机器学习降维方法实验报告:

PCA KPCA LLE

李梓铉

学号 3118103163

指导老师: 孟德宇教授

2018 年 12 月 7 日

1　实验背景

1.1　实验目的

机器学习通常需要大量的训练样本，而在现实中得到的数据通常很难满足训练样本的密集采样，而且属性维度的增加若满足采样条件，会使得样本的数字达到天文数字。在高维情形下出现的数据样本稀疏，巨大的矩阵导致的计算距离困难等等，就是机器学习方法面临的严重障碍“维数灾难”（curse of dimensionality）. 解决维数灾难的重要方法之一是降维（dimension reduction）, 通过数学变换将高维空间转变为一个低维子空间使得样本密度提高，降低距离计算的复杂度。一般来说有效的降维方法要保持前后的样本之间的距离。

降维的方法主要分为两类：

第一类也是降维最简单的方法就是基于线性变换的线性降维方法，多维缩放（MDS），主成分分析（PCA）算法均属于此类降维方法。线性降维方法是高维到低维空间的线性映射，但对于现实任务中的很多数据，需要使用非线性映射才能找到适当的低维嵌入。

第二类方法就是非线性方法，非线性方法中的第一种是核化线性降维，如核主成分分析（KPCA），使用核函数，将高维特征空间中得数据投影到一个超平面上实现降维。第二种是流形学习（manifold learning)，这是一种借鉴拓扑流形概念的降维方法。低维流形嵌入到高维空间的分布虽然很复杂，但局部仍然具有欧式空间的性质，因此可以在局部建议降维映射关系，然后在推广至全局。著名的流形学习方法有等度量映射（Isometric mapping）Isomapping，局部性嵌入（local linear embedding）LLE 等。

在本文中，将使用 PCA，Kpca，Isomap，LLE 四种方法对随机生成的高维数据进行降维。通过实验了解各类降维方法的特点，在理解原理的基础上对各类方法有更直观的认识，然后再对一个现实工程问题进行方法的应用。报告将包含以下 5 个主要部分:

- 主成分分析方法（PCA）
- 核主成分分析方法（KPCA）
- 局部性嵌入（LLE）
- 工程应用
- 结论与讨论

1.2　编译环境

使用 python 语言进行实验，相关的编译环境如下:

- 操作系统 Windows 10
- python3.6 编译器

- scikit-learn 0.20.1 （机器学习 python 库）
- numpy （python 数组处理库）
- matplotlib（python 可视化库）

2 PCA

2.1 方法介绍

主成分分析（Principal Component Analysis）是最常用的一种降维方法。PCA 通过用一个超平面对正交空间中的所有样本点进行近似表达。这个超平面要满足两点：1. 重构性：样本点到这个超平面的距离都足够近。2. 最大可分性：样本点在这个超平面上的投影能尽可能分开。

算法的主要过程可以总结如下：

1. 输入：样本集 D，期望的低维空间维数 n。
2. 对所有样本进行中心化。
3. 计算样本的协方差矩阵。
4. 对协方差矩阵做特征值分解。
5. 排序特征值取最大的 n 个特征值所对应的特征向量。
6. 输出: 特征向量对应的投影矩阵

降维后的低维空间的维数由使用者决定，可以通过选不同的维数通过交叉验证得到较好的维数 n，显然在 PCA 过程中，较小的特征值对应的特征向量对应的信息被舍弃了，这是降维的必要结果，因为较小的特征值对应的信息往往是无用的噪声信息，舍弃后可以使样本采样密度增大，一定程度上起到去噪的效果。

2.2 实验介绍

仿照 sci-kit 示例程序，本实验使用 numpy 生成三个随机数组，将数组通过线性变化，变换为三维空间中的点云，点云分布有显著的接近一个平面（x-y-z=0）上分布的特征。然后程序通过调用 sci-kit 的 PCA 程序，设置低维空间维数为 3，得到一个超平面 (近似平面 x-y-z=0) 来近似所有的样本点，并进行了可视化。

可视化结果如下。

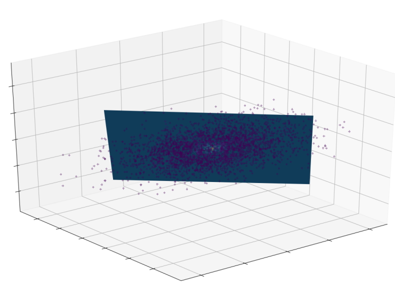

图 1: PCA 点云近似可视化结果

2.3 源代码

实验使用的源代码如下.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
=========================================================
Principal components analysis (PCA)
=========================================================

These figures aid in illustrating how a point cloud
can be very flat in one direction--which is where PCA
comes in to choose a direction that is not flat.

"""
print(__doc__)
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats


###############################################################################
# Create the data

e = np.exp(1)
np.random.seed(4)


def pdf(x):
    return 0.5 * (stats.norm(scale=0.25 / e).pdf(x)
                  + stats.norm(scale=4 / e).pdf(x))

y = np.random.normal(scale=0.5, size=(30000))
x = np.random.normal(scale=0.5, size=(30000))
z = np.random.normal(scale=0.1, size=len(x))

density = pdf(x) * pdf(y)
pdf_z = pdf(5 * z)

density *= pdf_z

a = x + y
b = 2 * y
c = a - b + z

norm = np.sqrt(a.var() + b.var())
a /= norm
b /= norm


###############################################################################
# Plot the figures
def plot_figs(fig_num, elev, azim):
    fig = plt.figure(fig_num, figsize=(4, 3))
    plt.clf()
    ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=elev, azim=azim)

    ax.scatter(a[::10], b[::10], c[::10], c=density[::10], marker='+', alpha=.4)
    Y = np.c_[a, b, c]

    # Using SciPy's SVD, this would be:
    # _, pca_score, V = scipy.linalg.svd(Y, full_matrices=False)

    pca = PCA(n_components=3)
    pca.fit(Y)
    pca_score = pca.explained_variance_ratio_
```

```
    V = pca.components_

    x_pca_axis, y_pca_axis, z_pca_axis = 3 * V.T
    x_pca_plane = np.r_[x_pca_axis[:2], - x_pca_axis[1::-1]]
    y_pca_plane = np.r_[y_pca_axis[:2], - y_pca_axis[1::-1]]
    z_pca_plane = np.r_[z_pca_axis[:2], - z_pca_axis[1::-1]]
    x_pca_plane.shape = (2, 2)
    y_pca_plane.shape = (2, 2)
    z_pca_plane.shape = (2, 2)
    ax.plot_surface(x_pca_plane, y_pca_plane, z_pca_plane)
    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])


elev = -40
azim = -80
plot_figs(1, elev, azim)

elev = 30
azim = 20
plot_figs(2, elev, azim)

plt.show()
```

3 KPCA

3.1 方法介绍

核主成分分析（Principal Component Analysis）是核化线性降维的一种方法。核函数是一种数学技巧变换技巧，可以将样本映射到一个高维特征空间去。KPCA 使用核函数可以对样本进行非线性的映射再做降维处理。KPCA 的优势在于，映射后的数据集在映射后通常可以保留聚类特征，有时对于扭曲的流形数据集也有很好的效果。KPCA 常用的核函数有 Gaussian Radial Basis Function (RBF) 核函数。Gaussian RBF Function 如下：

$$\phi\gamma(x,\iota) = exp(-\gamma\|x-l\|^2)$$

通过将数据集的每一个样本用核函数映射到一个高维特征空间去。相比原来的样本，新样本会有很多新的维度，同时也因此提高了数据集线性可分的概率。然后再在特征空间中实施 PCA 即可。核化的弊端在于训练集如果很大的话，那么核化后的数据的 feature 将非常多，并且为了获得投影，KPCA 要对所有样本求和，计算开销较大。

算法的主要过程可以总结如下：

1. 输入：样本集 D，期望的低维空间维数 n，核函数 ϕ
2. 得到样本在高维特征空间的像。
3. 实施 PCA
4. 输出: 特征向量对应的投影矩阵

3.2 实验介绍

仿照 sci-kit 示例程序，本实验使用 scikit 的 datasets 库方法，生成两组 sample，在 x1，x2 空间上分布近似为两个同心圆。先调用了 PCA，对样本进行了处理并输出了最大特征值对应的两个特征向量下的样本空间。然后调用 scikit 的 K_pca 方法，使用 rbf_kernel，对样本进行了处理。并输出了最大的两个特征值对应的特征向量描述的样本空间。然后输出了在原 x1，x2 空间中，经过 KPCA 处理的样本。可视化结果如下。

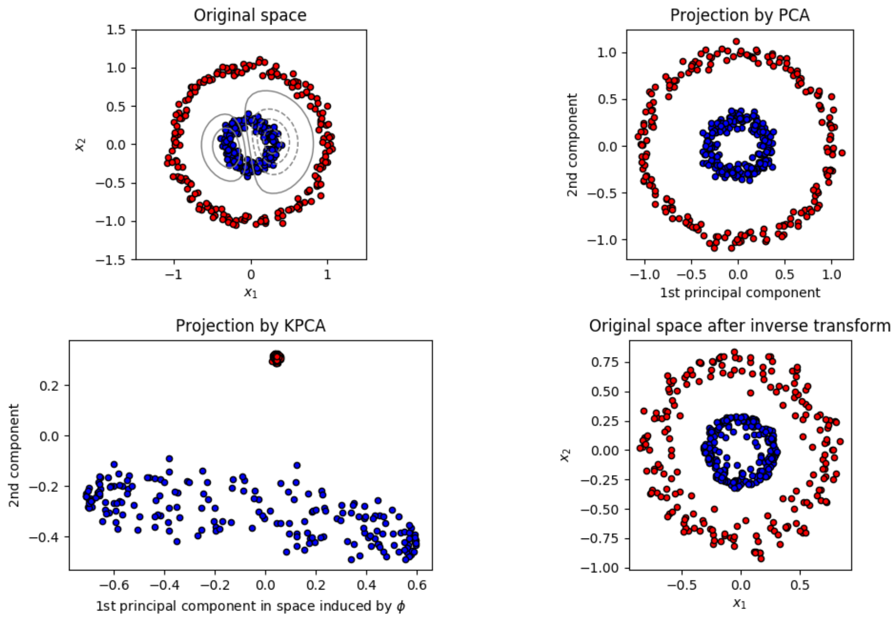

图 2: KPCA 可视化结果

可以看出从图 Projection by KPCA 看出，经过 KPCA 处理，新的样本变的线性可分了，并且根据 Original space after inverse transform 可以看出，KPCA 很好的保留了样本的原始聚类特征。而从 Projection by PCA 可以看出，线性降维后的样本和原来的样本很接近，并没有提取出样本中的聚类特征。

3.3 源代码

实验使用的源代码如下.

```
"""
==========
Kernel PCA
==========

This example shows that Kernel PCA is able to find a projection of the data
that makes data linearly separable.
"""
print(__doc__)


import numpy as np
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA, KernelPCA
from sklearn.datasets import make_circles

np.random.seed(0)

X, y = make_circles(n_samples=400, factor=.3, noise=.05)

kpca = KernelPCA(kernel="rbf", fit_inverse_transform=True, gamma=10)
```

```
X_kpca = kpca.fit_transform(X)
X_back = kpca.inverse_transform(X_kpca)
pca = PCA()
X_pca = pca.fit_transform(X)

# Plot results

plt.figure()
plt.subplot(2, 2, 1, aspect='equal')
plt.title("Original space")
reds = y == 0
blues = y == 1

plt.scatter(X[reds, 0], X[reds, 1], c="red",
            s=20, edgecolor='k')
plt.scatter(X[blues, 0], X[blues, 1], c="blue",
            s=20, edgecolor='k')
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")

X1, X2 = np.meshgrid(np.linspace(-1.5, 1.5, 50), np.linspace(-1.5, 1.5, 50))
X_grid = np.array([np.ravel(X1), np.ravel(X2)]).T
# projection on the first principal component (in the phi space)
Z_grid = kpca.transform(X_grid)[:, 0].reshape(X1.shape)
plt.contour(X1, X2, Z_grid, colors='grey', linewidths=1, origin='lower')

plt.subplot(2, 2, 2, aspect='equal')
plt.scatter(X_pca[reds, 0], X_pca[reds, 1], c="red",
            s=20, edgecolor='k')
plt.scatter(X_pca[blues, 0], X_pca[blues, 1], c="blue",
            s=20, edgecolor='k')
plt.title("Projection by PCA")
plt.xlabel("1st principal component")
plt.ylabel("2nd component")

plt.subplot(2, 2, 3, aspect='equal')
plt.scatter(X_kpca[reds, 0], X_kpca[reds, 1], c="red",
            s=20, edgecolor='k')
plt.scatter(X_kpca[blues, 0], X_kpca[blues, 1], c="blue",
            s=20, edgecolor='k')
plt.title("Projection by KPCA")
plt.xlabel("1st principal component in space induced by $\phi$")
plt.ylabel("2nd component")

plt.subplot(2, 2, 4, aspect='equal')
plt.scatter(X_back[reds, 0], X_back[reds, 1], c="red",
            s=20, edgecolor='k')
plt.scatter(X_back[blues, 0], X_back[blues, 1], c="blue",
            s=20, edgecolor='k')
plt.title("Original space after inverse transform")
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")

plt.subplots_adjust(0.02, 0.10, 0.98, 0.94, 0.04, 0.35)

plt.show()
```

4 LLE

4.1 方法介绍

局部线性嵌入（Locally Linear Embedding, 简称 LLE）是流形学习的一种。与 Isomap 通过建立临近连接图再使用 MDS 方法降维的方法不同，LLE 方法通过保持邻域样本之间的线性关系降维。即一个样本 x_1 可以通过邻域样本 x_i, x_j, x_k 线性表示的话，那么经过 LLE 降维后，该关系将继续保持。这使得 LLE 算法对于卷曲的 manifold 数据的处理非常有效，尤其是当噪声较小的时候。

算法的主要过程可以总结如下：

1. 输入：样本集 $D = x_1, x_2, ..., x_m$，期望的低维空间维数 n，临近参数 k
2. for i = 1,2,...,m do
3. 确定 x_i 的 k 临近
4. 通过 $\min\limits_{1,...,m} \sum_{i=1}^{m}(\|x_i - \sum\limits_{j\in Q_i}(w_{ij}x_j)\|)^2$ 求得对 x_i 线性重构的系数 $w_i j$，$j \in Q_i$
5. 对于 $j \notin Q_i$，令 $w_{ij} = 0$
6. endfor
7. $M = (I - W)^T(I - W)$
8. 对 M 进行特征分解
9. 返回 M 的最小的 n 个特征值的特征向量
10. 输出: 样本集在特征向量对应的低维空间的投影

但由于 LLE 算法先要寻找 k 邻近 $O(mlog(m)nlog(k))$，再计算 wight 矩阵 $O(mnk^3)$，然后建立低维度的表示 $O(m^2)$。最后一项的计算复杂度 m^2 导致 LLE 对于大型的数据集的计算基本是无能为力的。

4.2 实验介绍

仿照 sci-kit 示例程序，本实验使用 scikit 的 datasets 库方法，生成经典的 manifolding 数据“瑞士卷”样本，在空间中呈现卷曲的状态。然后使用 LLE 方法对数据样本进行了处理，输出了在新的低维空间上的数据分布。可视化结果如下：

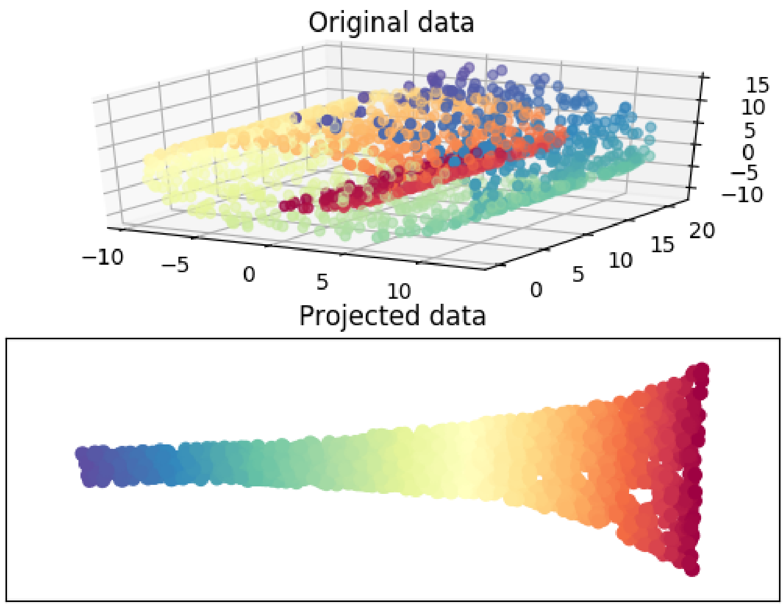

图 3: LLE 可视化结果

可以看出从图中看出，LLE 方法很好的实现了对 manifolding 数据在低维空间的展开。

4.3 源代码

实验使用的源代码如下.

```
"""
====================================
Swiss Roll reduction with LLE
====================================

An illustration of Swiss Roll reduction
with locally linear embedding
"""
print(__doc__)

import matplotlib.pyplot as plt

# This import is needed to modify the way figure behaves
from mpl_toolkits.mplot3d import Axes3D
Axes3D

#----------------------------------------------------------------------
# Locally linear embedding of the swiss roll

from sklearn import manifold, datasets
X, color = datasets.samples_generator.make_swiss_roll(n_samples=1500)

print("Computing LLE embedding")
X_r, err = manifold.locally_linear_embedding(X, n_neighbors=12,
```

```
                                          n_components=2)
print("Done. Reconstruction error: %g" % err)

#----------------------------------------------------------------------
# Plot result

fig = plt.figure()

ax = fig.add_subplot(211, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=color, cmap=plt.cm.Spectral)

ax.set_title("Original data")
ax = fig.add_subplot(212)
ax.scatter(X_r[:, 0], X_r[:, 1], c=color, cmap=plt.cm.Spectral)
plt.axis('tight')
plt.xticks([]), plt.yticks([])
plt.title('Projected data')
plt.show()
```

5 工程应用

5.1 背景介绍

前文的样本均为人为产生的，有很好的特征可供演示各类算法的特点。本节将对实际工程中得信号进行降维，以达到工程目的。

旋转机械属于能源与动力工程领域中使用较为广泛的机械设备，而转子是其核心部件。转子系统在运行过程中的振动信号会因裂纹故障发生显著的的变化。本节通过使用机器学习方法，对 Case Western Reserve University Bearing Data Center Website 上的转子故障数据，实现对转子故障的分析诊断。实验使用一台 2hp Reliance 电动马达进行，加速度数据在靠近和远离马达轴承的位置进行测量。使用电火花加工（EDM）在电机轴承上播种故障。本研究使用直径为 0.007 英寸到 0.04 英寸直径裂纹故障的转子数据。转子数据中包括出现故障的轴承以及正常的轴承安装到测试电机中，电机负载为 0 至 3 马力 (电机转速为 1797 至 1720 RPM）时的振动数据。共 9 个类别对应如下。'Normal': 0, 'B007': 1,'B014': 2,'B021': 3,'IR007': 4，'IR014': 5，'IR021': 6,'OR007@6': 7,'OR014@6': 8,'OR021@6': 9, 数字代表裂纹直径，如 007 为 0.007 英寸，字母 B，IR(inner race)，OR(outer race) 代表同一故障实验不同测点处的数据。

Fault Diameter	Motor Load (HP)	Approx. Motor Speed (rpm)	Inner Race	Ball	Outer Race Position Relative to Load Zone (Load Zone Centered at 6:00)		
					Centered @6:00	Orthogonal @3:00	Opposite @12:00
0.007"	0	1797	IR007_0	B007_0	OR007@6_0	OR007@3_0	OR007@12_0
	1	1772	IR007_1	B007_1	OR007@6_1	OR007@3_1	OR007@12_1
	2	1750	IR007_2	B007_2	OR007@6_2	OR007@3_2	OR007@12_2
	3	1730	IR007_3	B007_3	OR007@6_3	OR007@3_3	OR007@12_3
0.014"	0	1797	IR014_0	B014_0	OR014@6_0	*	*
	1	1772	IR014_1	B014_1	OR014@6_1	*	*
	2	1750	IR014_2	B014_2	OR014@6_2	*	*
	3	1730	IR014_3	B014_3	OR014@6_3	*	*
0.021"	0	1797	IR021_0	B021_0	OR021@6_0	OR021@3_0	OR021@12_0
	1	1772	IR021_1	B021_1	OR021@6_1	OR021@3_1	OR021@12_1
	2	1750	IR021_2	B021_2	OR021@6_2	OR021@3_2	OR021@12_2
	3	1730	IR021_3	B021_3	OR021@6_3	OR021@3_3	OR021@12_3
0.028"	0	1797	IR028_0	B028_0	*	*	*
	1	1772	IR028_1	B028_1	*	*	*
	2	1750	IR028_2	B028_2	*	*	*
	3	1730	IR028_3	B028_3	*	*	*

图 4: 故障信息表格

本研究以 0,1,2 马力负载时的时序信号作为训练集，3 马力负载时的时序信号作为测试集，通过使用 PCA 降维，然后使用的机器学习算法 Logistics Regression，实现 multi-label classification 任务，并对比降维前后的分类效果。

5.2 实验介绍

先对数据进行预处理，过程如下：

1. 读入数据 (load_data)
2. 切割数据 (wave_cut) 通过给定的时间窗大小 (2048) 和期望的 sample 数量 (256)，将一个时序信号切割成对应的 sample signal。
3. 对信号增加白噪声（add_noise）, 实现数据增强，使得切割得到的数据更贴近于实际信号。
4. 归一化（trans_norm）
5. FFT 快速傅立叶变换（trans_fft）实际是对数据的一次降维，将时序信号转化为有限个频域上的幅值。

处理后的信号如下图

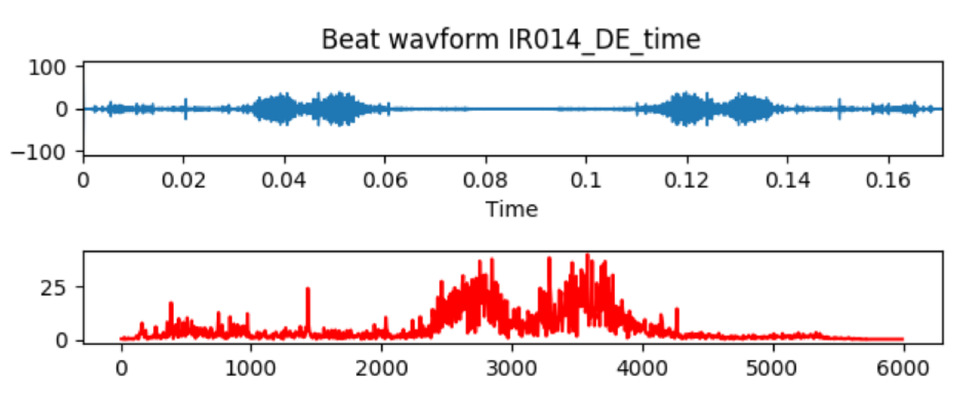

图 5: R014_DE 的时序信号以及频域信号

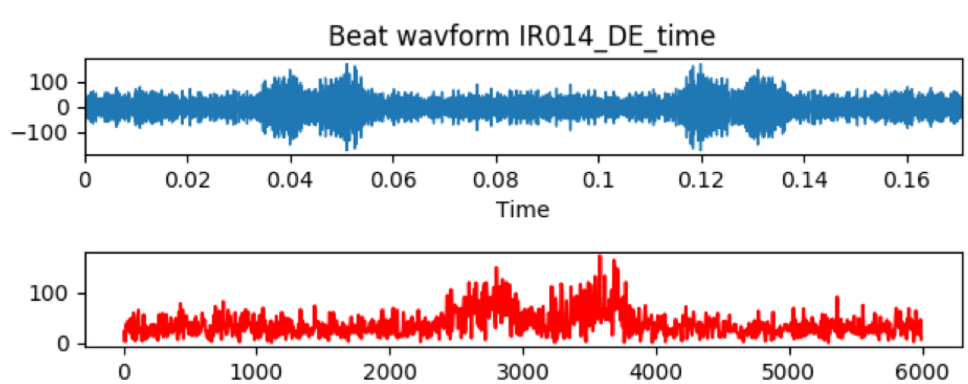

图 6: R014_DE 的时序信号以及频域信号

因为窗函数大小为 2048，所以经过傅里叶变换，信号频域大小为 1024。但是这些频率显然并不都是与故障相关的，直观上样本间方差较大的频率更有可能包含故障信息，因此对频域信号做 PCA （主成分分析）来进一步降维，通过多次试验，取低维空间维数 100，将振动数据转化为更低维子空间中信息更密集的样本点。然后分别使用 scikit 库中的 Logistics Regression 进行训练和分类。然后再使用未做 PCA 的样本进行训练和分类，得到实验结果。

PCA 前分类正确率为：0.8826

PCA 后分类正确率为：0.9004

可以看出，大幅压缩样本维度后，分类正确率并没有受到影响且有略微的提升。虽然分类正确率没有很高，但达到了基本的分类目的，用于工程实践还需要进一步的优化。

5.3 源代码

实验使用的源代码如下。（省略了 data_read 读入数据部分）

```
import numpy as np
from sklearn.model_selection import train_test_split
from data_read import *
import Config
from sklearn.multioutput import ClassifierChain
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import jaccard_similarity_score
from sklearn.linear_model import LogisticRegression

pca = PCA(n_components=100)
rf_per = np.empty([6,1],dtype=float)
knn_per = np.empty([6,1],dtype=float)
lg_per = np.empty([6,1],dtype=float)
rf_per_val = np.empty([6,1],dtype=float)
knn_per_val = np.empty([6,1],dtype=float)
lg_per_val = np.empty([6,1],dtype=float)
num = -1
#for nos in range(-10, 11, 4):
num = num+1
#print(nos)
nos=5
wave_train = WaveDate()
wave_train.load_data(class_name=cfg.CLASS_NAME, position_name=cfg.POSITION_NAME, load_name='1')
wave_train.wave_cut()
wave_train.add_noise(nos)
wave_train.trans_norm()
wave_train.trans_fft()
X1 = wave_train.data_cut
Y1 = wave_train.label

wave_train = WaveDate()
wave_train.load_data(class_name=cfg.CLASS_NAME, position_name=cfg.POSITION_NAME, load_name='2')
wave_train.wave_cut()
wave_train.add_noise(nos)
wave_train.trans_norm()
wave_train.trans_fft()
X2 = wave_train.data_cut
Y2 = wave_train.label

wave_train = WaveDate()
wave_train.load_data(class_name=cfg.CLASS_NAME, position_name=cfg.POSITION_NAME, load_name='0')
wave_train.wave_cut()
wave_train.add_noise(nos)
wave_train.trans_norm()
wave_train.trans_fft()
X0 = wave_train.data_cut
```

```
Y0 = wave_train.label



X = np.vstack((X1,X2,X0))
Y = np.hstack((Y1,Y2,Y0))
X = np.squeeze(X)



X=pca.fit_transform(X)
X_train, X_vali, Y_train, Y_vali = train_test_split(X, Y, test_size=0.2, random_state=42)

wave_test = WaveDate()
wave_test.load_data(class_name=cfg.CLASS_NAME, position_name=cfg.POSITION_NAME, load_name='3')
wave_test.wave_cut()
wave_test.add_noise(nos)
wave_test.trans_norm()
wave_test.trans_fft()
X_test = wave_test.data_cut
X_test = np.squeeze(X_test)
X_test=pca.transform(X_test)
Y_test = wave_test.label

# Fit an independent logistic regression model for each class using the
# OneVsRestClassifier wrapper.
base_lr = LogisticRegression(solver='lbfgs',max_iter=10000)
ovr = OneVsRestClassifier(base_lr)
ovr.fit(X_train, Y_train)

Y_pred_ovr = ovr.predict(X_vali)
ovr_jaccard_score = jaccard_similarity_score(Y_vali, Y_pred_ovr)
print (ovr_jaccard_score)

Y_pred_ovr = ovr.predict(X_test)
ovr_jaccard_score = jaccard_similarity_score(Y_test, Y_pred_ovr)
print (ovr_jaccard_score)
```

6 结论与讨论

实验的目的是通过使用代码，了解如何使用降维方法对数据进行降维，熟悉相关的编程环境以及调用方式。底层的算法实现并不是本实验的要求。

本文通过使用 Sci-kit learn python 库实现了三种机器学习算法的实验，尽管没有从最底层的代码进行实现，但是经过实验实现了三种降维算法的应用。PCA 属于线性降维方法，KPCA 属于非线性降维方法，LLE 属于非线性降维方法中的流形学习方法。并对一个工程实际数据进行了分析，实现了基本的分类目标。通过实际的实验，对于降维的意义通过可视化有了更加直观的认识。对于算法原理也有了更深刻的认识。降维的线性与非线性方法的区别与特点也在实验中有所体现。如果之后有时间，会花更多时间，从底层代码完成三种机器学习算法，而不是单纯调库。

报告写作过程中参考了周志华《机器学习》, 以及《Hands on machine learning with sci-kit learn》。代码 demo 部分来源于 scikit-learn 网站。

最后感谢孟教授这学期的教学，您的课干货满满，深入浅出。只是有些可惜课时太少，很希望未来机器学习这门课可以调整成全周的课程。顺颂时祺。