# MDS

pythonでは、skleanパッケージにmanifoldというライブラリがあり、この中に主な次元削減法が全て実装されている。例えばMDSなら、manifold.MDS()関数を使えば良い。

**pythonコード**

...

# MDSオブジェクトを生成（2次元に落とし込む）

mds = manifold.MDS(n\_components=2, max\_iter=100, n\_init=1)

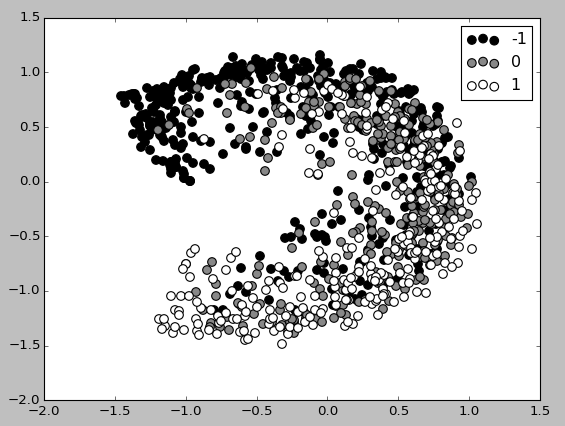
# manifold空間での2次元座標を計算

pos = mds.fit\_transform(data)

# 表示

plt.scatter(pos[:,0], pos[:,1])

plt.show()



## 数学的な話

数学的な話は、以下のページが分かりやすい。

<http://d.hatena.ne.jp/koh_ta/20110514/1305348816>

要するに、個のデータを指定された次元の空間上に配置する時に、データ、間の距離が、定義した距離行列の値に最も近くなるように（最小二乗誤差）、座標を決定するのだ。

つまり、

(1)

さて、データ行列の各行が、各データに対応しているとする。そして、行列を定義した時に、もし行列が分かれば、行列を求めることが出来る。

というわけで、まずは行列を解こう。行列の行列成分は、

(2)

ってことは、

こっらを式(1)に代入すると、

(3)

この式がとっても重要だ！この後の流れで、行列の対角成分が分かる。つまり、、が分かる。そしたら、この式で、任意の成分が求められるのだ。

では、つづきを行くよ。データ行列は、例によって、各列ごとについて、平均値を0とするようにnormalizeしているとすると、

(4)

式(2)をでsumしてやって、この式(4)を代入すると、

同様に、

続いて、式(3)をでsumしてやると、

(5)

同様に、式(3)をでsumしてやると、

(6)

さらに、式(3)を、でsumしてやると、

(7)

式(7)より、

なので、行列のトレース、つまり、対角成分の和が求まる。これを、式(5)、または、式(6)に代入してやれば（どっちの式でも良い）、

より、行列の対角成分が求まった。

そしたら、式(3)より、行列の任意の成分が以下のように計算できる。

(8)

以上で行列が求まったので、後は、行列を計算するだけだ。

行列は、なので、当然、対称行列だよね。式(2)からも、対称行列だと分かるね。対称行列は、次のように特異値分解できることが分かっている。

(9)

※一般的な行列の特異値分解がと違って、となることに注目して欲しい！

ここで、ご存知の通り、は固有値が並んだ対角行列だ。

なので、

よって、式(9)は次のように変形できる。

つまり、

(10)

ってことだね！

以上より、行列を求めることが出来る。

# Isomap

Isomapも同様に、manifold.Isomap()関数で簡単に実現できる。

**pythonコード**

...

# Isomapオブジェクトを生成（2次元に落とし込む）

isomap = manifold.MDS(n\_neighbors=10, n\_components=2)

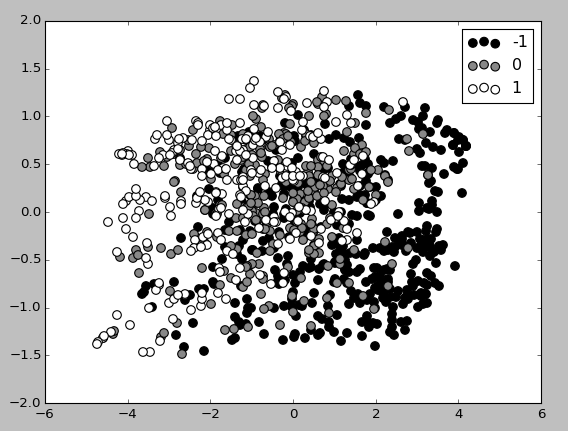
# manifold空間での2次元座標を計算

pos = isomap.fit\_transform(data)

# 表示

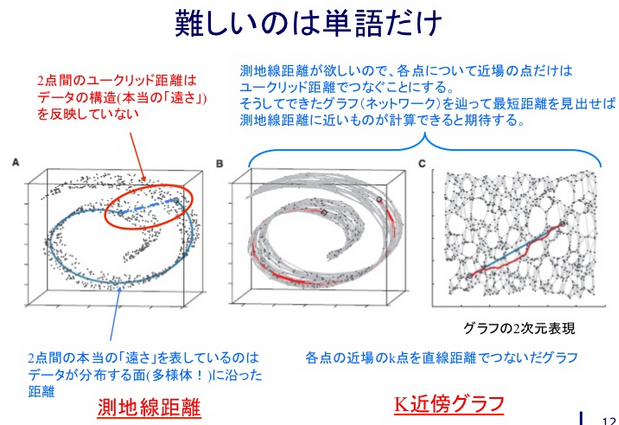
plt.scatter(pos[:,0], pos[:,1])

plt.show()

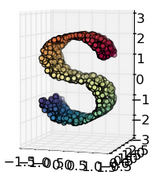


## 数学的な話

近傍点とのユークリッド距離に基づいてグラフを作成し、グラフ上での各2点間の最短距離を**geodesics**距離として計算する。以下のスライドが分かりやすいかな。



これを使って、あとは普通にMDSで2次元に表示するだけ。純粋なMDSと違って、近傍点のみを使うので、例えばsklearnのウェブにある下図のような構造を持つデータの時に、うまいことS字に抽出できるわけだ。



# Spectral Embedding

これも同様に、manifold.SpectralEmbedding()関数で簡単に実現できる。簡単すぎて、ちょっと恐い。

**pythonコード**

...

# Spectral Embeddingオブジェクトを生成（2次元に落とし込む）

se = manifold.SpectralEmbedding(n\_neighbors=10, n\_components=2)

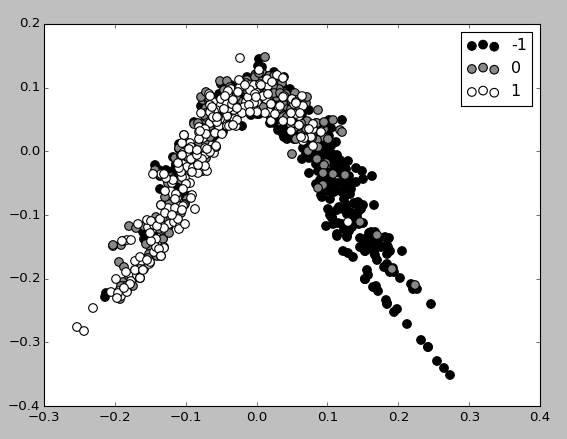
# manifold空間での2次元座標を計算

pos = se.fit\_transform(data)

# 表示

plt.scatter(pos[:,0], pos[:,1])

plt.show()



# t-distributed stochastic neighbor embedding (t-SNE)

**pythonコード**

...

# t-SNEオブジェクトを生成（2次元に落とし込む）

tsne = manifold.TSNE (n\_components=2 , init='pca', random\_state=0)

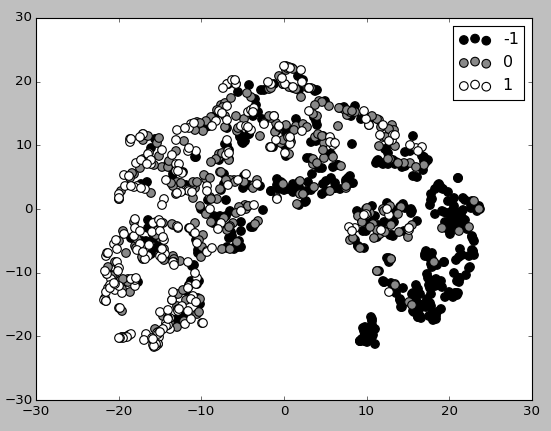
# manifold空間での2次元座標を計算

pos = tsne.fit\_transform(data)

# 表示

plt.scatter(pos[:,0], pos[:,1])

plt.show()



# Locally Linear Embedding

Locally linear embeddingは、manifold.LocallyLinearEmbedding()関数で簡単に実現できる。ただし、引数に、method=’standard’を指定する。この引数を変えることで、LTSA、Hessian LLE、Modified LLEなどを使用できる。

※俺のデータセットでは、なぜかLTSAとHessianはエラーが出て失敗した。

**pythonコード**

...

# Locally linear embeddingオブジェクトを生成（2次元に落とし込む）

lle = manifold. LocallyLinearEmbedding(n\_components=2 , n\_neighbors=10, eigen\_solver=’auto’, method=’standard’)

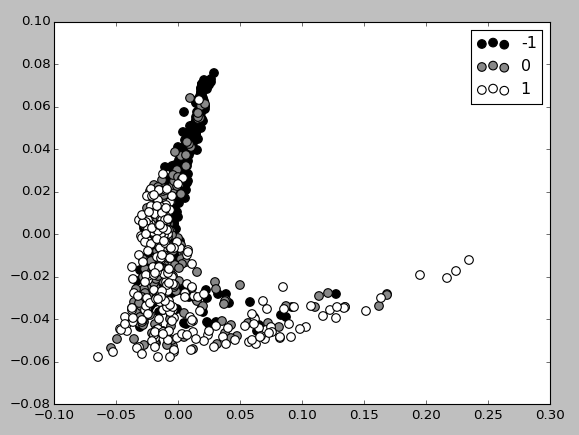
# manifold空間での2次元座標を計算

pos = lle.fit\_transform(data)

# 表示

plt.scatter(pos[:,0], pos[:,1])

plt.show()



# Modified LLE

manifold.LocallyLinearEmbedding()関数で簡単に実現できる。ただし、引数に、method=’modified’を指定する。

**pythonコード**

...

# Locally linear embeddingオブジェクトを生成（2次元に落とし込む）

lle = manifold. LocallyLinearEmbedding(n\_components=2 , n\_neighbors=10, eigen\_solver=’auto’, method=’modified’)

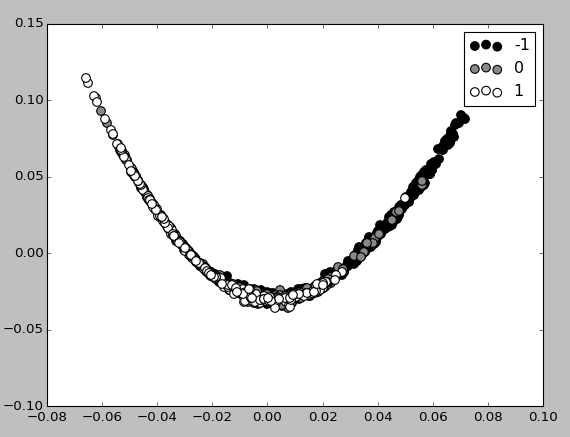
# manifold空間での2次元座標を計算

pos = lle.fit\_transform(data)

# 表示

plt.scatter(pos[:,0], pos[:,1])

plt.show()



# 所感

2次元でみると、ある程度はうまくグループ分けが出来ているケースもある。これは、Inverse方向でlinear regressionがそこそこ良い結果が出したことを示唆しているのかも知れない。

しかし一方で、ものすごい近い位置にあるにもかかわらず、異なるラベルのものもある。これは、linear regressionでは無理があることも示唆していると思う。

そもそも、ものすごい近くなのに、異なるラベルがあるわけだから、quadraticや何やらと、次元を一旦増やしてやる必要があるのだろう。

あるいは、何か別のnonlinear dimensionality reductionでは、うまく分類できるのか？

まずは、quadraticをやってみたいねぇ。。。