Dimension Reduction and Feature Selection

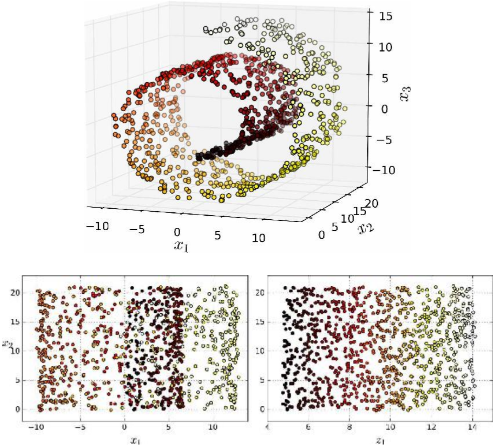
Elliott Ash

Text Data Course, Bocconi 2018

Dimensionality Reduction

- Especially in the case of text data, machine learning problems often involve thousands of features.
- Dimension reduction methods are needed:
    - not just for computational tractability, but also to help find a good solution.
    - also for data visualization – for example, to plot data in two dimensions.

Swiss role dataset example

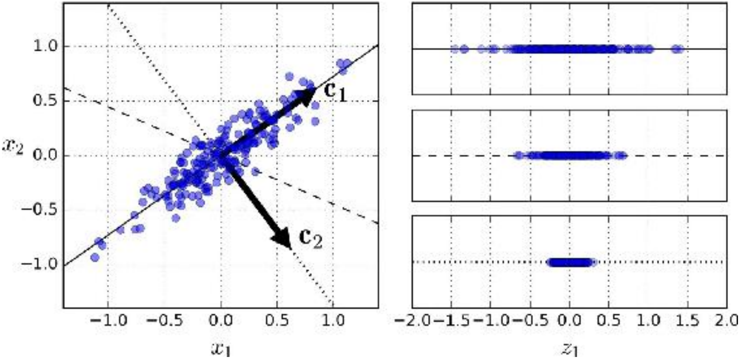


- The dimension reduction process matters: projecting down to two dimensions directly (left panel) might not isolate the variation we are interested in (as done in the right panel, which unrolls the Swiss Roll)

Manifold Learning

- The swiss roll is an example of a 2D manifold:
  - like many real-world data sets, the data are not uniformly distributed across the space.
  - can be modeled in a lower-dimensional subspace while retaining most of the information
- Dimension reduction methods in machine learning are motivated by this “manifold hypothesis.”

PCA



- PCA (principal components analysis), a popular dimension reduction technique.
  - identifies the axis that accounts for the largest amount of variance in the training set.
  - finds a second axis, orthogonal to the first, that accounts for the largest amount of the remaining variance, and so on
- The unit vector defining the *i*th axis is called the *i*th principal component.

PCA Projection

- For principal components $\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_n$, define

$$\boldsymbol{V}_n = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \boldsymbol{c}_1 & \boldsymbol{c}_2 & \dots & \boldsymbol{c}_n \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

- Dimension reduction works by projecting the data set down to the hyperplane defined by the first d principal components.

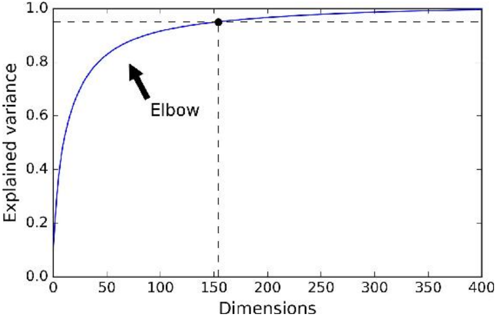
$$\boldsymbol{X}_{\text{PCA}} = \boldsymbol{X} \cdot \boldsymbol{V}_d$$

where the d subscript specifies that we have used the first d columns of $\boldsymbol{V}_n$

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2,svd_solver='randomized')
Xpca = pca.fit_transform(X)
pca.explained_variance_ratio_

plt.scatter(Xpca[:,0],Xpca[:,1], alpha=.1)
```

Choosing the number of dimensions



```
pca = PCA(n_components=.95)
X95 = pca.fit_transform(X)
pca.n_components_
```

PCA Inverse Transform

$$X_{\text{recovered}} = X_{\text{reduced}} \cdot \mathrm{V}_d'$$

```
Xrestore = pca.inverse_transform(X95)
plt.plot(Xrestore[0],X[0],'ro')
```

Incremental PCA and numpy memmap

- Standard PCA requires you to load the whole data set into memory.
- numpy memmap loads big arrays from disk as needed

```
X_mm = np.memmap('X.pkl',shape=(32567, 525))
```

- Incremental PCA splits the data into mini-batches and trains gradually.

```
from sklearn.decomposition import IncrementalPCA
inc_pca = IncrementalPCA(n_components=100,
                         batch_size=1000)
inc_pca.fit(X_mm)
```

Pros and Cons of PCA

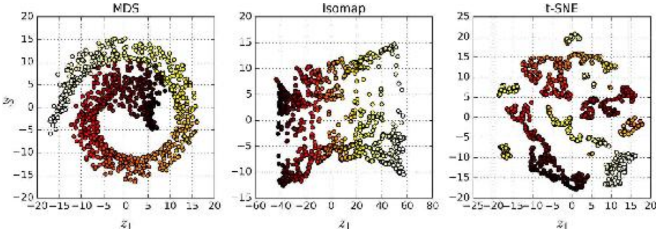
- Advantages:
  - fast to compute
  - good performance on many tasks in practice
  - components are orthogonal by construction
- Disadvantages:
  - lose (potentially a lot of) predictive information from X
  - Coefficients are not easily interpretable.
- Compromise:
  - keep strong predictors in feature set; take principal components of weak predictors

Principal Components Regression

- The "classic" way to deal with high-dimensionality.
  - Take the first few principal components of X and use those as predictors
  - Popular in macroeconomics and finance.

```
lin_reg = LinearRegression()
scores = cross_val_score(lin_reg,
                         X95[:,:10],
                         Y)
scores.mean(),scores.std()
```

Other Dimension Reduction Techniques



- **Multidimensional Scaling (MDS)** reduces dimensionality while trying to preserve the distances between the instances.
- **Isomap** creates a graph by connecting each instance to its nearest neighbors, then reduces dimensionality while trying to preserve the geodesic distances9 between the instances.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE)** reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. Useful for visualizing clusters of instances in high-dimensional space

Dimension Reduction for Visualization

```
from sklearn.manifold import MDS, Isomap, TSNE
mds = MDS()
iso = Isomap()
tsne = TSNE()

# these will be slow
Xmds = mds.fit_transform(X)
Xiso = iso.fit_transform(X)
Xtsne = tsne.fit_transform(X)
```

Partial Least Squares

- Partial Least Squares is another technique to determine linear combinations of the predictive variables.
- Unlike PCA, the PLS technique works by successively extracting factors from both predictive and target variables such that covariance between the extracted factors is maximized.

PLS Outline

- Assume X is a $n \times p$ matrix and Y is an nxq matrix (Y can include a multivariate response variable when $q > 1$).
- Find a linear decomposition of X and Y such that
  - $X = TPT + E$
    - T, X-scores ($n \times r$)
    - P, X-loadings ($p \times r$)
    - E, X-residuals ($n \times p$)
  - $Y = UQT + F$
    - U, Y-scores ($n \times r$)
    - Q, Y-loadings ($q \times r$)
    - F, Y-residuals ($n \times q$)
  - Covariance between T and U is maximized.

PLS Algorithm

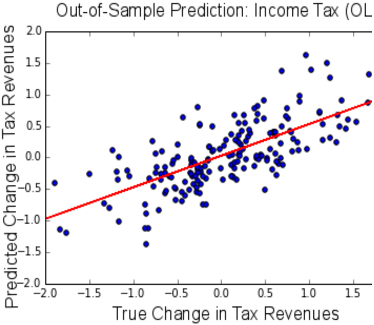
- First step:
  - first extracted x-score is $t = Xw$, where w is the eigen vector corresponding to the first eigen value of $X'YY'X$.
  - first y-score is $u = Yc$, where c is eigen vector corresponding to the first eigen value of $Y'XX'Y$.
  - note that $X'Y = Y'X = \text{Cov}[X, Y]$
- Next step:
  - Residualize original values of X and Y as $X_1 = X – tt'X$ and $Y_1 = Y - tt'Y$
  - Repeat first step on X_1 and Y_1 to get second components.
  - etc...

PLS with Scikit-learn

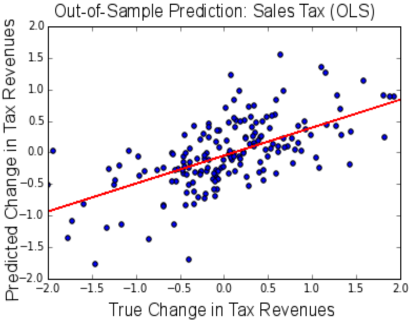
```python
from sklearn.cross_decomposition import PLSRegression
pls = PLSRegression(n_components=10)
Xpls, Ypls = pls.fit_transform(X,Y)
```

Ash 2016: PLS predictions of tax revenue changes using tax code text

Income Tax

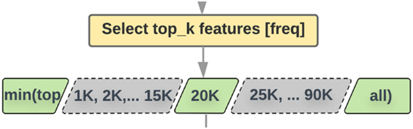


Sales Tax



Weak predictors filtered out; 80% training, 20% testing sample.

- Predicted change in revenue (vertical axis), plotted against true change in revenue (horizontal axis).
- Correlations between truth and prediction: 0.89 and 0.84.



- The Google text classification guide recommends a maximum 20,000 items in your feature set.

Feature selection using L1 Penalty

- A popular way to reduce dimensionality is to run lasso or elastic net, and exclude any predictors whose weights are regularized to zero.

```
from sklearn.linear_model import ElasticNet
enet_reg = ElasticNet(alpha=.1, l1_ratio=.5)
enet_reg.fit(X,Y)
nonzero = enet_reg.coef_ != 0
X_enet = X[:,nonzero]
```

Feature selection using univariate comparisions

- χ^2 is a very fast feature selection routine for classification tasks
  - features must be non-negative
  - works on sparse matrices

```
from sklearn.feature_selection import SelectKBest, chi2
select = SelectKBest(chi2, k=10)
X_new = select.fit_transform(X, Y)
[vocab[i] for i in np.argsort(select.scores_)[:10]]
```

- With negative predictors, use `f_classif`.
- For regression tasks, use `f_regression`.
- There are also `mutual_info_regression` and `mutual_info_classif`, which will recover non-linear relations between predictor and outcome
  - they are much slower, can run on a sample of the data.

Graph-based feature selection

- Graph-based feature selection works by clustering collinear features and then iteratively removing features.
  - much slower than univariate comparisons, but probably gives better performance in text classification tasks.
  - See Zhang and Hancock.