TSNE on creditcard dataset

In [31]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plot
```

In [32]:

```
data = pd.read_csv("../data/creditcard.csv")
```

In [106]:

```
data.shape
```

Out[106]:

```
(284807, 31)
```

In [120]:

```
data[data["Class"]==1].shape
data[data["Class"]==0].shape
```

Out[120]:

```
(492, 31)
```

We have 492 fraud transaction and 284k legitimate transaction.
It is a highly balanced dataset.
Since it is not possible on my system to run TSNE on the whole dataset, I will sample out 10000 transaction.
For better visualization of TSNE , I will be retaining all the fraud transaction and the rest will be legitimate transaction.

In [33]:

```
data.head(5)
```

Out[33]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.(
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.(
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.

5 rows × 31 columns

In [34]:

```
label = data["Class"]
```

In [35]:

```
fraud = data[data["Class"]==1]
```

Storing all the 492 fraud transaction.

In [37]:

```
fraud.head(5)
```

Out[37]:

	Time	V1	V2	V3	V4	V5	V6	V
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.53738
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.32557
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.56232
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.49619
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.71344

5 rows × 31 columns

In [36]:

```
legitimate = data[data["Class"]==0]
```

Storing all the legtimate trasaction.

In [39]:

```
legitimate.tail(5)
```

Out[39]:

	Time	V1	V2	V3	V4	V5	V6	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1

5 rows × 31 columns

In [62]:

```
fraud_legitimate = pd.concat([fraud,legitimate])
```

Merging the fraud and legitimate transaction , where the top 492 rows are fraud transaction.

In [63]:

```
label = fraud_legitimate["Class"]
```

In [64]:

```
label_removed_data = fraud_legitimate.drop("Class",axis="columns")
```

Storing the label in different column and dropping it from the original dataset.

In [65]:

```
label_removed_data.head(5)
```

Out[65]:

	Time	V1	V2	V3	V4	V5	V6	V
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.53738
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.32557
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.56232
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.49619
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.71344

5 rows × 30 columns

Standardazing the 10k sample

In [66]:

```
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler().fit_transform(label_removed_data)
print (standardized_data.shape)
```

```
(284807, 30)
```

In [67]:

```
print (standardized_data[0:5])
```

```
[[ -1.98803351e+00  -1.18049500e+00   1.18209005e+00  -1.06173009e+0
0
    2.82364668e+00  -3.78330025e-01  -1.07076393e+00  -2.05109117e+0
0
    1.16519974e+00  -2.52140290e+00  -2.54606016e+00   3.13706080e+0
0
   -2.90223023e+00  -5.98049169e-01  -4.47452634e+00   4.25781697e-0
1
   -1.30184927e+00  -3.33208188e+00  -2.00703593e-02   5.12206010e-0
1
    1.64621432e-01   7.04174770e-01  -4.82973057e-02  -7.44982341e-0
1
    5.28688700e-01   8.54040255e-02   3.68789192e-01   6.46988199e-0
1
   -4.34060559e-01  -3.53229393e-01]
 [ -1.98664368e+00  -1.55386353e+00  -1.91200646e+00   7.17863873e-0
1
    1.61642661e+00   9.85191621e-01  -7.99255047e-01   2.63177207e-0
1
   -5.67619269e-02  -2.46627888e-01  -7.70159542e-01  -4.06163304e-0
1
   -5.03543878e-01   6.79714909e-01  -1.76511543e+00   2.18573512e+0
0
    7.60945700e-01   7.06101844e-01   2.05842640e+00   3.48072765e-0
1
    2.72703884e+00   9.00851536e-01   6.00078596e-01   2.20345179e+0
0
   -4.85107060e-01   5.36754832e-01  -3.01438878e-01  -6.26246820e-0
1
    1.08349292e-01   1.76175820e+00]
 [ -1.90262257e+00  -1.17596291e+00   1.06536753e+00  -2.37259151e-0
1
    1.64580752e+00  -5.95277461e-01  -5.68860979e-02   4.54549898e-0
1
   -3.34195428e-01  -2.16864033e-01  -1.40094102e+00   1.99166237e+0
0
   -6.56537899e+00   2.30462752e-02  -1.53360197e+00  -7.63481996e-0
1
   -2.60449679e+00  -5.63008428e+00  -3.12066783e+00  -1.63928380e+0
0
   -5.57800834e-01  -4.00486340e-01  -1.28481561e+00   2.76601411e-0
1
   -1.44192379e-01  -2.99484186e-01  -1.12525600e+00   9.80249581e-0
2
   -4.63607465e-01   6.06031428e-01]
 [ -1.84947237e+00  -2.24536253e+00   8.22601802e-01  -1.71003476e+0
0
    1.89268391e+00  -8.17341490e-01  -1.28092503e+00  -2.82614301e+0
0
   -2.08295370e-01  -2.25524392e-01  -4.40983319e+00   4.79650272e+0
0
   -1.09215606e+01   1.85247446e-01  -7.06357147e+00  -8.00400665e-0
3
   -8.39722898e+00  -1.48332637e+01  -6.12228978e+00   3.78770450e-0
1
   -2.22600349e-01   7.80879950e-01   2.43857859e-01  -6.98535443e-0
1
   -8.83385088e-02   4.84205493e-01  -1.36344272e+00  -2.04923334e+0
0
    2.57381989e+00  -1.17342308e-01]
 [ -1.83824849e+00   6.30132158e-01   1.82869862e+00  -2.83897130e+0
```

```
0
    3.34268556e+00   2.62576763e+00  -1.01912295e+00   1.38505930e+0
0
   -4.15588523e-01  -1.16768851e+00  -2.24776055e+00   2.05870553e+0
0
   -4.61332073e+00   1.47133338e+00  -6.34193167e+00  -3.70623876e-0
1
    2.94647301e+00   7.93489005e+00   3.62990322e+00  -3.34363941e+0
0
    1.17532211e-02  -5.16074308e-01  -9.70346912e-01  -1.05179771e+0
0
   -2.69572150e+00   2.85625685e+00   1.17537644e+00  -2.48152495e-0
2
    4.44715042e-01  -3.49231307e-01]]
```

In [68]:

```
from sklearn.manifold import TSNE
```

In [69]:

```
model = TSNE(n_components=2,random_state=2)
```

In [70]:

```
top_10000 = standardized_data[0:1000]
```

Taking only the top 10000 datapoints as sample.(492 fraud + 9508 legitimate

In [71]:

```
label_10000 = label[0:1000]
```

In [72]:

```
tsne_data =model.fit_transform(top_10000)
```

In [73]:

```
tsne_data = np.vstack((tsne_data.T,label_10000)).T
```

In [74]:

```
import seaborn as sn
```

In [95]:

```
neighbourhood = [30,35,40,45,50,100]
iteration = [500,1000,3000,5000]
```

In [92]:

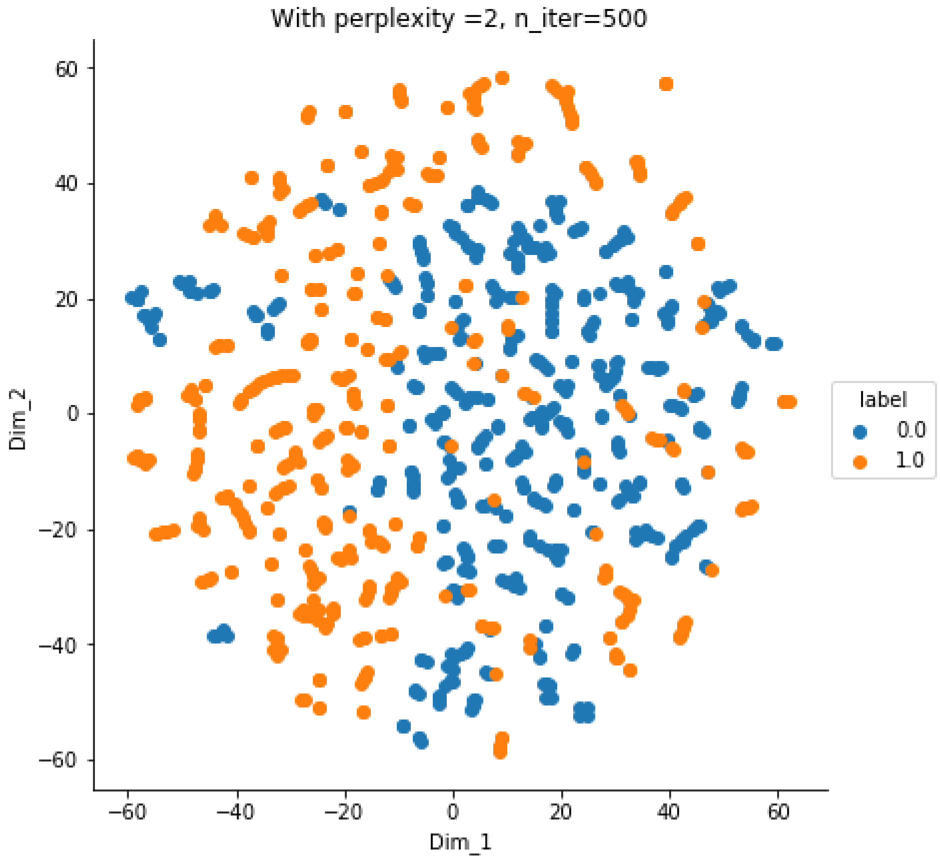
```
def plot_tsne(it,per):
    model = TSNE(n_components=2, random_state=0, perplexity=per,  n_iter=it)
    tsne_data = model.fit_transform(top_10000)

# creating a new data fram which help us in ploting the result data
    tsne_data = np.vstack((tsne_data.T, label_10000)).T
    tsne_df = pd.DataFrame(data=tsne_data, columns=("Dim_1", "Dim_2", "label"))

# Ploting the result of tsne
    sn.FacetGrid(tsne_df, hue="label", size=6).map(plot.scatter, 'Dim_1', 'Dim_
2').add_legend()
    plot.title("With perplexity ="+str(per)+", n_iter="+str(it))
    plot.show()
```

In [122]:

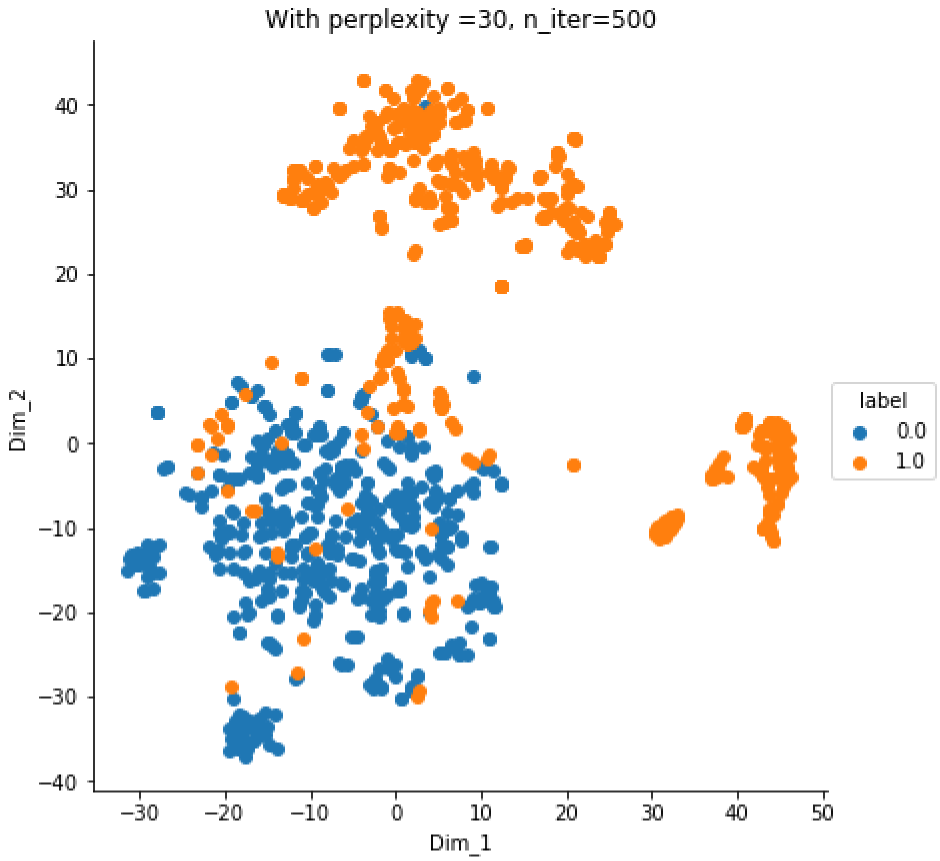
```
plot_tsne(500,2)
```

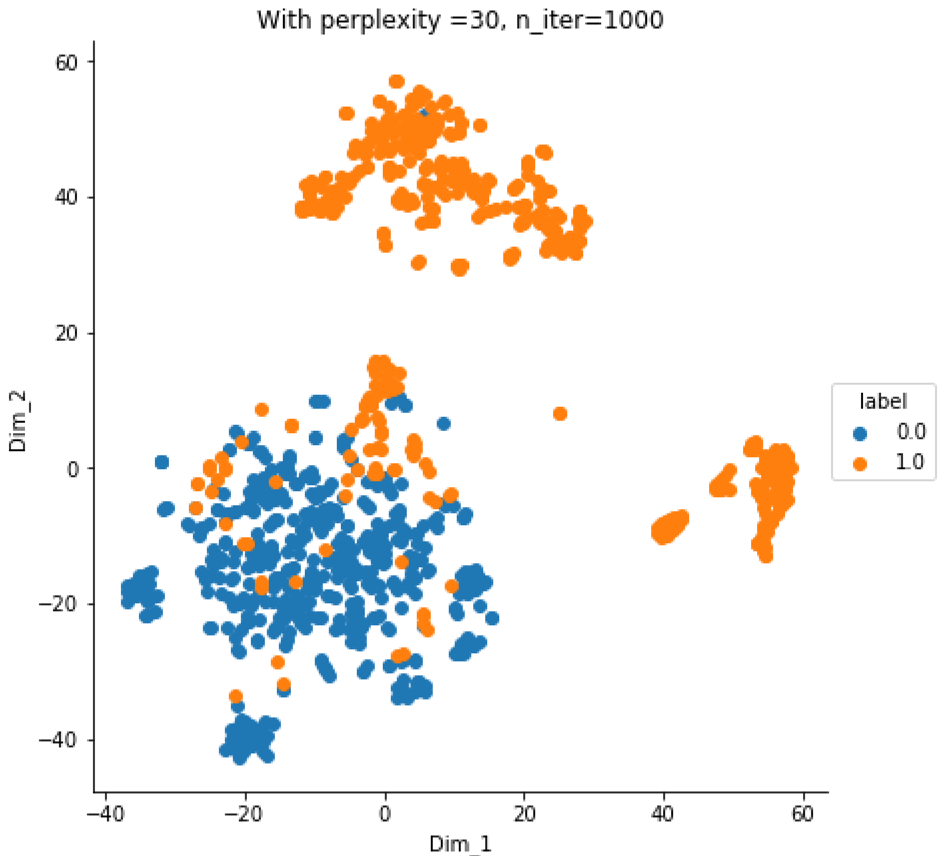


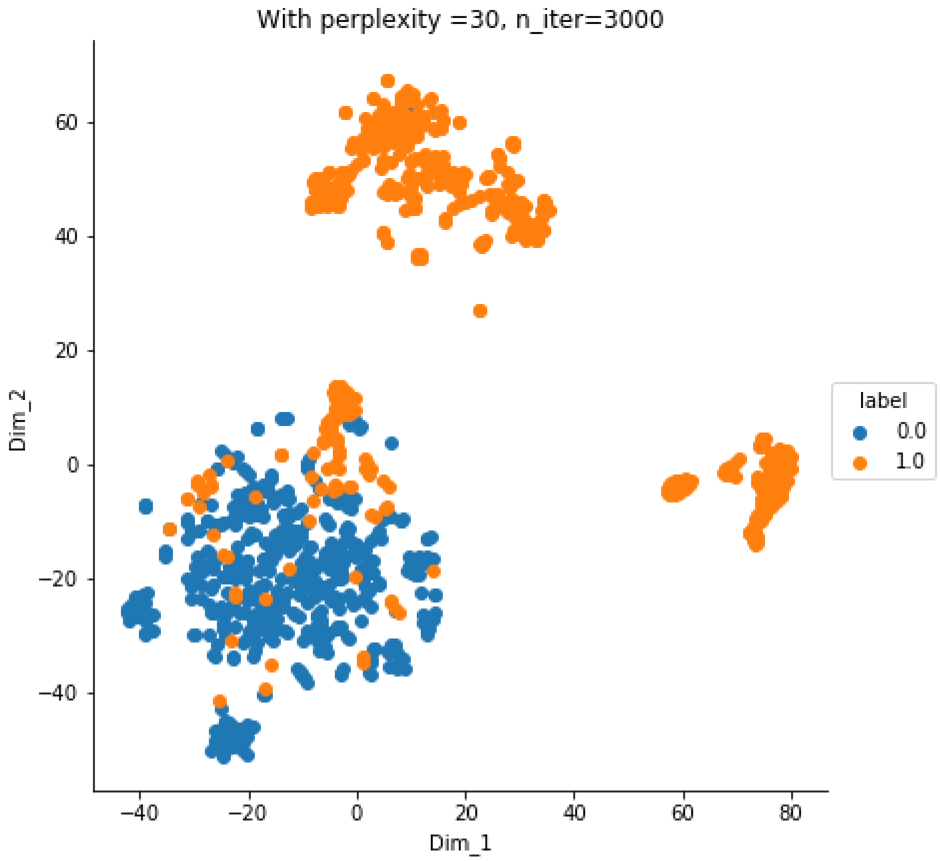
With perplexity = 2 and n_iter = 500 , we can see that all the datapoints are overlapping.

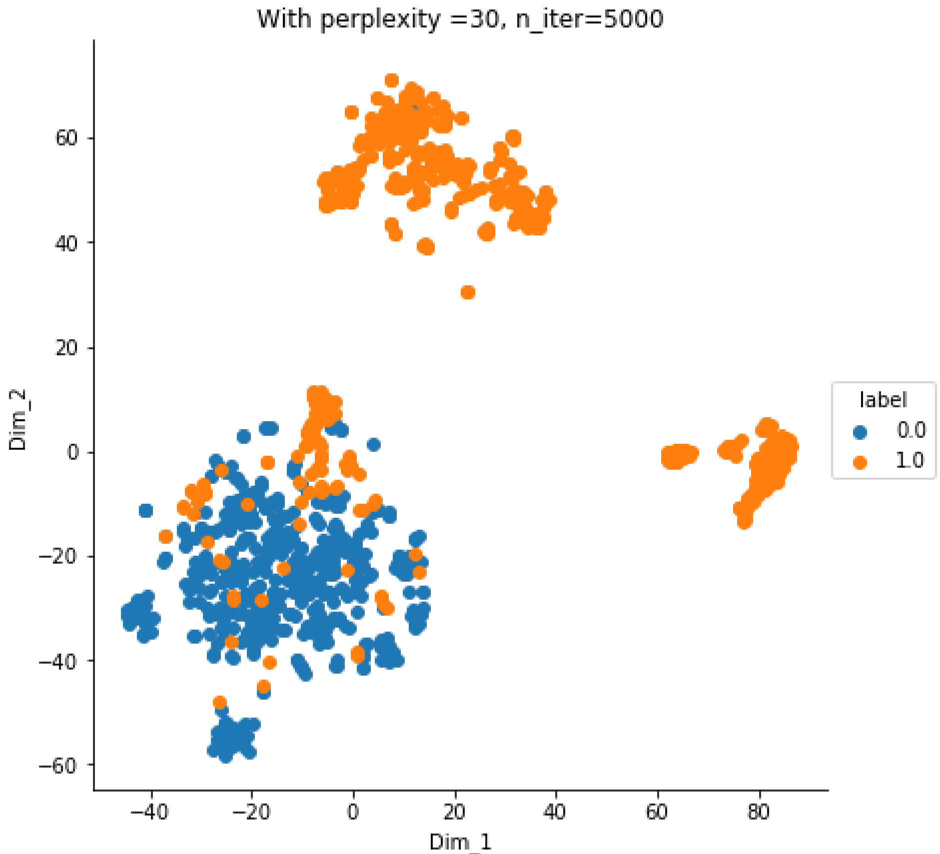
In [96]:

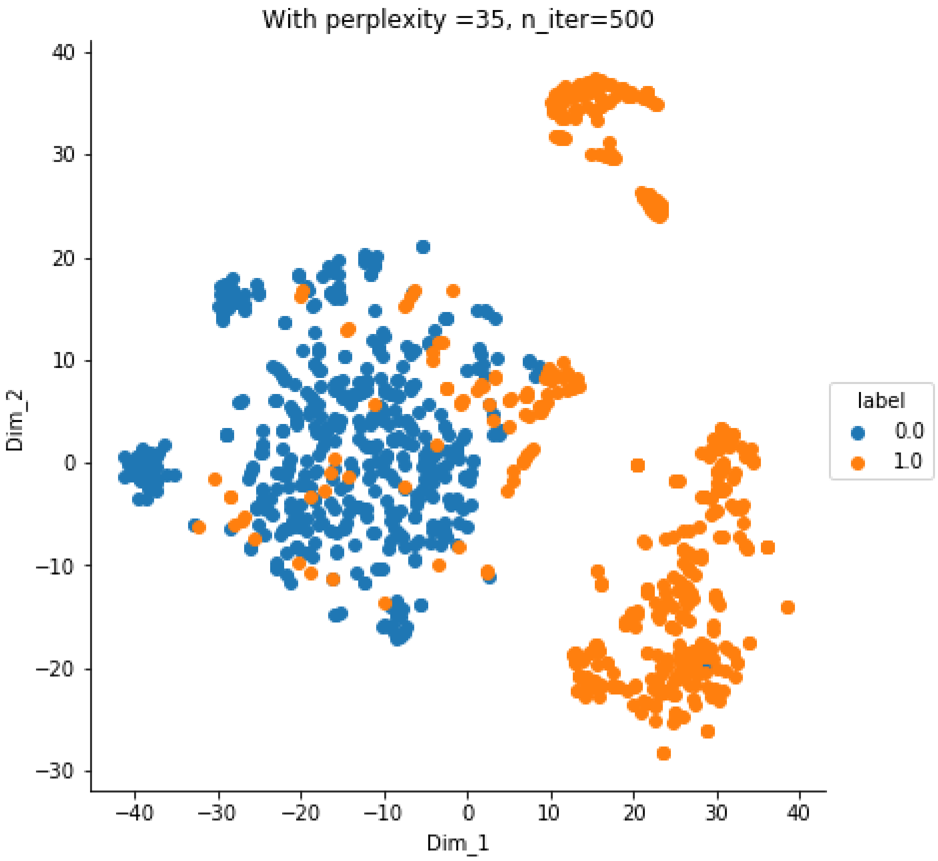
```
for i in neighbourhood:
    for j in iteration:
#         print (i,j)
        plot_tsne(j,i)
```

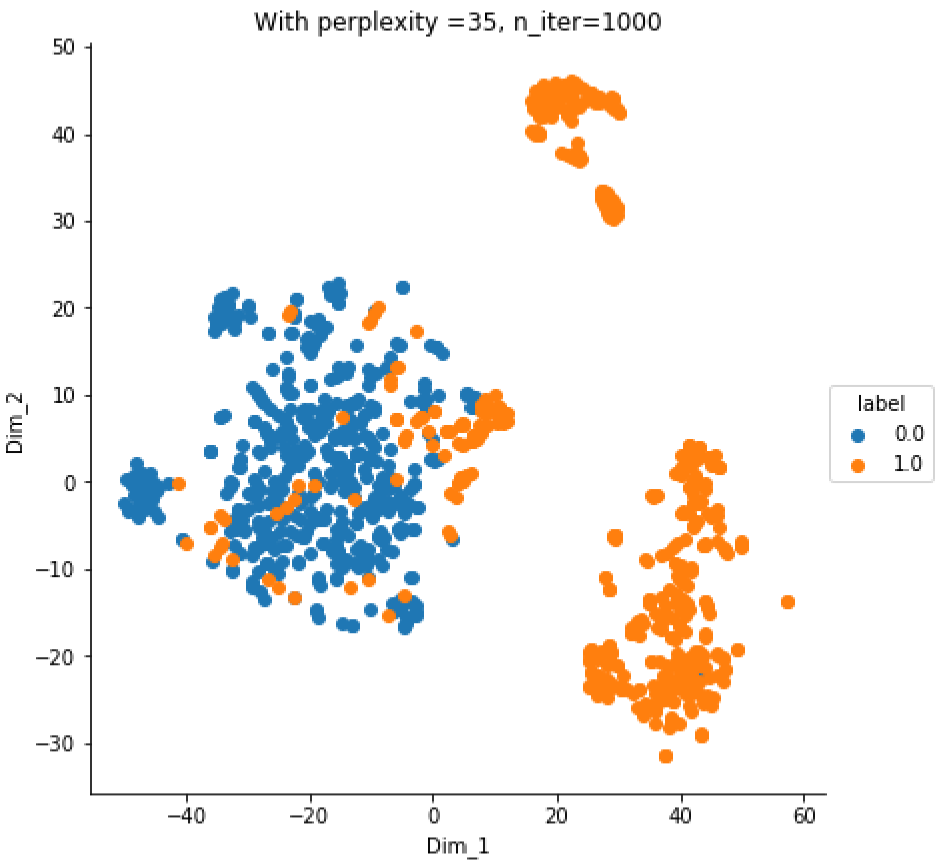

With perplexity =30, n_iter=500
Dim_2
Dim_1
40
30
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30
40
50
label
0.0
1.0

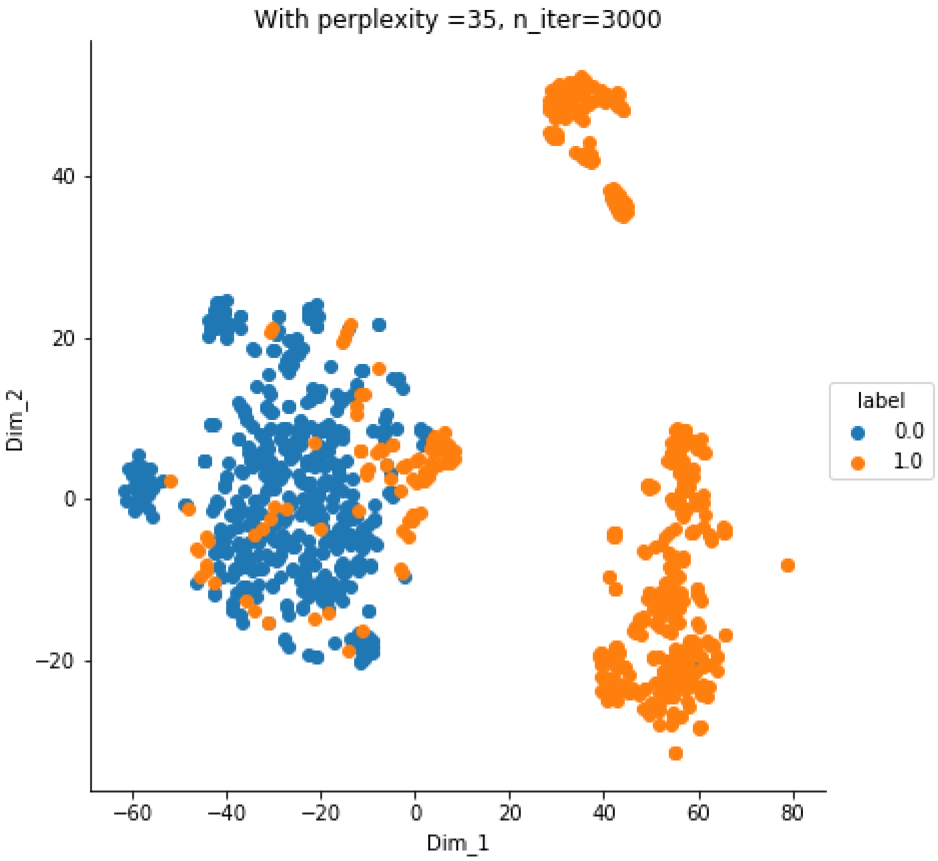

With perplexity =30, n_iter=1000
Dim_2
Dim_1
60
40
20
0
−20
−40
−40
−20
0
20
40
60
label
0.0
1.0

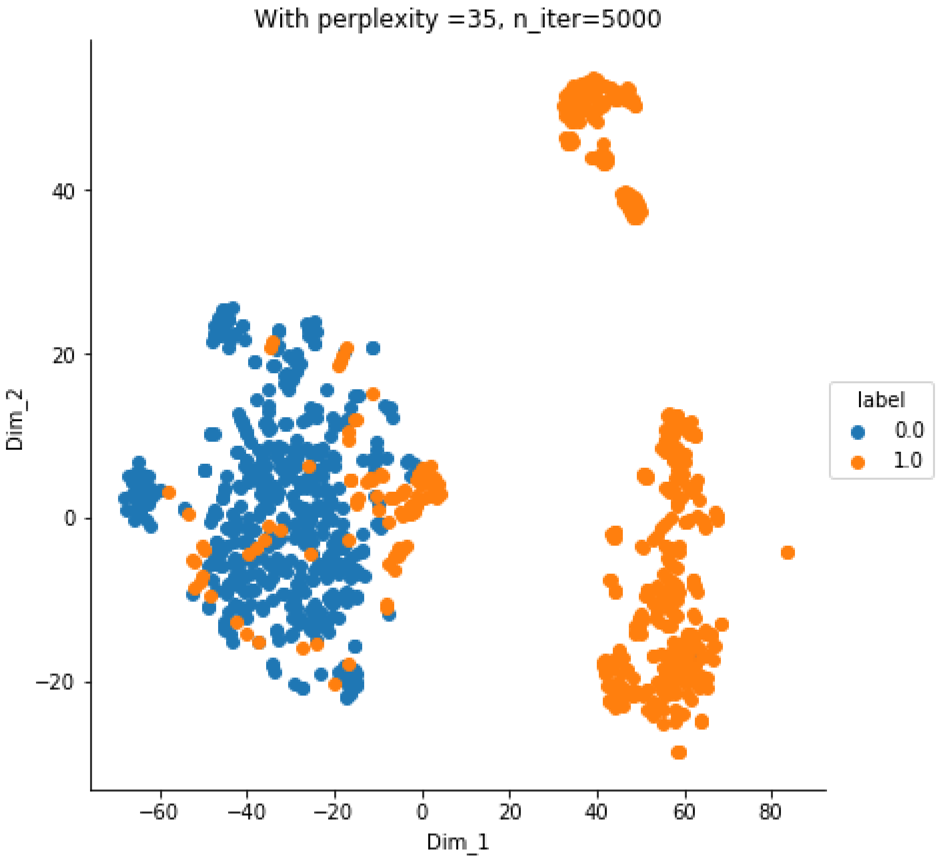

With perplexity =30, n_iter=3000
Dim_2
Dim_1
60
40
20
0
−20
−40
−40
−20
0
20
40
60
80
label
0.0
1.0

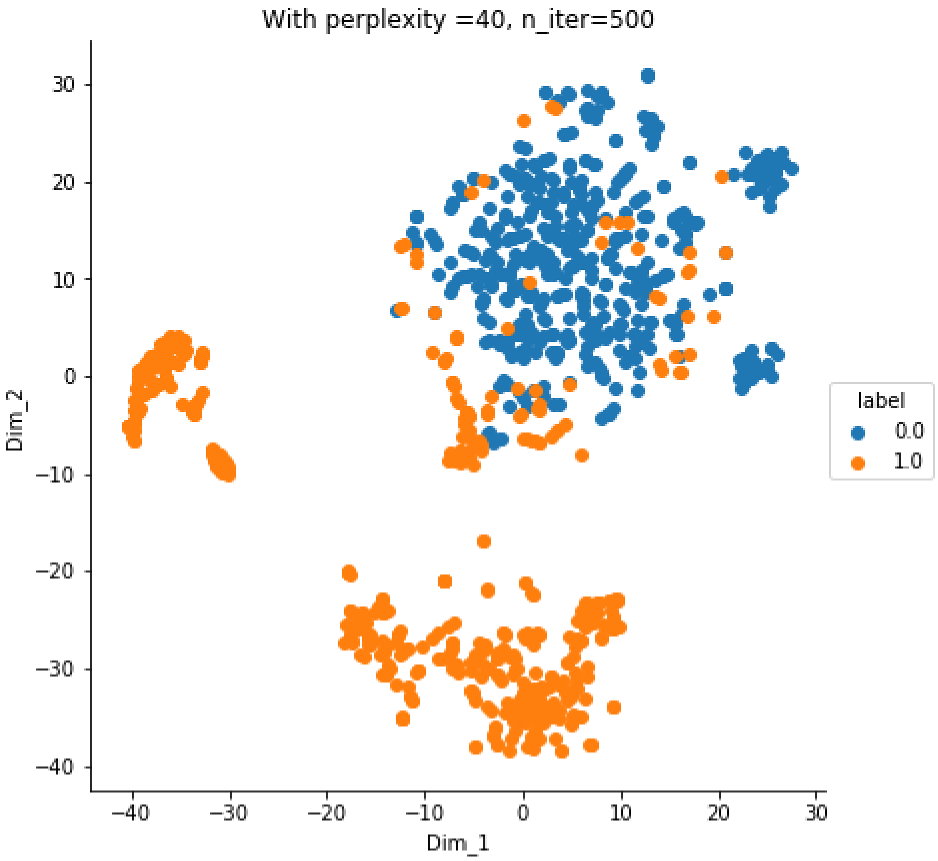

With perplexity =30, n_iter=5000
Dim_2
Dim_1
60
40
20
0
−20
−40
−60
−40
−20
0
20
40
60
80
label
0.0
1.0


With perplexity =35, n_iter=500
Dim_2
Dim_1
40
30
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30
40
label
0.0
1.0

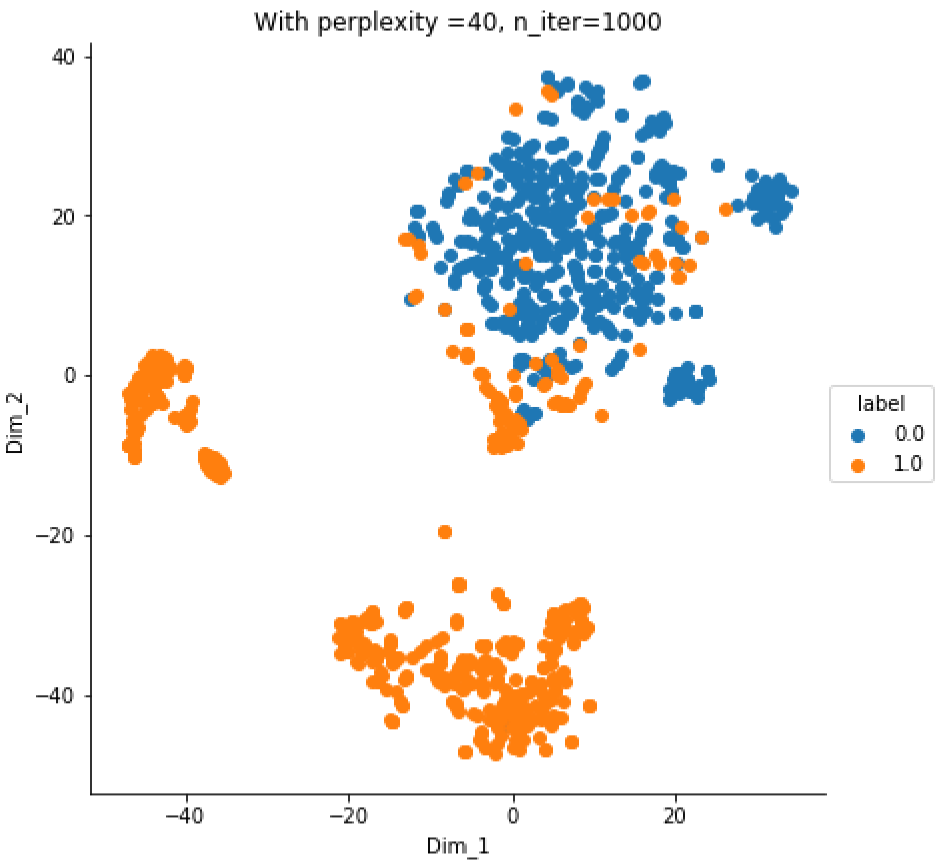

With perplexity =35, n_iter=1000
Dim_2
Dim_1
50
40
30
20
10
0
−10
−20
−30
−40
−20
0
20
40
60
label
0.0
1.0


With perplexity =35, n_iter=3000
Dim_2
Dim_1
40
20
0
−20
−60
−40
−20
0
20
40
60
80
label
0.0
1.0

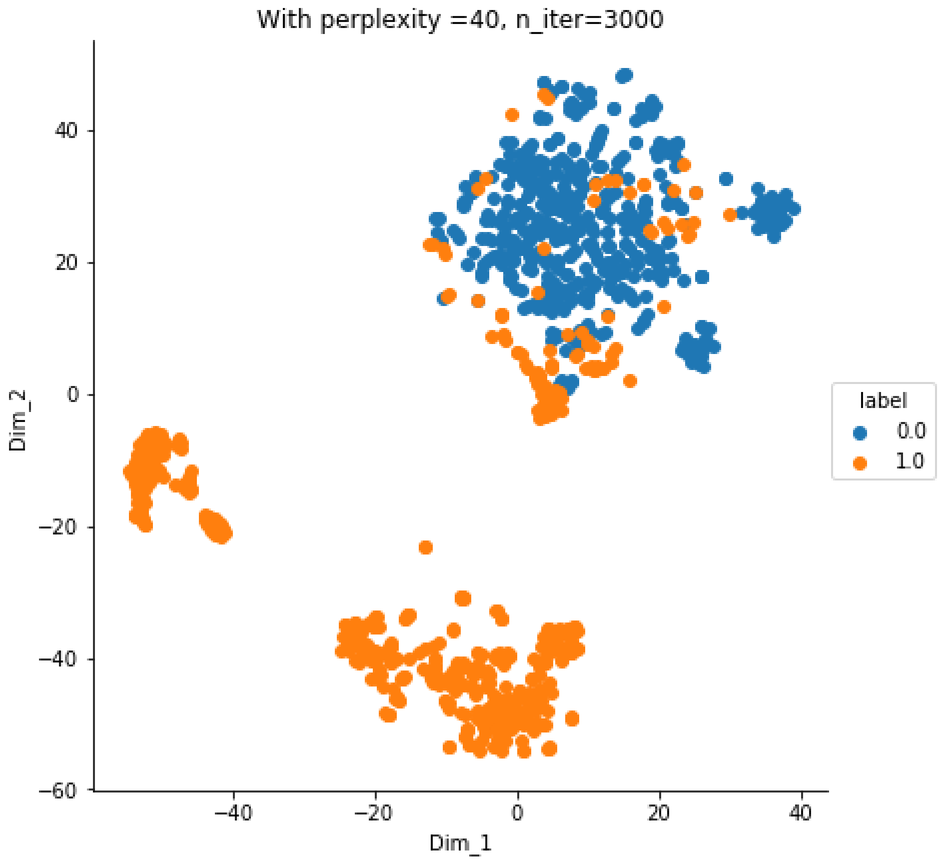

With perplexity =35, n_iter=5000
Dim_2
Dim_1
40
20
0
−20
−60
−40
−20
0
20
40
60
80
label
0.0
1.0

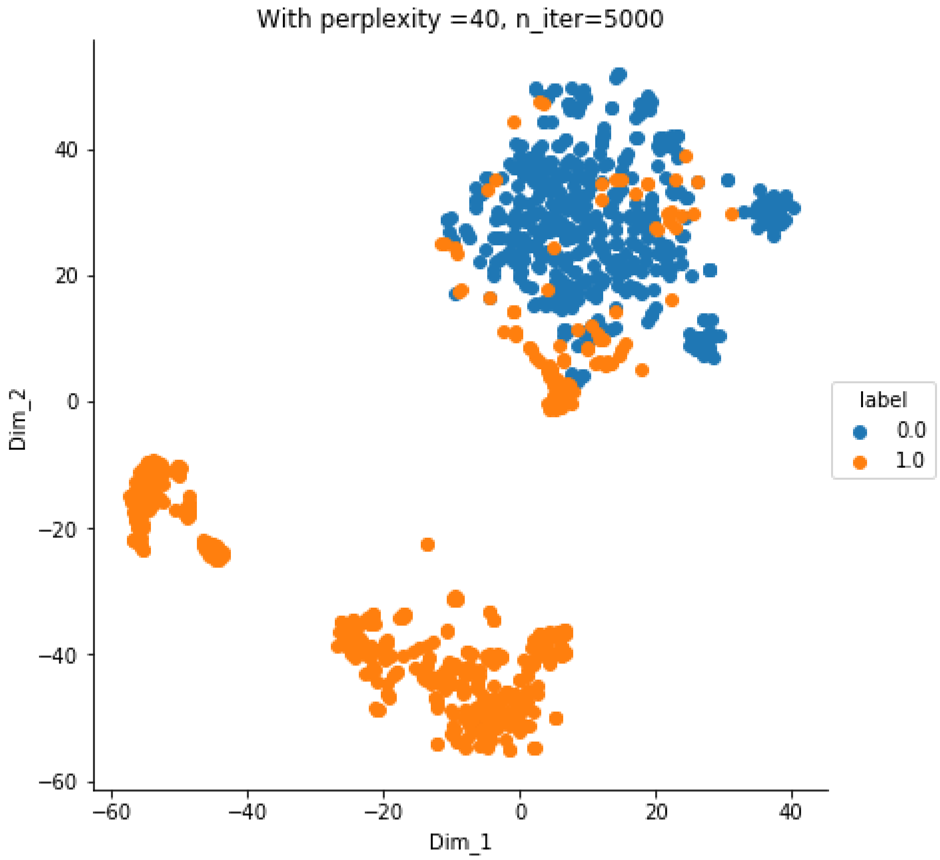


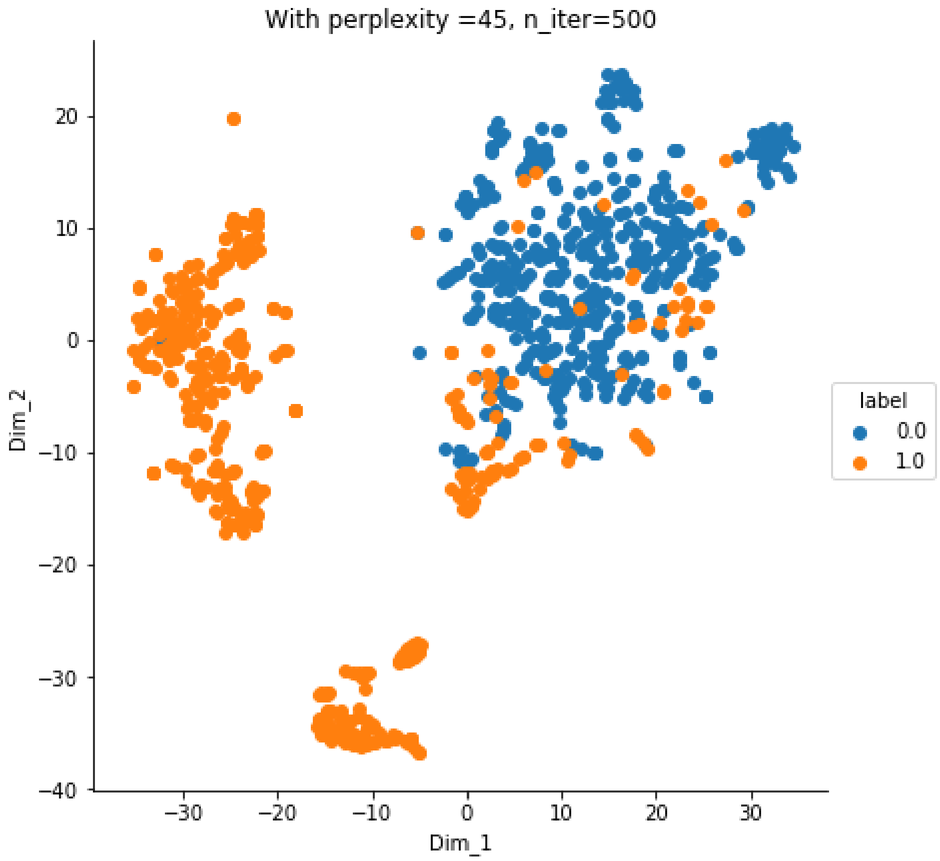
With perplexity =40, n_iter=500
Dim_2
Dim_1
label
0.0
1.0

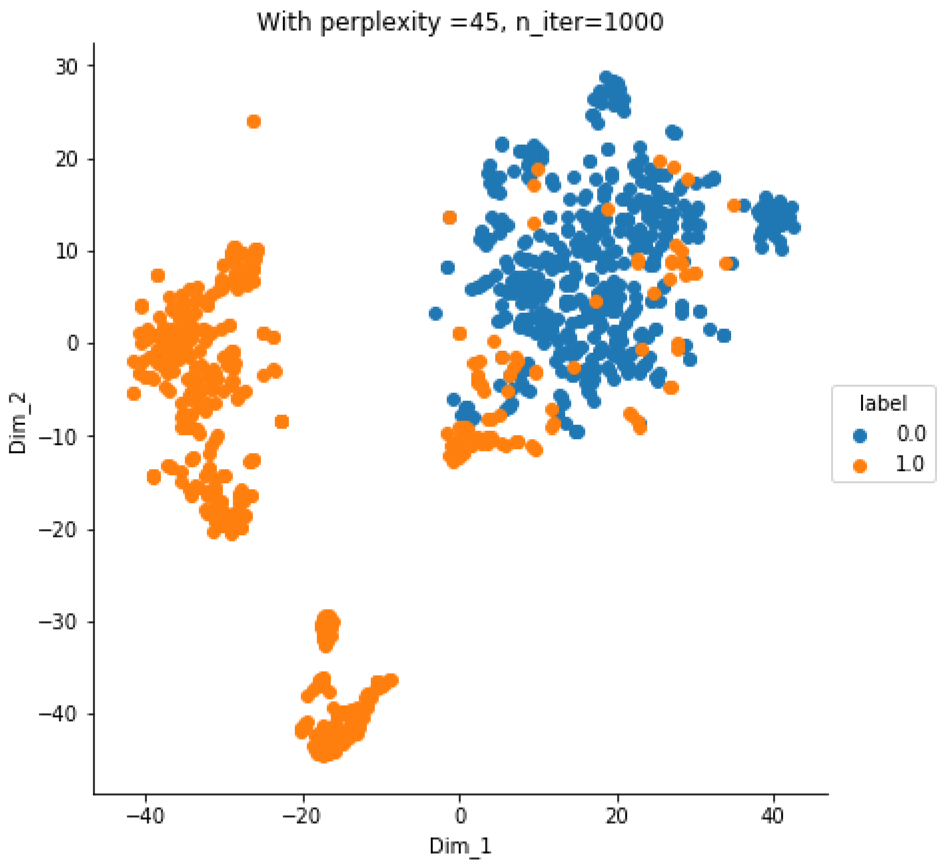


With perplexity =40, n_iter=1000
Dim_2
Dim_1
label
0.0
1.0

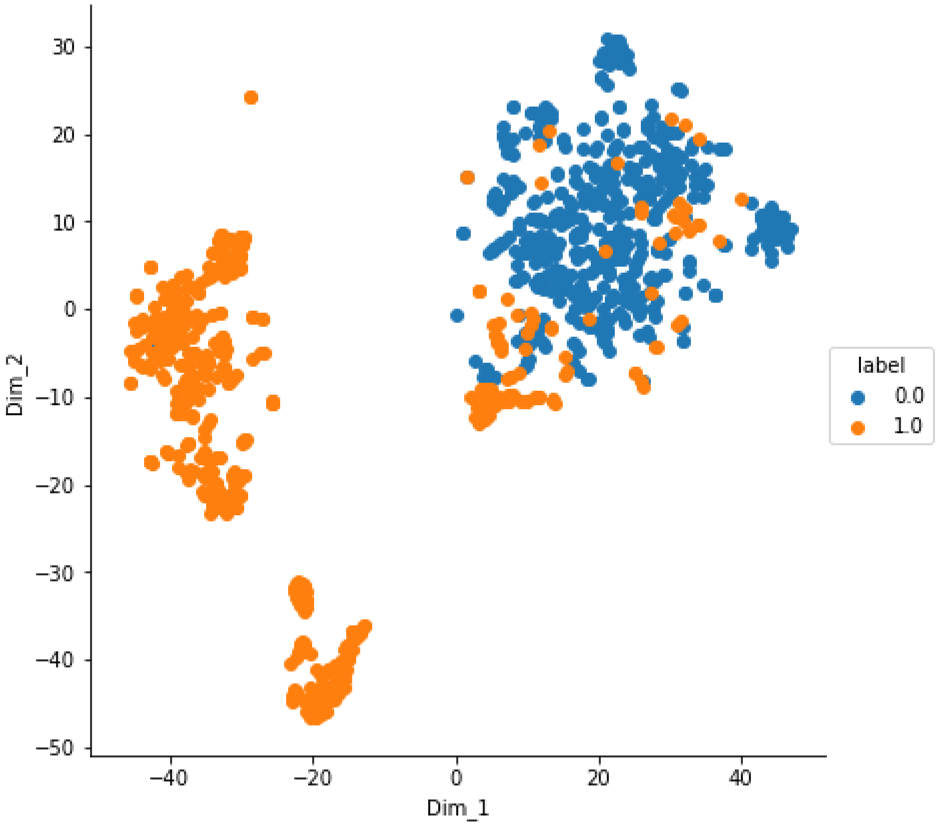

With perplexity =40, n_iter=3000
Dim_2
Dim_1
label
0.0
1.0
40
20
0
−20
−40
−60
−40
−20
0
20
40


With perplexity =40, n_iter=5000
Dim_2
Dim_1
label
0.0
1.0
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40

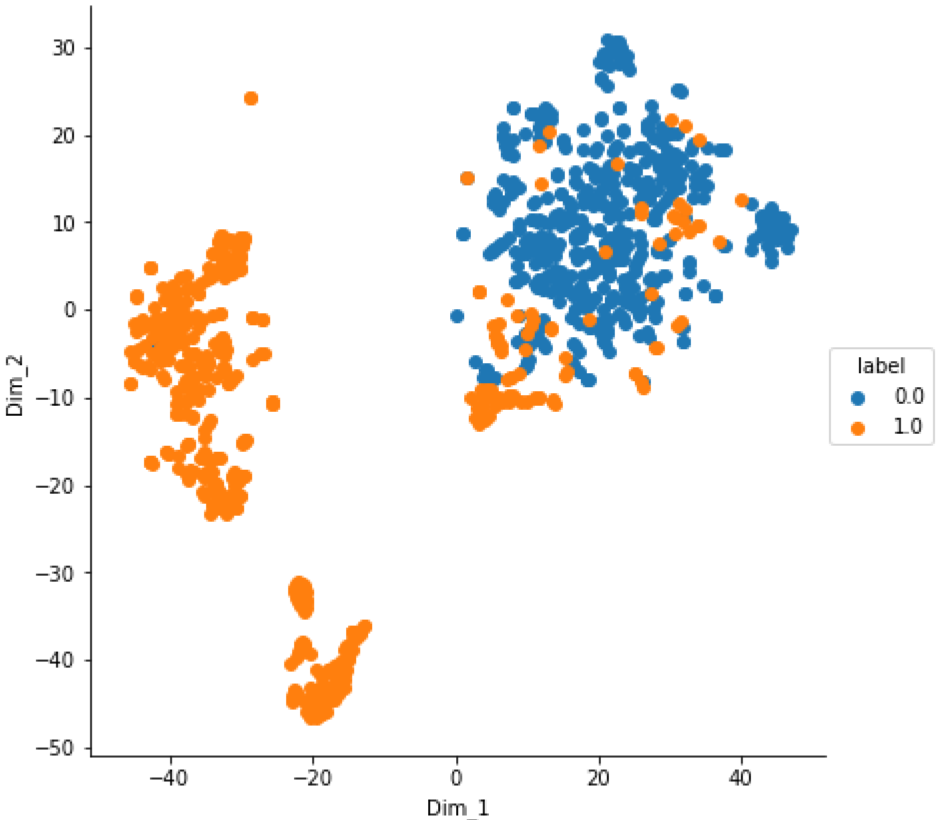

With perplexity =45, n_iter=500
Dim_2
Dim_1
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30
label
0.0
1.0

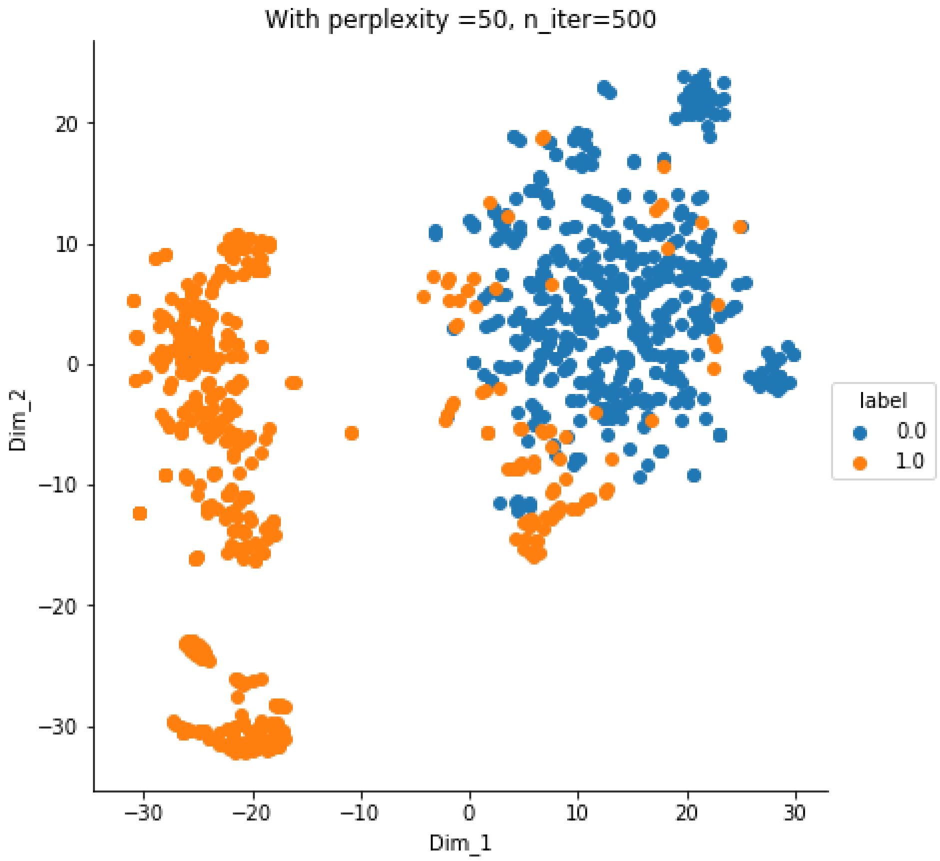

With perplexity =45, n_iter=1000
Dim_2
Dim_1
30
20
10
0
−10
−20
−30
−40
−40
−20
0
20
40
label
0.0
1.0

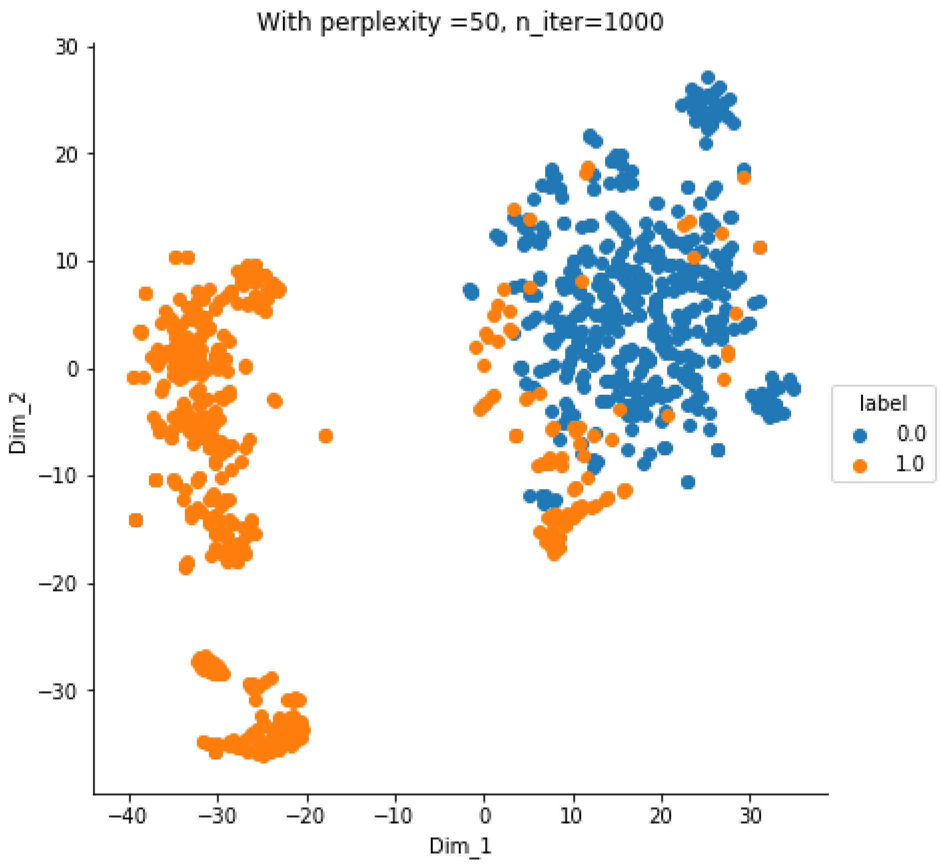
With perplexity =45, n_iter=3000

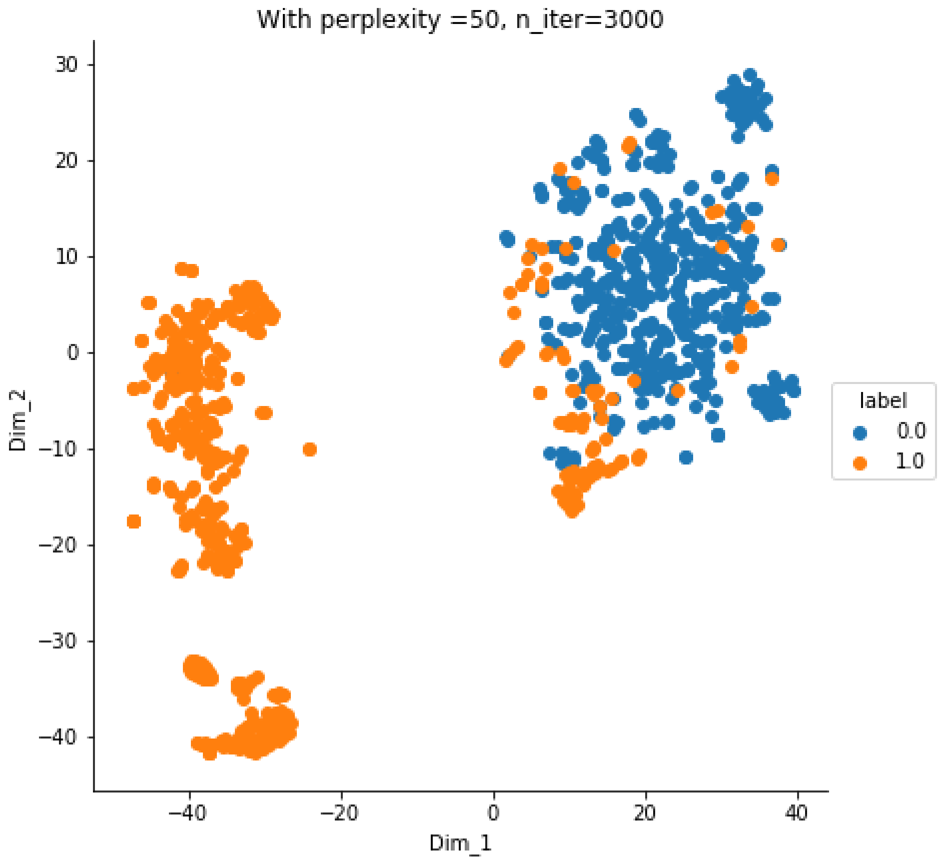


With perplexity =45, n_iter=5000

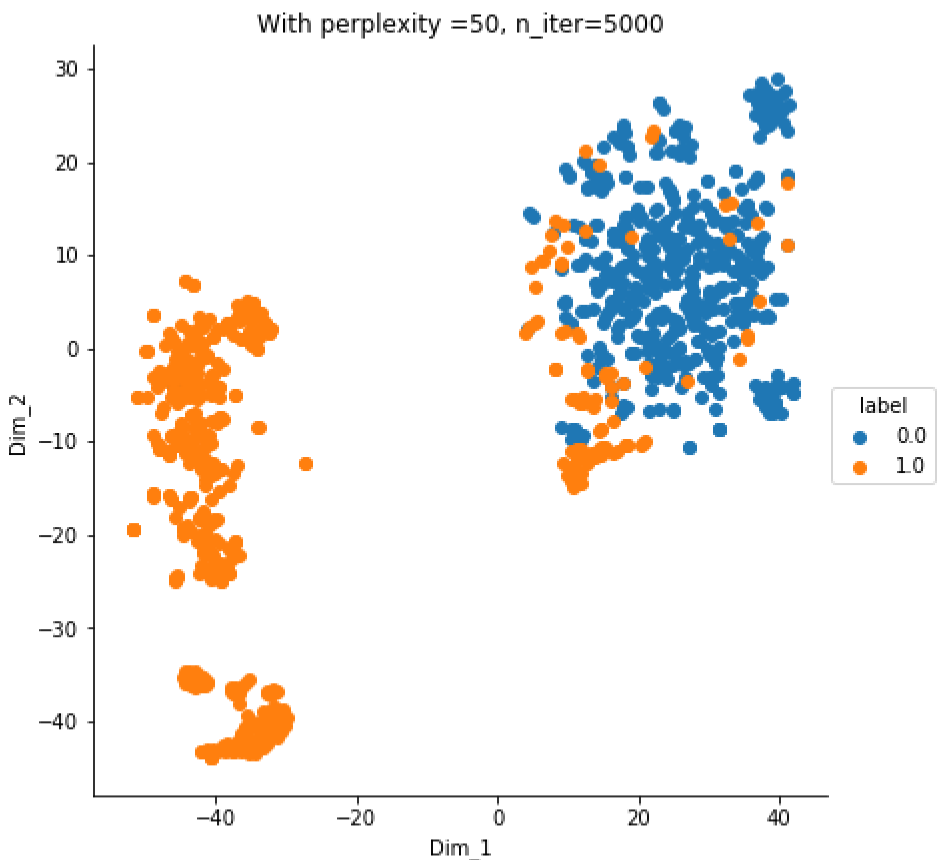



With perplexity =50, n_iter=500
Dim_2
Dim_1
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30
label
0.0
1.0

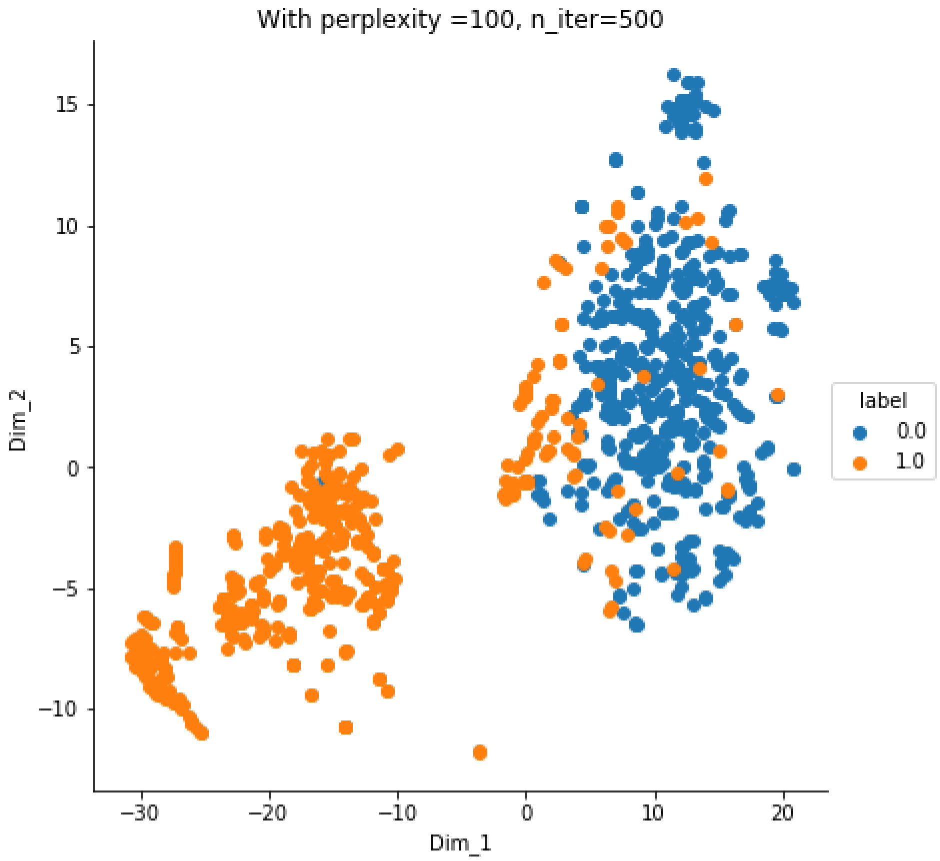

With perplexity =50, n_iter=1000
Dim_2
Dim_1
30
20
10
0
−10
−20
−30
−40
−30
−20
−10
0
10
20
30
label
0.0
1.0

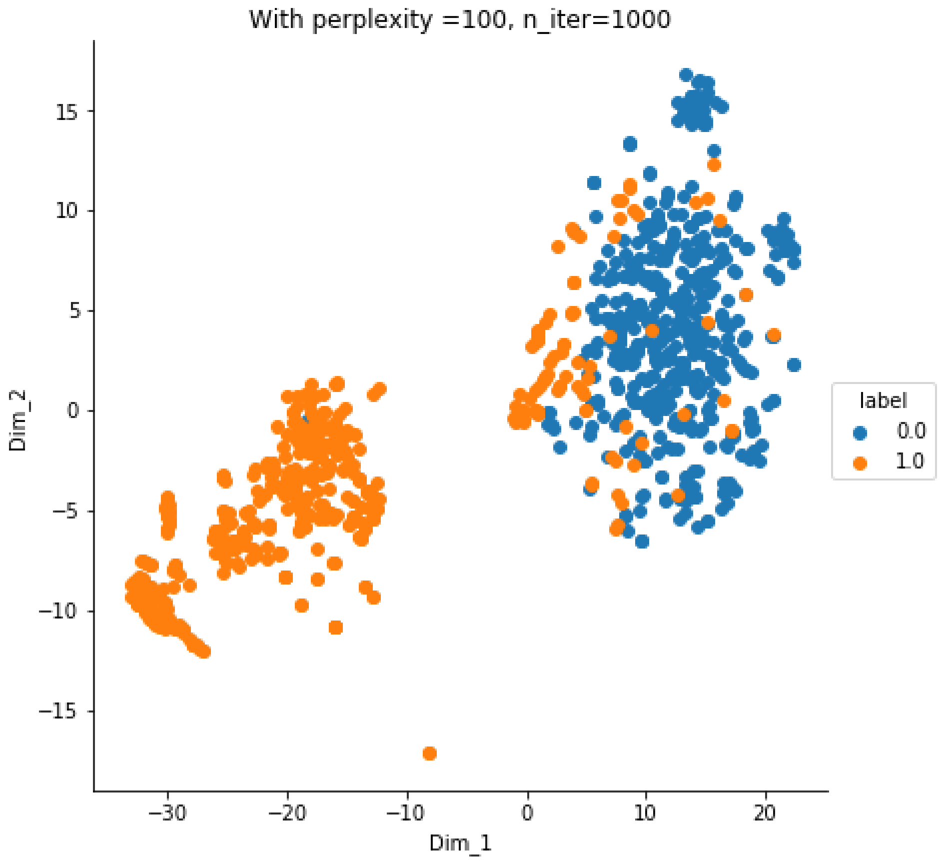


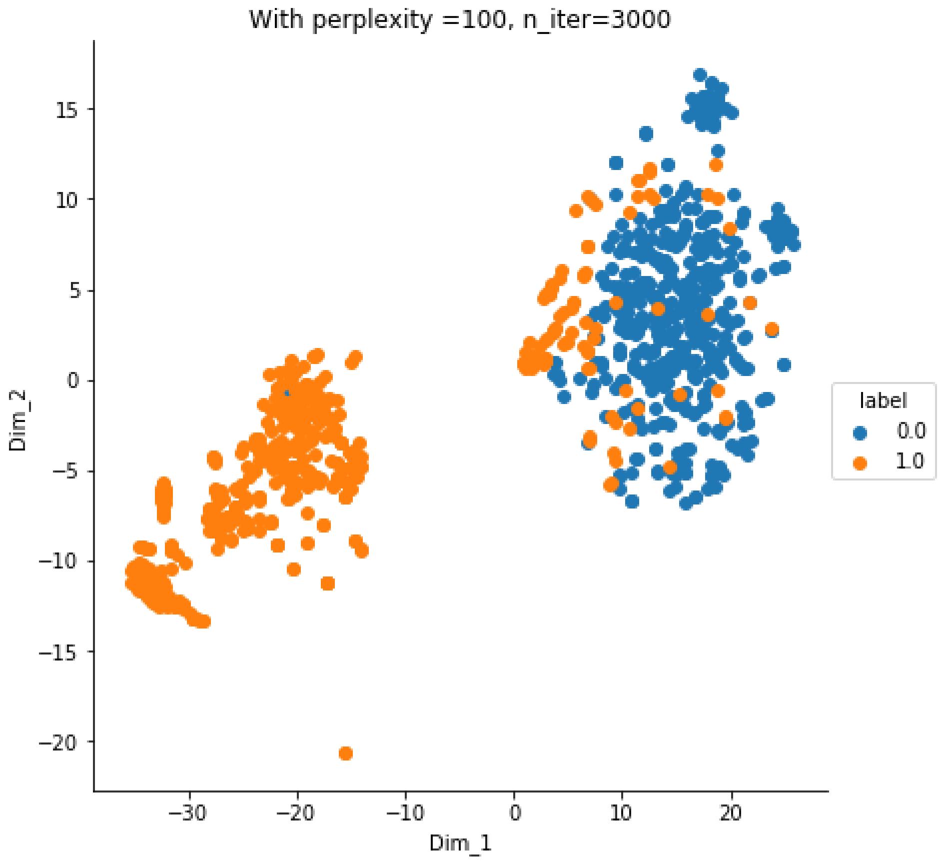
With perplexity =50, n_iter=3000
Dim_2
Dim_1
30
20
10
0
−10
−20
−30
−40
−40
−20
0
20
40
label
0.0
1.0

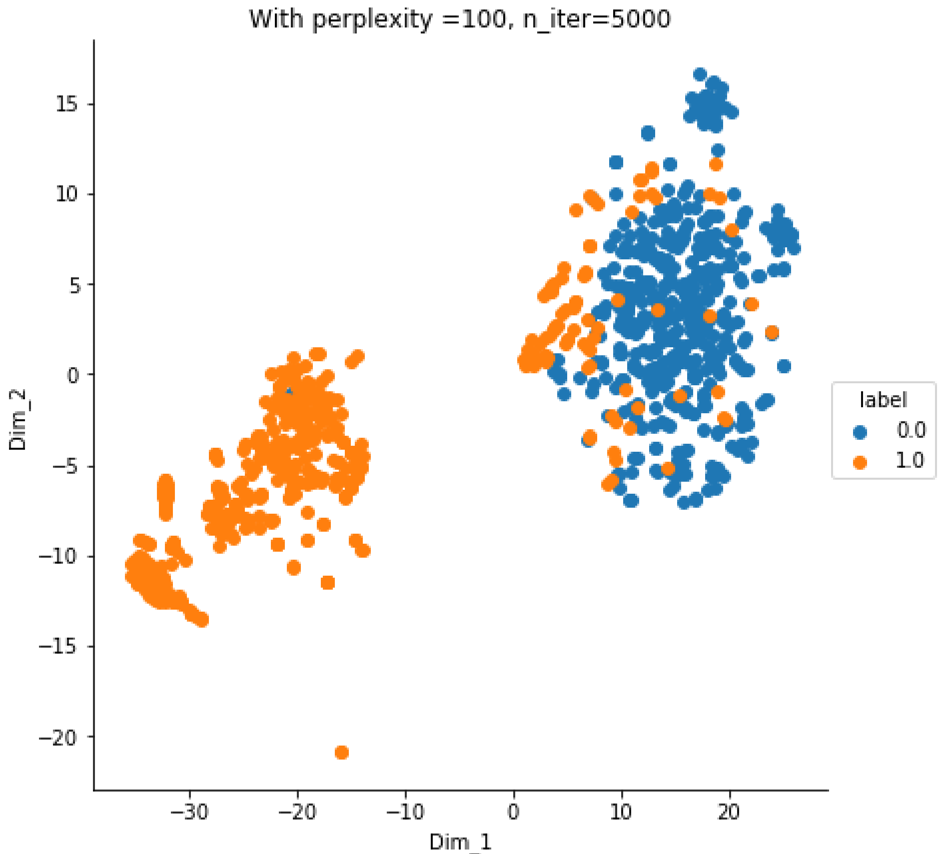


With perplexity =50, n_iter=5000
Dim_2
Dim_1
30
20
10
0
−10
−20
−30
−40
−40
−20
0
20
40
label
0.0
1.0


With perplexity =100, n_iter=500
15
10
5
0
−5
−10
Dim_2
−30
−20
−10
0
10
20
Dim_1
label
0.0
1.0


With perplexity =100, n_iter=1000
15
10
5
0
−5
−10
−15
Dim_2
−30
−20
−10
0
10
20
Dim_1
label
0.0
1.0


With perplexity =100, n_iter=3000
Dim_2
Dim_1
label
0.0
1.0


With perplexity =100, n_iter=5000
Dim_2
Dim_1
label
0.0
1.0

As we experiment with the value of perplexity and the number of iterations , we can see that three clusters are formed(two for fraud and one for legitimate), if the point belongs to one of the fraud cluster we can definitely say that it was a fraud transaction.However we can not say that a transaction is legitimate if it falls int the blue cluster , as there are many orange points in the blue cluster.
As we increase the number of perplexity and no. of iterations , the two fraud clusters come very close and can, be seen as one.