Q1 Report: Dimensionality Reduction

Author: YANG Rongfeng

Date: 2020-05-29

To complete the task, I implement two dimensionality reduction algorithms: PCA and LLE. The first step is to determine the ideal dimensions to reduce. However, the dataset is highly imbalanced with 61 positive samples and 3625 samples so I use SMOTE algorithm to resample dataset. After resampling, we get 3625 positive samples and 3625 negative samples.

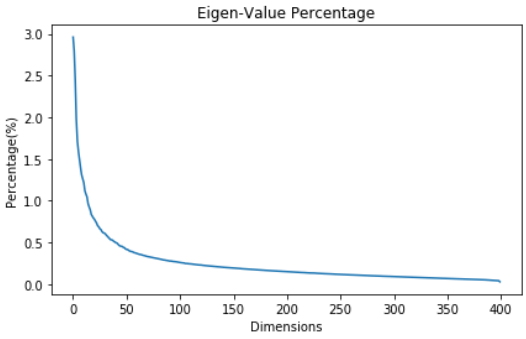
```
smote = SMOTE(random_state=0)
X_smote, y_smote = smote.fit_sample(X, y)
```

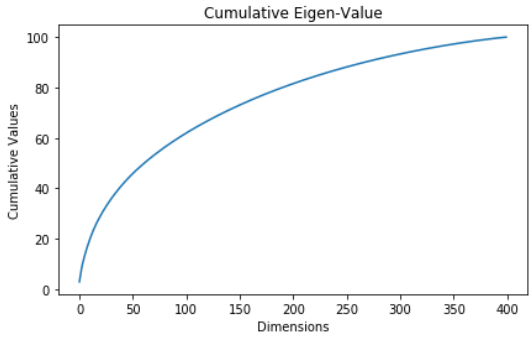
1. Find the ideal number of dimensions that reduce

At this step, we firstly split the dataset into training set and testing set. Then we compute the covariance matrix of the training data matrix after standardization. Next, we find all the eigenvalues and eigenvectors of the covariance matrix and sort the eigenvalues from big to small. We calculate the cumulative value of the following fraction for each ***i***

$$\frac{eigenvalue_i}{\sum_i eigenvalues_i}$$

Finally, we draw the curve for the cumulative eigenvalue and we can choose the ideal dimensions to reduce according to the graph.





In the right figure, we can see that if we want to keep about 90% of information, we can choose about **250 dimensions**.

2. Principal Component Analysis (PCA)

PCA, also called as principal component analysis, is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on.

PCA is either done by singular value decomposition of a design matrix or by doing the following 2 steps:

- calculating the data covariance (or correlation) matrix of the original data
- performing eigenvalue decomposition on the covariance matrix

The detailed steps of the PCA algorithm is shown as follows:

Input: A D-dimensional training set $X = \{x_1, x_2, ...x_N\}$ and the new lower dimensionality d ($d \leq D$)

Output: The approximated reduced $x_r \in \mathbb{R}^d$ for each $x \in \mathbb{R}^D$

1. Compute the mean $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$

2. Compute the covariance matrix $Cov(x) = \frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x_1}\right)\left(x_i - \bar{x_1}\right)^T$

3. Find the spectral decomposition of $Cov(X)$, obtaining the eigenvectors $\epsilon_1, \epsilon_2, \cdots \epsilon_D$ and their corresponding eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_D$. Note that the eigenvalues are sorted, such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$

4. For any $x \in \mathbb{R}^D$, its lowest dimensional representation is:

$$y = \left(\epsilon_1^T\left(x - \bar{x}\right), \epsilon_2^T\left(x - \bar{x}\right), \cdots, \epsilon_d^T\left(x - \bar{x}\right)\right)^T \in R^D$$

and the original x can be approximated as reduced x_r

$$x_r \approx \bar{x} + \left(\epsilon_1^T\left(x - \bar{x}\right)\epsilon_1 + \epsilon_2^T\left(x - \bar{x}\right)\epsilon_2 + \cdots + \epsilon_d^T\left(x - \bar{x}\right)\epsilon_d\right)\epsilon_d$$

Algorithm 1. The PCA algorithm

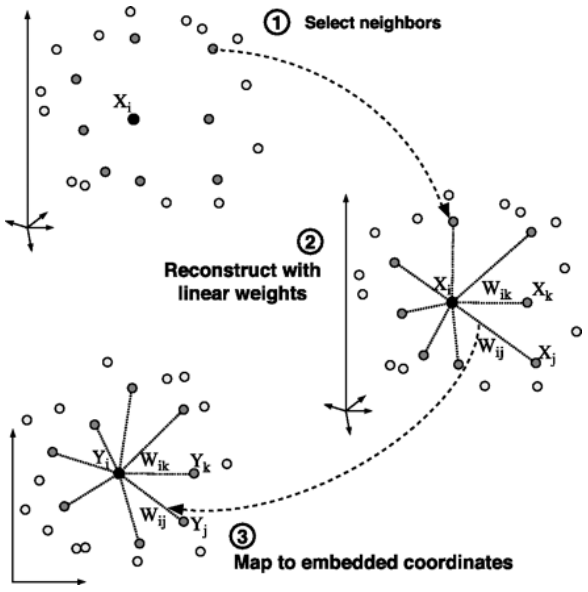
I test the effect of PCA mainly on classification task. Three different classifiers are used: Logistic Regression, LightGBM and Random Forest. The original dataset is split to two parts, 80% for training and 20% for testing. We will compare the performance in **Section 4.**

```
# PCA
pca = PCA(n_components=250)

# The detailed setting of the model
LR = LogisticRegression(penalty='l2', C=1, solver='lbfgs', multi_class='multinomial')
lgb = LGBMClassifier(boosting_type='gbdt', max_depth=None, learning_rate=1e-3,
                     n_estimators=5)
rf = RandomForestClassifier(n_estimators=5, random_state=0, max_depth=2)
```

3. Locally Linear Embedding (LLE)

Locally-Linear Embedding (LLE) was presented at approximately the same time as Isomap. It has several advantages over Isomap, including faster optimization when implemented to take advantage of sparse matrix algorithms, and better results with many problems. LLE also begins by finding a set of the nearest neighbors of each point. It then computes a set of weights for each point that best describes the point as a linear combination of its neighbors. Finally, it uses an eigenvector-based optimization technique to find the low-dimensional embedding of points, such that each point is still described with the same linear combination of its neighbors. LLE tends to handle non-uniform sample densities poorly because there is no fixed unit to prevent the weights from drifting as various regions differ in sample densities. LLE has no internal model.



The detailed steps of the LLE algorithm is below:

Input X: A *D*-dimensional training set $X = \{x_1, x_2, ...x_N\}$ and the new lower dimensionality d ($d \leq D$)

Output Y: d by N matrix consisting of d < D dimensional embedding coordinates for the input points.

1. Find neighbors in *X* spaces.
   for i=1:N
      Compute the distance from x_i to every point x_j, find the *K* smallest distances.
      Assign the corresponding points to be neighbors of x_i
   end

2. Solve for reconstruction weights W.
   for i=1:N
      Create matrix Z consisting of all neighbors of x_i

Subtract x_i from every column of Z
Compute the local covariance $C = Z^T Z$
Compute $W = C^{-1}$
Set $W_{ij} = 0$ if j is not a neighbor of i
Set the remaining elements in the i^{th} row of W equal to $\frac{w}{sum(w)}$

end

3. Compute embedding coordinates Y using weights W.
Create sparse matrix $M = (I - W)^T (I - W)$
Find the bottom $(d+1)$ eigenvectors of M
Set the q^{th} row of Y to be the $(q+1)$ smallest eigenvector

---

Algorithm 2. The LLE algorithm

Then I also use the same classifier models in PCA to test the performance of LLE. We will compare the performance in **Section 4.**

```
# LLE
lle = LocallyLinearEmbedding(n_components=250)

# The detailed setting of the model
LR = LogisticRegression(penalty='l2', C=1, solver='lbfgs', multi_class='multinomial')
lgb = LGBMClassifier(boosting_type='gbdt', max_depth=None, learning_rate=1e-2,
                     n_estimators=5)
rf = RandomForestClassifier(n_estimators=5, random_state=0, max_depth=2)
```

4 Experiment Results

The performance of two kinds of dimensionality reduction algorithms is shown in the table below. We can see that with same embedding dimensions (250), PCA performs better than LLA in the same classification task with same model parameters.

	Logistic Regression	LightGBM	Random Forest
PCA	0.54	0.75	0.85
LLE	0.56	0.59	0.58

Table. The accuracy on the testing dataset

To analyze the results, we must know the purpose of these two dimensionality reduction algorithms. LLE tries to keep the relationship of neighbors, which is a kind of basic manifold learning method. The example of LLE shows that when mapping the points from (A) to (C), the shape of points distribution is kept. However, if the data points in high dimensional space have no obvious manifold structure, it may not perform very well.

Moreover, when I implement the LLE algorithm, it'll produce complex number which make the classifier hard to handle. Therefore, I cast complex values to real by discarding the imaginary part and this will make some information loss.

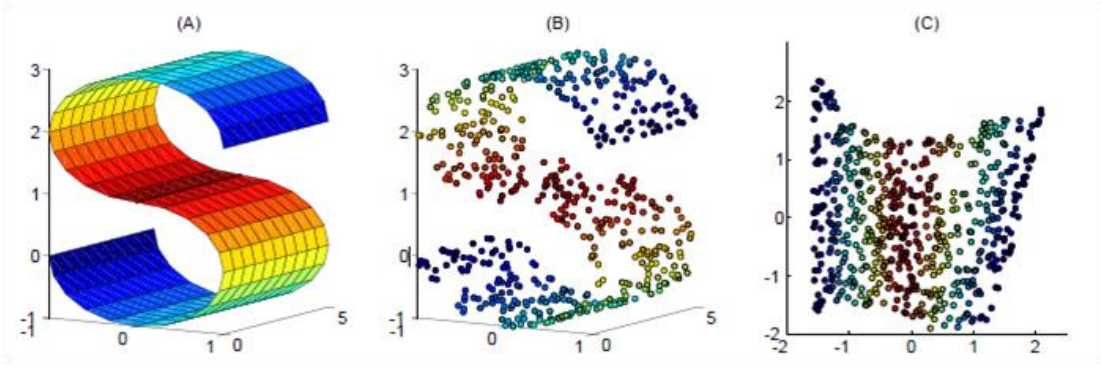


Figure. LLE example

Reference

[1] PCA 中降维数确定方法: http://blog.sina.com.cn/s/blog_628cc2b70102xq70.html
[2] python 大战机器学习——数据降维: https://www.cnblogs.com/acm-jing/p/7528874.html
[3] 非平衡数据的处理 SMOTE: https://www.cnblogs.com/xyou/p/9075443.html
[4] https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction#Locally-linear_embedding
[5] LLE Algorithm Pseudocode: https://cs.nyu.edu/~roweis/lle/algorithm.html
[6] 机器学习降维算法三：LLE (Locally Linear Embedding) 局部线性嵌入: https://www.cnblogs.com/xbinworld/archive/2012/07/09/LLE.html
[7] LLE 降维: http://www.luyixian.cn/news_show_304405.aspx