Project 6 Report : Dimensionality reduction techniques

Sajal Kumar

Implementation details on parameters

My implementation uses all 3 methods : Principal Component Analysis, Linear Discriminant Analysis and Kernel Principal Component Analysis on their default setting except the following general parameters that could be changed:

- `random_state` : Random seed (set to 1 by default).
- `n_jobs` : Number of parallel threads allowed (set to 4 by default).

I also allows changes to the `gamme` parameter in Kernel PCA, denoted by `gamma_kernel_pca` in the cod (set to 1/n_features by default).

Apart from the above mentioned parameter some other dimensionality reduction technique specific parameters were changed but were not provided to the user:

- PCA
  - `n_components` was set to 'mle'.
  - `svd_solver` was set to 'full'.
- Kernel PCA
  - `kernel` was set to 'rbf'.

Additionally, I implemented a `var_explained` $\in [0, 1]$ parameter that is used to determine the number of transformed attributes (principal components). It was set to 0.9 by default.

The above represents the 'standard' setting for all regression methods when no parameter is changed.

Logistic regression was used for the classification task. I used it at its default setting except :

- `fit_intercept` was set to False.
- `solver` was set to 'sag'.
- `multi_class` was set to 'ovr'.

`var_explained` for Kernel PCA

Both PCA and LDA provide access to the member variable `explained_variance_ratio_`, however, Kernel PCA doesn't as it is no longer in the same space as the input X. After some online search, I found that `numpy.var` could be used to compute the variance of each transformed attribute and then `explained_variance_ratio_`. Hence, I wrote a tiny script to calculate the variance related to quantities so that I could evaluate the 3 methods with limited bias.

Performance evaluation

I computed the runtime, prediction on training data, prediction on testing data and dimension usage on the original data as well as the three (PCA, LDA, Kernel PCA) transformed data to judge the quality of results before and after dimensionality reduction. Dimension usage $\in [0, 1]$ was added to note the percentage of dimensions used to achieve `var_explained` threshold. Since Kernel PCA is in a different space, the dimension usage for it was found to be greater than 1 for both data-sets.

Performance on Iris Data-set

Info	**Og**	**PCA**	**LDA**	**KernelPCA**
runtime	NA	0.001	0.002	1.83
pred on train	0.87	0.84	0.66	0.99
pred on test	0.8	0.73	0.66	0.86
dim usage	1	0.5	0.25	2.25

Table 1: Result on 'Iris' data-set with 'standard' configuration of original and transformed inputs.

Table 1 shows the runtime (in seconds), prediction on training data, prediction on testing data and dimension usage for 'Iris' data-set using the 'standard' configuration on the original (Og) and transformed dataset. The data-set was scaled using the 'StandardScaler' method from sklearn. The data was split into 70% training and 30% testing using stratified partition. LDA performs the worst and Kernel PCA does the best, however, if we look at the dimension usage, it can be seen that Kernel PCA uses way a lot more dimensions, which may explain the improved performance.

Figure 1 shows the top two principal components for each method. It can be seen that while PCA and LDA do a good job capturing more than 95% (≥ 0.95) variance, Kernel PCA can only capture 60% variance. This clearly means that if one were to fix the number of principal components to two, Kernel PCA would show the worst performance.

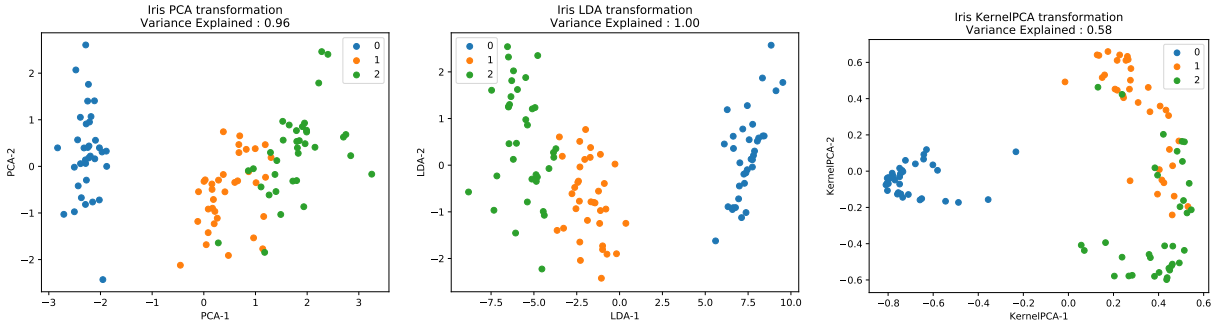

Figure 1: Top two principal components for each method.

Info	Og	PCA	LDA	KernelPCA
runtime	NA	0.14	0.025	2.78
pred on train	0.93	0.90	0.78	0.97
pred on test	0.91	0.90	0.76	0.96
dim usage	1	0.48	0.09	5

Table 2: Result on 'Digits' data-set with 'standard' configuration of original and transformed inputs.

Performance on Digits Data-set

Table 2 shows the runtime (in seconds), prediction on training data, prediction on testing data and dimension usage for 'Digits' data-set using the 'standard' configuration on the original (Og) and transformed dataset. The data-set was scaled using the 'StandardScaler' method from sklearn. The data was split into 70% training and 30% testing using stratified partition. Again LDA performs the worst and Kernel PCA does the best. This time around perhaps some explanation can be offered about LDA's poor performance, if we look at the dimension usage, LDA uses less than 1% of the dimension, (even PCA uses almost 50% of the dimensions) to achieve the target variance. Having less number of dimensions is probably what is hurting LDA as it is benefiting Kernel PCA that causes 5 times increase in dimension.

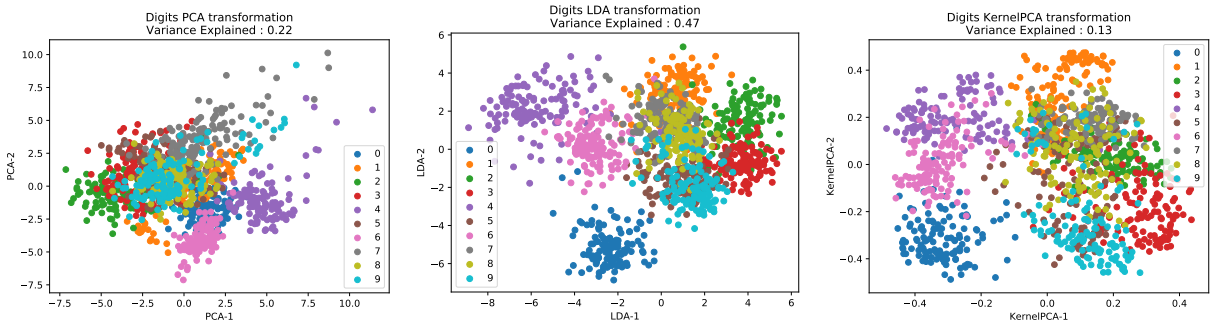

Figure 2: Top two principal components for each method.

Figure 2 shows the top two principal components for each method. No method could capture more than 90% variance in the data set, however, LDA already captured about 50% of the variance using just two components. Similar to that in the case of 'Iris', Kernel PCA reports the least variance explanation score with two components in the 'Digits' dataset. Again this clearly means that fixing the number of principal components would degrade Kernel PCA's performance.

A more comprehensive evaluation

Since most of the parameters were automatically computed, the only parameter I could have changed was `var_explained`, while, there were no major change in the results when `var_explained` was set to 0.98 for 'Iris' data-set. LDA saw a major improvement in performance on the 'Digits' data-set.

Info	**Og**	**LDA-0.9**	**LDA-0.98**
runtime	NA	0.025	0.026
pred on train	0.93	0.78	0.93
pred on test	0.91	0.76	0.91
dim usage	1	0.09	0.14

Table 3: Improvement in LDA's performance for 'Digits' data-set with `var_explained` set to 0.98

Table 3 shows the performance improvement for LDA, wherein, it is now using 14% of the dimensions. Interestingly, for `var_explained`=0.98, the accuracy for LDA exceeded that of PCA by 1.5% on both training and testing.