HW 3

Xianlong Zhang

Question 1:

(a)

The class SparseVector is similar to a standard dictionary, but it would return nothing if a value of a key is None, however, a standard dictionary would return None.

```
>>> a = {1:1,2:None}
>>> b = sv.SparseVector(a)
>>> b[2]
>>>
```

(b)

From the following examples, we can see that both standard dictionary and SparseVector have cmp() and len() methods, but SparseVector also has dot() and __add__() methods, which are not presented in dictionary.

```
>>> import SparseVector as sv
>>> a = {1 : 1, 2 : 2}
>>> b = {3: 3, 4: 4}
>>> c = sv.SparseVector(a)
>>> d = sv.SparseVector(b)
>>> len(a)
2
>>> len(c)
2
>>> cmp(a, b)
-1
>>> cmp(c, d)
-1
>>> c.dot(d)
0
>>> a.dot(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'dict' object has no attribute 'dot'
>>> a.__add__(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'dict' object has no attribute '__add__'
>>> c.__add__(d)
{1: 1, 2: 2, 3: 3, 4: 4}
```

(c)

```
>>> import SparseVector as sv
>>> a = {1: 1, 2: 2}
>>> b = {1: 3, 2: 4}
>>> c = sv.SparseVector(a)
>>> d = sv.SparseVector(b)
>>> c.__add__(d)
{1: 4, 2: 6}
>>> c.dot(d)
11
>>> c.__mul__(2)
{1: 2, 2: 4}
>>> c.__rmul__(2)
{1: 2, 2: 4}
```

(d)

The modified code is as following:

```
def norm(self, p = 2):
    val_list = self.values()
    p_norm = np.linalg.norm(val_list, p)
    return p_norm
```

and for p = 1, 2, 3, the outputs are as following:

```
>>> import SparseVector as sv
>>> a = {'a': 3, 'b': 4}
>>> b = sv.SparseVector(a)
>>> b.norm(1)
7.0
>>> b.norm(2)
5.0
>>> b.norm(3)
4.4979414452754147
>>> b.norm()
5.0
```

Question 2:

(a)

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

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

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

Since $e^{y\beta^T x} \geq 0, \nabla^2 l \geq 0$ as well, so $l(\beta; x_i, y_i)$ is a convex function.

(b)

As shown in (a), $\nabla^2 l = \frac{-yx*e^{-y\beta^T x}}{1+e^{-y\beta^T x}} = \frac{-yx}{1+e^{y\beta^T x}}$.

(c)

Since both $\nabla^2 l$ and ∇l are associated with $y\beta^T x = y\sum_{i=1}^{d}\beta_i x_i$, once $x_j = 0$, $y\beta^T x = y\sum_{\substack{i=1 \\ i\neq j}}^{d}\beta_i x_i$, then they don't depend on β_i for β_i multiply 0.

(d)

Since $l(\beta: x_i, y_i)$ is convex, $\sum_{i=1}^{n} l(\beta: x_i, y_i)$ is also convex. Let $f(\beta) = \lambda||\beta||^2$, then $\nabla l = 2\lambda\beta$ and $\nabla^2 f = 2\lambda$.So if $\lambda \geq 0, L(\beta) = \sum_{i=1}^{n} l(\beta: x_i, y_i) + \lambda||\beta||^2$ is convex.

(e)

From (d) we know $\nabla^2 f = 2\lambda$, only when $\nabla^2 f > 0$, can L be a strictly convex function, so when $\lambda = 0$, L is not strictly convex.

(f)

From (d) we know $\nabla^2 f = 2\lambda$, only when $\nabla^2 f \geq mI$, where $m > 0$, can L be a convex function, so when $\lambda > 0$, $\nabla^2 f \geq \lambda I$, then L is strongly convex.

Question 3:

(a)

```
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)))

-------------
```

```
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 * ((-1.0 * y) / (1.0 + np.exp(1.0 * 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 logist
ic 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 a
nd y is a binary value
            - beta: a sparse vector β
            - lam: the regularization parameter λ
    """
    loss = SparseVector({})
    for (x, y) in data:
        loss = gradLogisticLoss(beta, x, y) + loss
    return loss + 2.0 * lam * beta
```

(b)

```
 def test(data,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 fol
 lowing 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 pa
 rameter 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: 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

        The return values are
            - ACC, PRE, REC

    """
    total_points = [(beta.dot(x), y) for (x, y) in data]
    P = N = TP = FP = TN = FN = 0.0
    for (x, y) in total_points:
        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))
        FN += ((x < 0) and (float(y) == 1.0))
    ACC = 1.0 * (TP + TN) / (P + N)
    PRE = 1.0 * TP / (TP + FP)
    REC = 1.0 * TP / (TP + FN)
    return ACC, PRE, REC
```

(c)

When $\lambda = 0$,

```
k =  0  t =  4.37429308891      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 =  9.72321820259      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 =  13.5640070438      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 =  18.3814451694      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 =  23.1850612164      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 =  27.7522661686      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 =  32.5636901855      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 =  37.0968980789      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 =  41.6409590244      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 =  46.1966061592      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 =  50.9850490093      L(β_k) =  449.768322793         ||∇L(β_k)||_2 =  319.530735224  gamma =  0.000470184984576      ACC =  0.984       PRE =  0.9797
79411765   REC =  0.990706319703
k =  11         t =  55.0649831295      L(β_k) =  425.924544297         ||∇L(β_k)||_2 =  166.721201409  gamma =  0.002176782336         ACC =  0.983       PRE =  0.9887
42964353   REC =  0.979553903346
k =  12         t =  59.8572990894      L(β_k) =  405.994303585         ||∇L(β_k)||_2 =  388.199259096  gamma =  0.000470184984576      ACC =  0.986       PRE =  0.9798
53479853   REC =  0.994423791822
k =  13         t =  63.9570782185      L(β_k) =  374.250964742         ||∇L(β_k)||_2 =  147.167375122  gamma =  0.002176782336         ACC =  0.987       PRE =  0.9943
50282486   REC =  0.981412639405
k =  14         t =  68.7469301224      L(β_k) =  364.103711306         ||∇L(β_k)||_2 =  357.709734875  gamma =  0.000470184984576      ACC =  0.989       PRE =  0.9852
67034991   REC =  0.994423791822
k =  15         t =  72.5881271362      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 =  77.3849141598      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 =  80.5235280991      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 =  85.0751681328      L(β_k) =  238.234800471         ||∇L(β_k)||_2 =  427.837337396  gamma =  0.00078364164096       ACC =  0.997       PRE =  0.9944
54713494   REC =  1.0
k =  19         t =  89.3996372223      L(β_k) =  183.512335128         ||∇L(β_k)||_2 =  104.689949815  gamma =  0.0013060694016        ACC =  0.997       PRE =  0.9944
54713494   REC =  1.0
Algorithm ran for 20 iterations. Converged: False
Saving trained β in beta0
```

When $\lambda = 5$,

```
k =  0  t =  4.36257505417      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 =  9.67115688324      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 =  13.4962880611      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 =  18.3148829937      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 =  23.1265828609      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 =  27.921546936       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 =  32.4437570572      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 =  37.2230899334      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 =  41.5286319256      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 =  46.3018610477      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 =  50.6083328724      L(β_k) =  564.819571171         ||∇L(β_k)||_2 =  169.651898158  gamma =  0.0013060694016        ACC =  0.984    PRE =  0.9762
77372263   REC =  0.994423791822
k =  11         t =  55.3971760273      L(β_k) =  555.998065612         ||∇L(β_k)||_2 =  306.79989283   gamma =  0.000470184984576      ACC =  0.982    PRE =  0.9797
04797048   REC =  0.986988847584
k =  12         t =  59.6776349545      L(β_k) =  536.412663316         ||∇L(β_k)||_2 =  141.992440299  gamma =  0.0013060694016        ACC =  0.984    PRE =  0.9762
77372263   REC =  0.994423791822
k =  13         t =  64.4529459476      L(β_k) =  528.459592179         ||∇L(β_k)||_2 =  244.661438016  gamma =  0.000470184984576      ACC =  0.985    PRE =  0.9815
83793738   REC =  0.990706319703
k =  14         t =  68.7462689877      L(β_k) =  515.159794983         ||∇L(β_k)||_2 =  115.957447378  gamma =  0.0013060694016        ACC =  0.986    PRE =  0.9798
53479853   REC =  0.994423791822
k =  15         t =  73.2651200294      L(β_k) =  506.701683219         ||∇L(β_k)||_2 =  177.001664477  gamma =  0.00078364164096       ACC =  0.989    PRE =  0.9888
68274583   REC =  0.990706319703
k =  16         t =  78.0229840279      L(β_k) =  501.145796447         ||∇L(β_k)||_2 =  186.787072656  gamma =  0.000470184984576      ACC =  0.988    PRE =  0.9834
55882353   REC =  0.994423791822
k =  17         t =  82.1310420036      L(β_k) =  492.950591036         ||∇L(β_k)||_2 =  93.6448314909  gamma =  0.002176782336         ACC =  0.991    PRE =  0.9925
51210428   REC =  0.990706319703
k =  18         t =  86.9248418808      L(β_k) =  485.712728307         ||∇L(β_k)||_2 =  209.942518822  gamma =  0.000470184984576      ACC =  0.991    PRE =  0.9871
08655617        REC =  0.996282527881
k =  19         t =  91.0333390236      L(β_k) =  476.11914913  ||∇L(β_k)||_2 =  83.0571978465  gamma =  0.002176782336         ACC =  0.994    PRE =  0.994423791822
        REC =  0.994423791822
Algorithm ran for 20 iterations. Converged: False
Saving trained β in beta5
```

When $\lambda = 10$,

```
k =  0  t =  4.44599699974      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 =  9.83867001534      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 =  13.7310819626      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 =  18.859153986       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 =  22.9906489849      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 =  27.8735570908      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 =  32.717592001       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 =  37.3243839741      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 =  42.1683249474      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 =  46.5427920818      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 =  51.3788881302      L(β_k) =  659.584652805         ||∇L(β_k)||_2 =  232.290087336  gamma =  0.000470184984576      ACC =  0.986    PRE =  0.9798
53479853        REC =  0.994423791822
k =  11         t =  55.9662950039      L(β_k) =  649.522167599         ||∇L(β_k)||_2 =  130.536027197  gamma =  0.00078364164096       ACC =  0.986    PRE =  0.9833
94833948        REC =  0.990706319703
k =  12         t =  60.8012759686      L(β_k) =  644.278385475         ||∇L(β_k)||_2 =  152.946794124  gamma =  0.000470184984576      ACC =  0.986    PRE =  0.9798
53479853        REC =  0.994423791822
k =  13         t =  65.1470649242      L(β_k) =  638.714924259         ||∇L(β_k)||_2 =  98.4101786566  gamma =  0.0013060694016        ACC =  0.988    PRE =  0.9870
37037037        REC =  0.990706319703
k =  14         t =  69.979694128       L(β_k) =  633.804194941         ||∇L(β_k)||_2 =  173.008101859  gamma =  0.000470184984576      ACC =  0.986    PRE =  0.9798
53479853        REC =  0.994423791822
k =  15         t =  74.5807199478      L(β_k) =  627.906060556         ||∇L(β_k)||_2 =  95.9654162555  gamma =  0.00078364164096       ACC =  0.99     PRE =  0.9870
84870849        REC =  0.994423791822
k =  16         t =  79.19500494        L(β_k) =  624.494970313         ||∇L(β_k)||_2 =  108.111406092  gamma =  0.00078364164096       ACC =  0.988    PRE =  0.9816
84981685        REC =  0.996282527881
k =  17         t =  84.0177760124      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 =  88.3618330956      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 =  93.1919789314      L(β_k) =  615.292058287         ||∇L(β_k)||_2 =  126.552208232  gamma =  0.000470184984576      ACC =  0.991    PRE =  0.9889
09426987        REC =  0.994423791822
Algorithm ran for 20 iterations. Converged: False
Saving trained β in beta10
```

(d)

Recall is TP/(TP + FN) whereas precision is TP/(TP+FP). Recall means the fraction of relevant instances retrieved over total number of relevant instances, and precision means the fraction of relevant instances retrieved among all relevant instances. In this case, since the mushroom maybe poisonous, and harmful to healthy, we prefer high precision and low recall.

Question 4:

(a)

The code I modified is as following:

```
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 + 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

    """
    total_points = dataRDD.map(lambda (x, y) : (beta.dot(x), y))
    positive = total_points.filter(lambda (x, y) : x > 0)
    negative = total_points.filter(lambda (x, y) : x <= 0)
    P = positive.count()
    N = negative.count()
    TP = positive.filter(lambda (x, y) : y == 1.0).count()
    FP = positive.filter(lambda (x, y) : y == -1.0).count()
    TN = negative.filter(lambda (x, y) : y == -1.0).count()
    FN = negative.filter(lambda (x, y) : y == 1.0).count()
    ACC = (1.0 * (TP +TN)) / (P + N)
    PRE = (1.0 * TP) / (TP + FP)
    REC = (1.0 * TP) / (TP + FN)
    return ACC, PRE, REC


def train(dataRDD,beta_0,lam,max_iter,eps,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 '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 '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 termi
nates.')
    parser.add_argument('--N',type=int,default=2,help='Level of parallelism')

    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(appName='Parallel Logistic Regression')
    if not args.verbose :
        sc.setLogLevel("ERROR")


    print 'Reading training data from',args.traindata
    traindata = readDataRDD(args.traindata, sc)
    traindata = traindata.repartition(args.N).cache()
    print 'Read',traindata.count(),'data points with',getAllFeaturesRDD(traindata).count(),'features in total'


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

    beta0 = SparseVector({})

    print '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)
    print 'Algorithm ran for',k,'iterations. Converged:',gradNorm<args.eps
    print 'Saving trained β in',args.beta
    writeBeta(args.beta,beta)
```

(b)

From the following output, compared with LogisticRegression.py, when executed with $\lambda = 0$ over the mushrooms train and test datasets for at most 20 iterations, the results are the same, but the ParallelLogisticRegression.py is faster.

```
k =  0  t =  3.68310117722      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 =  7.81329798698      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 =  10.8044121265      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 =  14.514521122       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 =  18.2041420937      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 =  21.6951191425      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 =  25.3614721298      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 =  28.8576259613      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 =  32.3308141232      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 =  35.7955350876      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 =  39.4355700016      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 =  42.5344400406      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 =  46.1630301476      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 =  49.3060920238      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 =  52.9189350605      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 =  55.8371419907      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 =  59.4352481365      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 =  61.8547711372      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 =  65.3052890301      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 =  68.5802130699      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 beta4.0
```

(c)

Fig 1. logisticRegression

```
k =  0  t =  30.1513941288      L(β_k) =  825.538292047         ||∇L(β_k)||_2 =  583.67949253   gamma =  0.0060466176   ACC =  0.86254728877
7PRE =  0.786427145709    REC =  0.994949494949
LogisticRegression.py:92: RuntimeWarning: overflow encountered in exp
  return np.log(1.0 + np.exp(-float(y) * beta.dot(x)))
k =  1  t =  66.7890150547      L(β_k) =  235.997770698         ||∇L(β_k)||_2 =  298.527990692  gamma =  0.00362797056  ACC =  0.92307692307
7PRE =  0.98833819242     REC =  0.856060606061
k =  2  t =  103.07351613       L(β_k) =  165.313481914         ||∇L(β_k)||_2 =  170.445788056  gamma =  0.00362797056  ACC =  0.95838587641
9PRE =  0.939467312349    REC =  0.979797979798
k =  3  t =  137.519681215      L(β_k) =  127.084846636         ||∇L(β_k)||_2 =  70.1305338894  gamma =  0.0060466176   ACC =  0.95964691046
7PRE =  0.971502590674    REC =  0.94696969697
k =  4  t =  169.893415213      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 =  204.438184023      L(β_k) =  102.790003817         ||∇L(β_k)||_2 =  73.7283234092  gamma =  0.0060466176   ACC =  0.96595208070
6PRE =  0.971867007673    REC =  0.959595959596
k =  6  t =  234.682845116      L(β_k) =  88.4034890022         ||∇L(β_k)||_2 =  39.9420196745  gamma =  0.01679616     ACC =  0.95964691046
7PRE =  0.93961352657     REC =  0.982323232323
k =  7  t =  267.08089304       L(β_k) =  79.6646784348         ||∇L(β_k)||_2 =  58.5179490289  gamma =  0.010077696    ACC =  0.96216897856
2PRE =  0.98670212766     REC =  0.936868686869
k =  8  t =  299.192389011      L(β_k) =  69.2890442477         ||∇L(β_k)||_2 =  42.2328868211  gamma =  0.010077696    ACC =  0.97099621689
8PRE =  0.965087281796    REC =  0.977272727273
k =  9  t =  327.224176168      L(β_k) =  61.5010061918         ||∇L(β_k)||_2 =  27.2594855791  gamma =  0.0279936      ACC =  0.96090794451
5PRE =  0.99727520436     REC =  0.924242424242
k =  10         t =  359.548557043      L(β_k) =  57.1027717799         ||∇L(β_k)||_2 =  48.0178834958  gamma =  0.010077696    ACC =  0.972
257250946         PRE =  0.9675   REC =  0.977272727273
k =  11         t =  385.413913012      L(β_k) =  46.8764140886         ||∇L(β_k)||_2 =  17.4834056128  gamma =  0.046656       ACC =  0.965
952080706         PRE =  0.994638069705   REC =  0.936868686869
k =  12         t =  415.605792046      L(β_k) =  40.3072811261         ||∇L(β_k)||_2 =  28.17195718    gamma =  0.01679616     ACC =  0.970
996216898         PRE =  0.967418546366   REC =  0.974747474747
k =  13         t =  441.530544996      L(β_k) =  34.7375225095         ||∇L(β_k)||_2 =  13.9589882034  gamma =  0.046656       ACC =  0.974
779319042         PRE =  0.994736842105   REC =  0.954545454545
k =  14         t =  469.389839172      L(β_k) =  31.1045706768         ||∇L(β_k)||_2 =  19.5463166714  gamma =  0.0279936      ACC =  0.969
73518285  PRE =  0.965    REC =  0.974747474747
k =  15         t =  495.239914179      L(β_k) =  27.5298505433         ||∇L(β_k)||_2 =  13.6770946229  gamma =  0.046656       ACC =  0.974
779319042         PRE =  0.994736842105   REC =  0.954545454545
k =  16         t =  523.283146143      L(β_k) =  25.1736211979         ||∇L(β_k)||_2 =  16.2624151488  gamma =  0.0279936      ACC =  0.973
518284994         PRE =  0.974683544304   REC =  0.972222222222
k =  17         t =  544.935036182      L(β_k) =  21.8901100291         ||∇L(β_k)||_2 =  8.24527473729  gamma =  0.1296         ACC =  0.965
952080706         PRE =  0.994638069705   REC =  0.936868686869
k =  18         t =  572.977116108      L(β_k) =  20.0810908656         ||∇L(β_k)||_2 =  19.118647409   gamma =  0.0279936      ACC =  0.973
518284994         PRE =  0.982005141388   REC =  0.964646464646
k =  19         t =  586.044164181      L(β_k) =  15.4223747302         ||∇L(β_k)||_2 =  4.96431570345  gamma =  1.0    ACC =  0.94703656998
7PRE =  0.997191011236    REC =  0.896464646465
Algorithm ran for 20 iterations. Converged: False
Saving trained β in beta0
```

Fig 2. ParallelLogisticRegression

```
k =  0  t =  27.665102005       L(β_k) =  825.538292047         ||∇L(β_k)||_2 =  583.67949253   gamma =  0.0060466176   ACC =  0.862547288777
      PRE =  0.786427145709   REC =  0.994949494949
LogisticRegression.py:92: RuntimeWarning: overflow encountered in exp
  return np.log(1.0 + np.exp(-float(y) * beta.dot(x)))
LogisticRegression.py:92: RuntimeWarning: overflow encountered in exp
  return np.log(1.0 + np.exp(-float(y) * beta.dot(x)))
k =  1  t =  59.939260006       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 =  92.2970509529      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 =  123.066473007      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 =  152.172796011      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 =  182.914736986      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 =  210.379126072      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 =  239.543246984      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 =  269.162065029      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 =  295.07715106       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 =  324.147171021      L(β_k) =  57.1027717799         ||∇L(β_k)||_2 =  48.0178834958  gamma =  0.010077696    ACC =  0.97225
7250946       PRE =  0.9675   REC =  0.977272727273
k =  11         t =  348.490487099      L(β_k) =  46.8764140886         ||∇L(β_k)||_2 =  17.4834056128  gamma =  0.046656       ACC =  0.96595
2080706       PRE =  0.994638069705   REC =  0.936868686869
k =  12         t =  376.012730122      L(β_k) =  40.3072811261         ||∇L(β_k)||_2 =  28.17195718    gamma =  0.01679616     ACC =  0.97099
6216898       PRE =  0.967418546366   REC =  0.974747474747
k =  13         t =  400.318277121      L(β_k) =  34.7375225095         ||∇L(β_k)||_2 =  13.9589882034  gamma =  0.046656       ACC =  0.97477
9319042       PRE =  0.994736842105   REC =  0.954545454545
k =  14         t =  426.219958067      L(β_k) =  31.1045706768         ||∇L(β_k)||_2 =  19.5463166714  gamma =  0.0279936      ACC =  0.96973
518285        PRE =  0.965    REC =  0.974747474747
k =  15         t =  450.453726053      L(β_k) =  27.5298505433         ||∇L(β_k)||_2 =  13.6770946229  gamma =  0.046656       ACC =  0.97477
9319042       PRE =  0.994736842105   REC =  0.954545454545
k =  16         t =  476.266033173      L(β_k) =  25.1736211979         ||∇L(β_k)||_2 =  16.2624151488  gamma =  0.0279936      ACC =  0.97351
8284994       PRE =  0.974683544304   REC =  0.972222222222
k =  17         t =  497.339781046      L(β_k) =  21.8901100291         ||∇L(β_k)||_2 =  8.24527473729  gamma =  0.1296         ACC =  0.96595
2080706       PRE =  0.994638069705   REC =  0.936868686869
k =  18         t =  523.239076138      L(β_k) =  20.0810908656         ||∇L(β_k)||_2 =  19.118647409   gamma =  0.0279936      ACC =  0.97351
8284994       PRE =  0.982005141388   REC =  0.964646464646
k =  19         t =  537.884590149      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
```

The output of logisticRegression.py and ParallelLogisticRegression.py is shown in Fig 1, and Fig 2, respectively, and from them we can conclude that the outputs are the same, but ParallelLogisticRegression.py is faster.

Question 5:

(a)

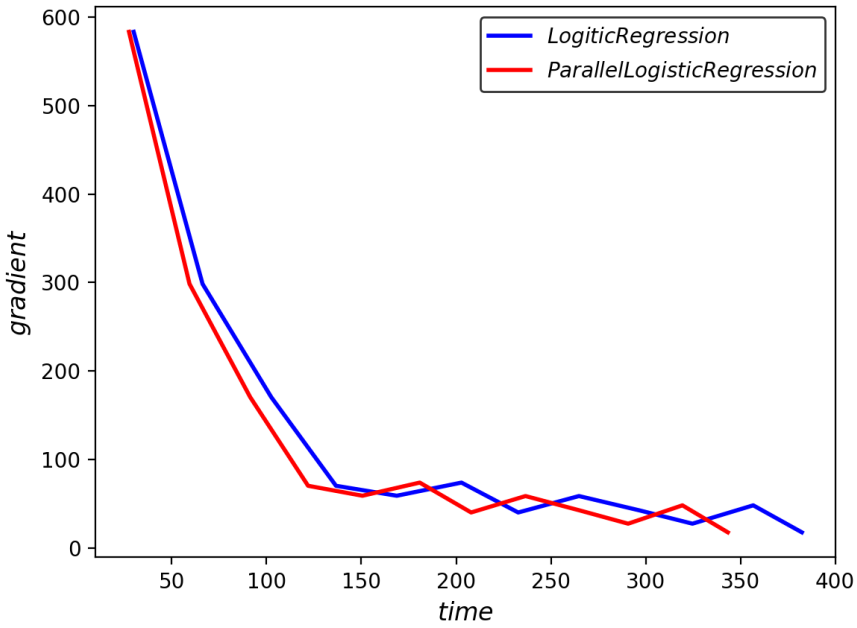

Figure 3 Comparison on gradient

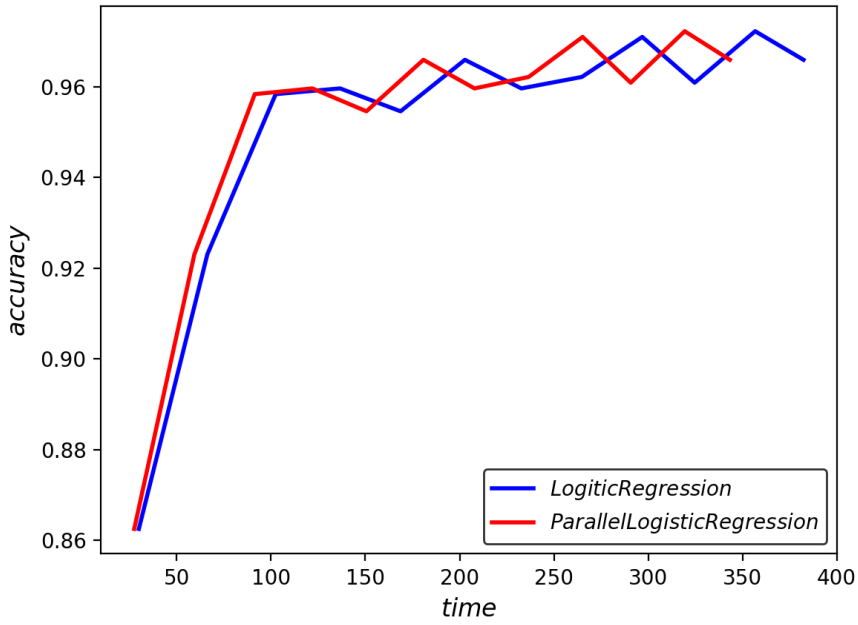

Figure 4 Comparison on accuracy

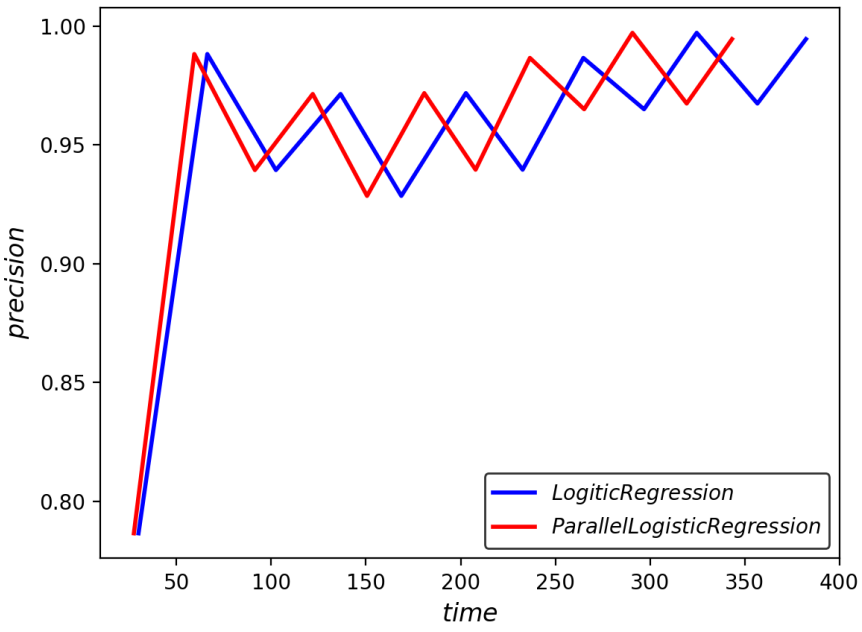

Figure 5 Comparison on precision

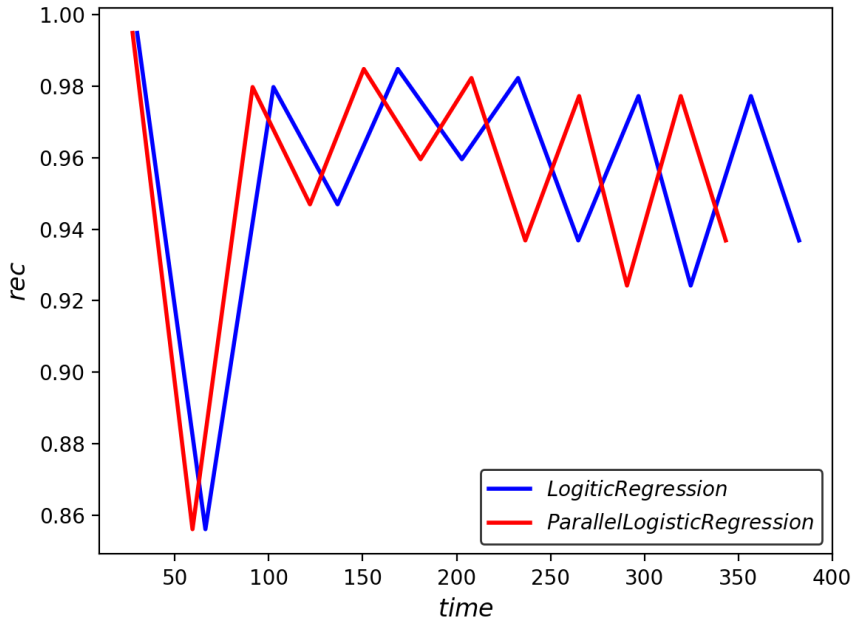

Figure 6 Comparison on recall

(b)

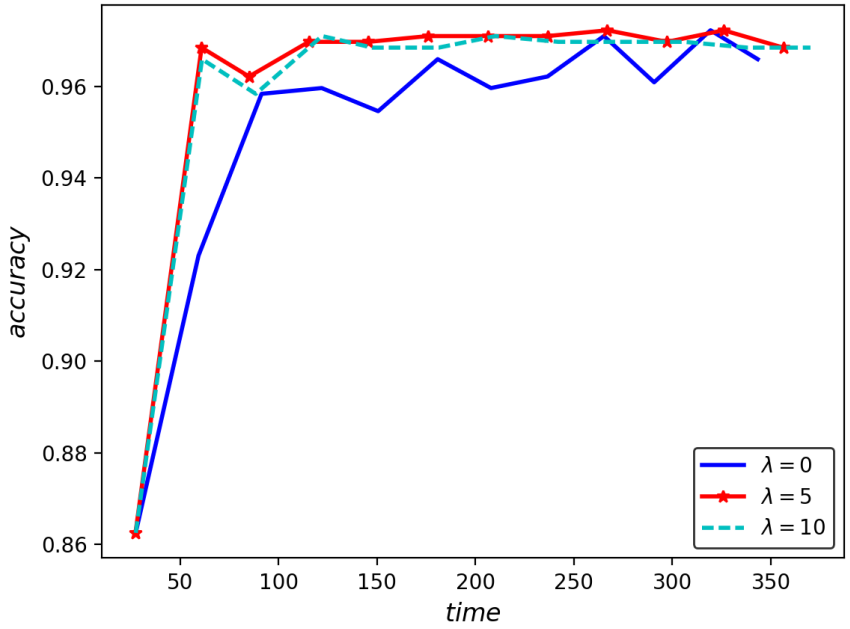

Figure 7 Comparison on accuracy with different lambda

(c)

The iteration number I used is just 12, so from the figure in (b), when lambda = 5, the result is better. I believe the result might change if there were more iterations.

For lambda = 5, the features of top 10 positive values are as followings:

features	'doctor'	'medic'	'inform'	'diseas'	'treatment'	'and'	'effect'	'gordon'	'problem'	'bank'
values	0.409115	0.365873	0.35246	0.330965	0.308238	0.302832	0.300839	0.289572	0.28599	0.282045

And the features of top 10 negative values are as followings:

features	'basebal'	'game'	'player'	'team'	'win'	'plai'	'fan'	'run'	'pitch'	'philli'
values	-0.882569	-0.725519	-0.63883	-0.62689	-0.456391	-0.453508	-0.420471	-0.41965	-0.412251	-0.39157