0.971867007673 →REC = 0.959595959596
k = 6 →t = 169.122973919 →L(β_k) = 88.4034890022 ⟶||∇L(β_k)||_2 = 39.9420196745 →
 ⟶⟶gamma = 0.01679616 ⟶ACC = 0.959646910467 →PRE = 0.93961352657 ⟶REC = 0.982323232323
k = 7 →t = 192.062329054 →L(β_k) = 79.6646784348 ⟶||∇L(β_k)||_2 = 58.5179490289 →
 ⟶⟶gamma = 0.010077696 ⟶ACC = 0.962168978562 →PRE = 0.98670212766 ⟶REC = 0.936868686869
k = 8 →t = 215.216693878 →L(β_k) = 69.2890442477 ⟶||∇L(β_k)||_2 = 42.2328868211 →
 ⟶⟶gamma = 0.010077696 ⟶ACC = 0.970996216898 →PRE = 0.965087281796 →REC = 0.977272727273
k = 9 →t = 235.935568094 →L(β_k) = 61.5010061918 ⟶||∇L(β_k)||_2 = 27.2594855791 →
 ⟶⟶gamma = 0.0279936 →ACC = 0.960907944515 →PRE = 0.99727520436 ⟶REC = 0.924242424242
k = 10 ⟶t = 259.068108082 →L(β_k) = 57.1027717799 ⟶||∇L(β_k)||_2 = 48.0178834958 →
 ⟶⟶gamma = 0.010077696 ⟶ACC = 0.972257250946 →PRE = 0.9675 →REC = 0.977272727273
k = 11 ⟶t = 278.502521992 →L(β_k) = 46.8764140886 ⟶||∇L(β_k)||_2 = 17.4834056128 →
 ⟶⟶gamma = 0.046656 →ACC = 0.965952080706 →PRE = 0.994638069705 →REC = 0.936868686869
k = 12 ⟶t = 300.533195972 →L(β_k) = 40.3072811261 ⟶||∇L(β_k)||_2 = 28.17195718 ⟶
 ⟶⟶gamma = 0.01679616 ⟶ACC = 0.970996216898 →PRE = 0.967418546366 →REC = 0.974747474747
k = 13 ⟶t = 319.591930866 →L(β_k) = 34.7375225095 ⟶||∇L(β_k)||_2 = 13.9589882034 →
 ⟶⟶gamma = 0.046656 →ACC = 0.974779319042 →PRE = 0.994736842105 →REC = 0.954545454545
k = 14 ⟶t = 340.013494015 →L(β_k) = 31.1045706768 ⟶||∇L(β_k)||_2 = 19.5463166714 →
 ⟶⟶gamma = 0.0279936 →ACC = 0.96973518285 ⟶PRE = 0.965 ⟶REC = 0.974747474747
k = 15 ⟶t = 359.024389029 →L(β_k) = 27.5298505433 ⟶||∇L(β_k)||_2 = 13.6770946229 →
 ⟶⟶gamma = 0.046656 →ACC = 0.974779319042 →PRE = 0.994736842105 →REC = 0.954545454545
k = 16 ⟶t = 379.424942017 →L(β_k) = 25.1736211979 ⟶||∇L(β_k)||_2 = 16.2624151488 →
 ⟶⟶gamma = 0.0279936 →ACC = 0.973518284994 →PRE = 0.974683544304 →REC = 0.972222222222
k = 17 ⟶t = 396.005378962 →L(β_k) = 21.8901100291 ⟶||∇L(β_k)||_2 = 8.24527473729 →
 ⟶⟶gamma = 0.1296 ⟶ACC = 0.965952080706 →PRE = 0.994638069705 →REC = 0.936868686869
k = 18 ⟶t = 416.722774029 →L(β_k) = 20.0810908656 ⟶||∇L(β_k)||_2 = 19.118647409 →
 ⟶⟶gamma = 0.0279936 →ACC = 0.973518284994 →PRE = 0.982005141388 →REC = 0.964646464646
k = 19 ⟶t = 428.148526907 →L(β_k) = 15.4223747302 ⟶||∇L(β_k)||_2 = 4.96431570345 →
 ⟶⟶gamma = 1.0 ⟶ACC = 0.947036569987 →PRE = 0.997191011236 →REC = 0.896464646465
Algorithm ran for 20 iterations. Converged: False
```

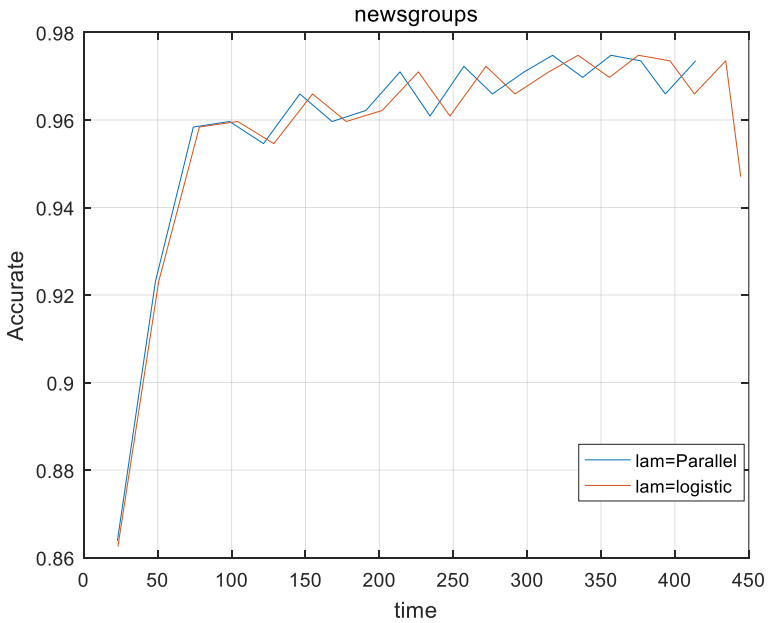
logisticRegressison use 444s however ParallelLogisticRegression use 428 which is faster. Thus, using parallel in large data set can improve the speed

Q5

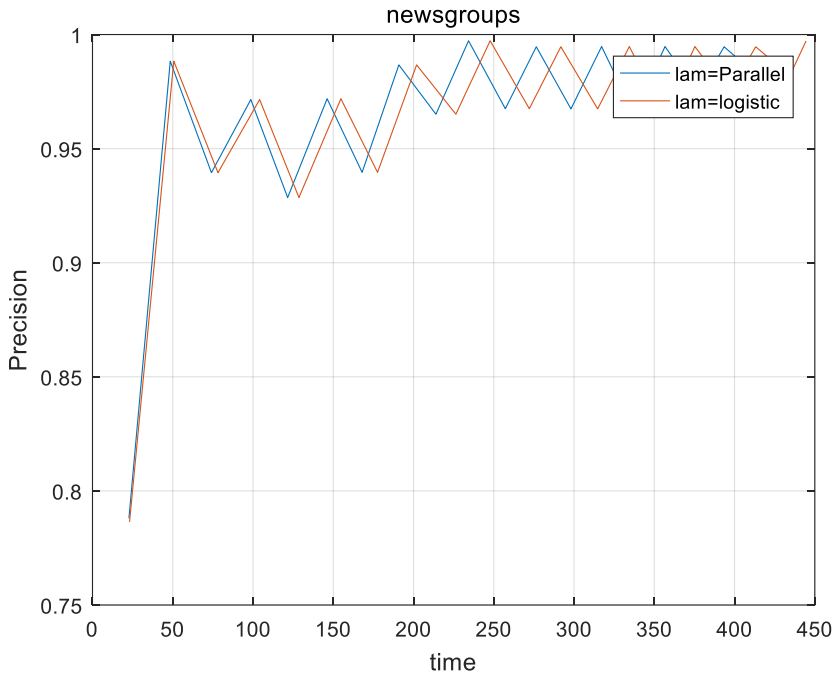
5a)



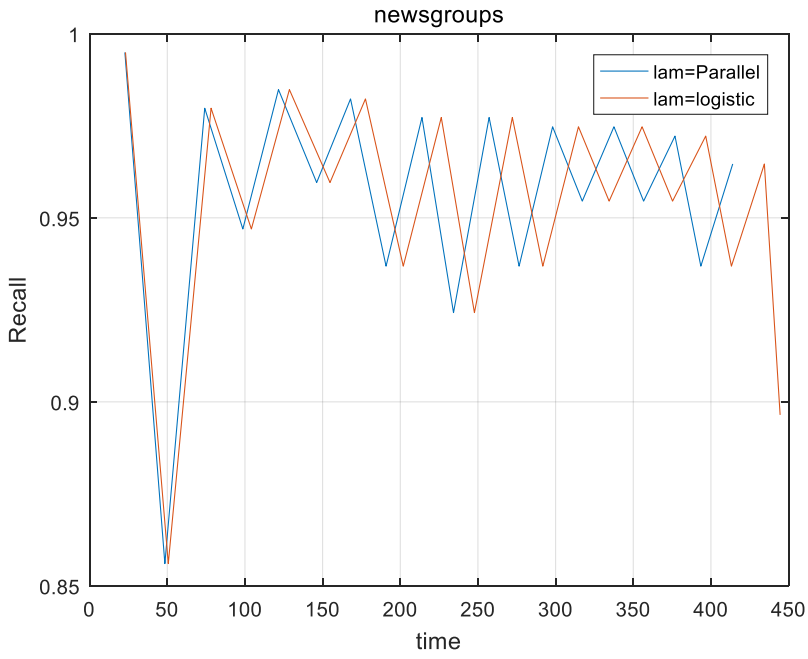
Gradient norm 2



Accurate

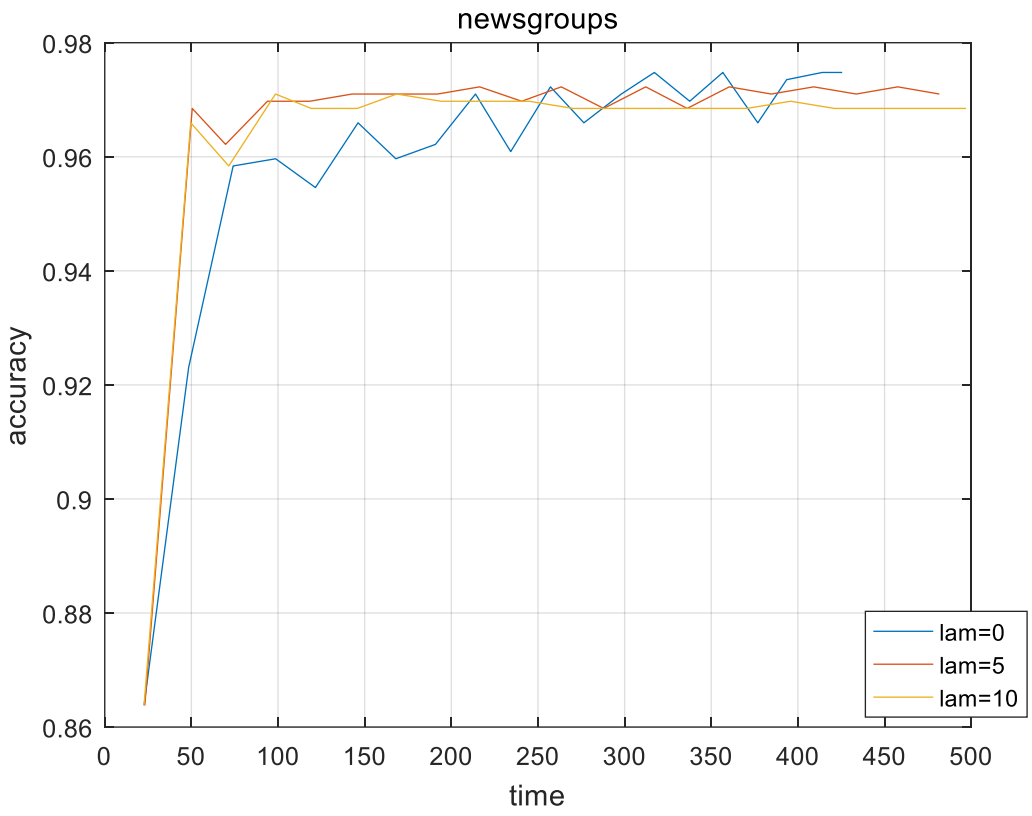


Precision



Recall

5b)



c) lambda = 0 has the greatest result

10 Most Negative

' basebal'	**-2.56823700000000**
' game'	**-2.10310900000000**
' player'	**-1.94976900000000**
' team'	**-1.91179500000000**
' yanke'	**-1.60964700000000**
' win'	**-1.45783200000000**
' philli'	**-1.38217800000000**
' plai'	**-1.38204400000000**
' stat'	**-1.30981000000000**
' pitch'	**-1.24824500000000**

10 Most Positive

' inform'	**1.25335100000000**
' doctor'	**1.18740100000000**
' effect'	**1.14800100000000**
' treatment'	**1.03699300000000**
' diseas'	**1.03653500000000**
' medic'	**0.917249000000000**
' treat'	**0.884590000000000**
' problem'	**0.878204000000000**
' health'	**0.808931000000000**
' peopl'	**0.778835000000000**