WORLDQUANT UNIVERSITY

FINANCIAL ENGINEERING

GROUP WORK PROJECT

FEATURE SELECTION AND ENGINEERING

Group members:

Name	Email	Non-contribution
Quyen Ho Thanh	thquyen11@hotmail.com	
Wei Hao Lew	lewweihao93@hotmail.com	
Lu Liu	liulu0502@gmail.com	
Truong Nguyen	nmtruong93@gmail.com	
Yernur Orakbayev	yernur.orakbayev@nu.edu.kz	

June 2020

Contents

1. Background

The aim of this report is to explain the cross-validation techniques for forecasting financial time series and determine which features are helpful in predicting the target variable. The research covers the following terms:

- PCA
- K-fold cross-validation
- Walk forward analysis
- Purged K-Fold CV

In this report, we will also investigate feature engineering techniques such as PCA and present its implementation in Python.

2. Research

2.1. PCA

Optimize the objective function

Dimensionality reduction plays a crucial role in machine learning field when deal with large data. In the real world, feature vectors as well as number of data can be large which cause difficult in storage and computation performance. Therefore, dimensionality reduction is an important step in many problems solving.

Dimensionality reduction, in simple words, is looking for a way that takes an input data $x \in \mathbb{R}^D$ with dimension D very large and transform into a new data point $z \in \mathbb{R}^K$ that has lower dimension K < D. The simplest method in Dimensionality Reduction algorithm based on linear model, called Principle Component Analysis (PCA). The simplest way to decrease data dimension from D to K < D is just hold on K the most important element.

However, this is definitely not the best job because we do not know which components are more important. Or, in the worst-case scenario, the amount of information each component carries are the same, removing any component will result in the loss of a large amount of information. Nevertheless, if we can represent the original data vectors in a new coordinate system in which the importance between the components is clearly different, then we can ignore the least important components.

PCA is a method of searching a new system so that the data's information is mainly concentrated in several coordinates, the rest only brings an insignificant amount of information. And for simplicity in calculation, PCA will find standard orthonormal system as a new coordinate system.

Suppose the new orthonormal base system is **U** and we want to keep it **K** coordinates in this new base system. Without loss of generality, suppose that is **K** first components. Observe figure below:

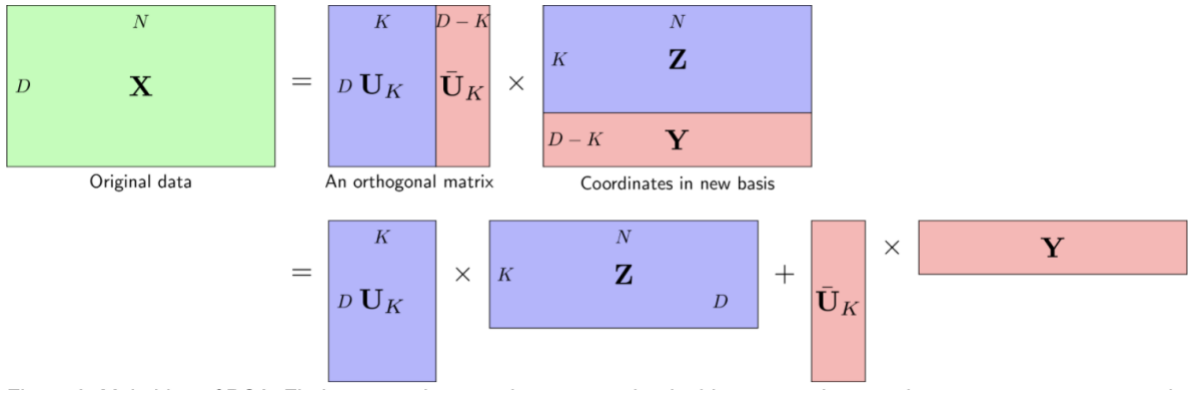


Figure 1. Matrix factorization in PCA *(Tiep, 2018)*

The figure above allows us writing the data matrix:

$$\boldsymbol{X} = \boldsymbol{U_K Z} + \overline{\boldsymbol{U_K}}\boldsymbol{Y} \tag{1}$$

From this we also offer:

$$\begin{bmatrix} \boldsymbol{Z} \\ \boldsymbol{Y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{U_K^T} \\ \overline{\boldsymbol{U_K^T}} \end{bmatrix} \boldsymbol{X} \Longrightarrow \begin{cases} \boldsymbol{Z} = \boldsymbol{U_K^T X} \\ \boldsymbol{Y} = \overline{\boldsymbol{U_K^T}} \boldsymbol{X} \end{cases} \tag{2}$$

The purpose of the PCA is to find orthogonal matrices **U** so that most of the information is kept in green $\boldsymbol{U_K Z}$ and the red part $\overline{\boldsymbol{U}}_{\boldsymbol{K}}\boldsymbol{Y}$ be omitted and replaced with a matrix regardless of the data points. In other words, we will approximate **Y** by a matrix whose entire columns are the same. Call each column that is **b** and can be considered a bias, then we will approximate:

$$\boldsymbol{Y} \approx \boldsymbol{b}\mathbf{1}^T$$

where $\mathbf{1}^T \in \mathbb{R}^{1xN}$is a row vector with all element equal to **1**. Suppose it has seen found **U**, we need to find **b** satisfy:

$$\boldsymbol{b} = argmin_b \parallel \boldsymbol{Y} - \boldsymbol{b}\mathbf{1}^T \parallel_F^2 = argmin_b \parallel \overline{\boldsymbol{U_K^T}}\boldsymbol{X} - \boldsymbol{b}\mathbf{1}^T \parallel_F^2$$

Solve the derivative equation respect to **b** of the target function is 0

$$\left(\boldsymbol{b}\mathbf{1}^T - \overline{\boldsymbol{U_K^T}}\boldsymbol{X}\right)\mathbf{1} = 0 \Longrightarrow N\boldsymbol{b} = \overline{\boldsymbol{U_K^T}}\boldsymbol{X}\mathbf{1} \Longrightarrow \boldsymbol{b} = \overline{\boldsymbol{U_K^T}}\overline{\boldsymbol{x}}$$

where $\mathbf{1}^T\mathbf{1} = N$ and $\overline{\boldsymbol{x}} = \frac{1}{N}\boldsymbol{X}\mathbf{1}$ is a mean vector of the whole columns in **X**.

with have **b**, the original data will be approximated with:

$$\boldsymbol{X} = \boldsymbol{U_K Z} + \overline{\boldsymbol{U_K}}\boldsymbol{Y} \approx \boldsymbol{U_K Z} + \overline{\boldsymbol{U_K}}\boldsymbol{b}\mathbf{1}^T = \boldsymbol{U_K Z} + \overline{\boldsymbol{U_K}}\overline{\boldsymbol{U_K^T}}\overline{\boldsymbol{x}}\mathbf{1}^T \approx \widehat{\boldsymbol{X}} \tag{3}$$

From (1), (2), (3) we define the loss function of PCA as an error of approximation

$$\boldsymbol{J} = \frac{1}{N} \parallel \boldsymbol{X} - \widehat{\boldsymbol{X}} \parallel_F^2 = \frac{1}{N} \parallel \overline{\boldsymbol{U_K}}\boldsymbol{Y} - \overline{\boldsymbol{U_K}}\overline{\boldsymbol{U_K^T}}\overline{\boldsymbol{x}}\mathbf{1}^T \parallel_F^2 = \frac{1}{N} \parallel \overline{\boldsymbol{U_K}}\overline{\boldsymbol{U_K^T}}\boldsymbol{X} - \overline{\boldsymbol{U_K}}\overline{\boldsymbol{U_K^T}}\overline{\boldsymbol{x}}\mathbf{1}^T \parallel_F^2$$

$$= \frac{1}{N} \| \overline{\boldsymbol{U_K}} \overline{\boldsymbol{U}}_{\boldsymbol{K}}^{\boldsymbol{T}} (\boldsymbol{X} - \bar{\boldsymbol{x}} \mathbf{1}^T) \|_F^2 \quad (4)$$

Notice that, if the columns of a matrix **V** form an orthonormal system then with a matrix **W**, we always have:

$$\| \boldsymbol{VW} \|_F^2 = trace(\boldsymbol{W}^{\boldsymbol{T}} \boldsymbol{V}^{\boldsymbol{T}} \boldsymbol{VW}) = trace(\boldsymbol{W}^{\boldsymbol{T}} \boldsymbol{W}) = \| \boldsymbol{W} \|_F^2$$

Therefore, the loss function (4) in can rewrite to:

$$\boldsymbol{J} = \frac{1}{N} \| \overline{\boldsymbol{U}}_{\boldsymbol{K}}^{\boldsymbol{T}} (\boldsymbol{X} - \bar{\boldsymbol{x}} \mathbf{1}^T) \|_F^2 = \frac{1}{N} \| \overline{\boldsymbol{U}}_K^T \widehat{\boldsymbol{X}} \|_F^2 = \frac{1}{N} \| \widehat{\boldsymbol{X}}^T \overline{\boldsymbol{U}}_K \|_F^2 = \frac{1}{N} \sum_{i=K+1}^{D} \| \widehat{\boldsymbol{X}}^T \boldsymbol{u}_i \|_2^2$$

$$= \frac{1}{N} \sum_{i=K+1}^{D} \boldsymbol{u}_i^T \widehat{\boldsymbol{X}} \widehat{\boldsymbol{X}}^T \boldsymbol{u}_i = \sum_{i=K+1}^{D} \boldsymbol{u}_i^T \boldsymbol{S} \boldsymbol{u}_i \quad (5)$$

with $\widehat{\boldsymbol{X}} = \boldsymbol{X} - \bar{\boldsymbol{x}} \mathbf{1}^T$ is data normalized and with **S** is the covariance matrix of data. We call this matrix $\widehat{\boldsymbol{X}}$ is zero-corrected data *or* normalized data. May notice $\hat{x}_n = x_n - \bar{x}_j \ \forall n = 1, 2, \dots, N$.

The remaining job is to find the $\mathbf{u_i}$ so the loss is minimal. First of all, we have an interesting comment. Recall the definition of covariance $\boldsymbol{S} = \frac{1}{N} \widehat{\boldsymbol{X}}^T \widehat{\boldsymbol{X}}$. With matrix **U** orthogonal, substitute K= 0 into the (5), we have:

$$\boldsymbol{L} = \sum_{i=1}^{D} \boldsymbol{u}_i^T \boldsymbol{S} \boldsymbol{u}_i = \frac{1}{N} \| \widehat{\boldsymbol{X}}^T \boldsymbol{U} \|_F^2 = \frac{1}{N} trace\left(\widehat{\boldsymbol{X}}^T \boldsymbol{U} \boldsymbol{U}^T \widehat{\boldsymbol{X}}\right) = \frac{1}{N} trace\left(\widehat{\boldsymbol{X}} \widehat{\boldsymbol{X}}^T\right) = trace(\boldsymbol{S}) = \sum_{i=1}^{D} \lambda_i$$

With $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$ are positive values of positive semi-determinant matrices S. Note that these specific values are real and non-negative.

So **L** independent of the choice of orthogonal matrices **U** and equal to the sum of the elements on the diagonal of **S**. In other words, L is the sum of the variances according to each component of the original data.

So the minimum function loss **J** is given by (5) equivalent to the max:

$$\boldsymbol{F} = \boldsymbol{L} - \boldsymbol{J} = \sum_{i=1}^{D} \boldsymbol{u}_i \boldsymbol{S} \boldsymbol{u}_i^T \quad (6)$$

If **S** is a positive semi-determinant matrix, the optimal problem

$$\max_{U_K} \sum_{i=1}^{K} \boldsymbol{u}_i^T \boldsymbol{S} \boldsymbol{u}_i$$

satisfy $\boldsymbol{U}_K^T \boldsymbol{U}_K = \boldsymbol{I}$

when $u_1, \dots, u_K$ are eigenvectors corresponding to K largest eigenvalues of **S**. At that time, value of objective function is $\sum_{i=1}^{K} \lambda_i$ with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$ are eigenvalues of **S**.

Greatest eigenvalue λ_1 of the covariance matrix **S** is also known as the first principal component, the second eigenvalue λ_2 also known as the second principal component, etc. We just keep it K main component when apply dimensionality reduction using PCA.

Step to implement PCA

1. Calculate the expected vector of all data:

$$\overline{\boldsymbol{x}} = \frac{1}{\boldsymbol{N}} \sum_{n=1}^{\boldsymbol{N}} \boldsymbol{x}_n$$

2. Subtract each data point to the expected vector of all data:

$$\widehat{\boldsymbol{x}}_n = \boldsymbol{x}_n - \overline{\boldsymbol{x}}$$

3. Set $\widehat{\boldsymbol{X}} = [\widehat{\boldsymbol{x}}_1, \widehat{\boldsymbol{x}}_2, \dots, \widehat{\boldsymbol{x}}_D]$ is the normalized data matrix, calculate covariance matrix

$$\boldsymbol{S} = \frac{1}{\boldsymbol{N}} \widehat{\boldsymbol{X}} \widehat{\boldsymbol{X}}^T$$

4. Calculate the eigenvalues and eigenvectors with norm equal to 1 of this matrix, arrange them in descending order of eigenvalues.
5. Choose K eigenvector corresponding to K largest eigenvalues for matrix construction UK whose columns form an orthogonal system. K these vectors, also known as principal components, form a subspace *close* to the normalized distribution of the original data.
6. Project standardized data $\widehat{\boldsymbol{X}}$ into the subspace.
7. New data is the coordinates of the data points on the new space

$$\boldsymbol{Z} = \boldsymbol{U}_K^T \widehat{\boldsymbol{X}}$$

The original data can be approximated according to the new data as follows:

$$\boldsymbol{x} = \boldsymbol{U}_K \boldsymbol{Z} + \overline{\boldsymbol{x}}$$

The step for implementing PCA can be seen in the figure below:

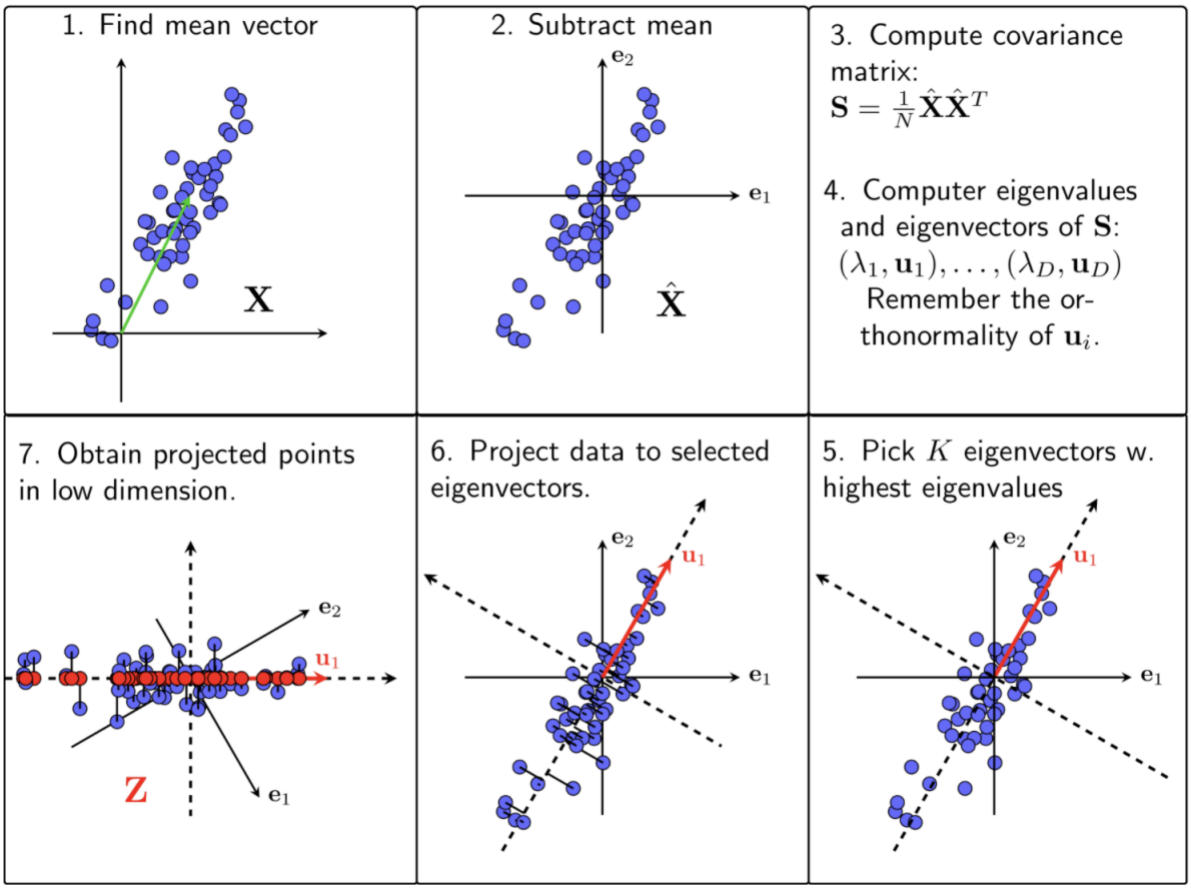


Figure 2. Step to implement PCA *(Tiep, 2018)*.

2.2. K-Fold cross-validation

k-Fold is one of the cross-validation techniques to measure the accuracy of our machine learning model. The fundamental idea is split our data into the training set and the testing set, then train our model on the train set and test it with the unseen data of the test set, depending on the performance of the model on the test data, we can determine if our machine learning model is under-fitting, over-fitting or well-fitting.

We can say that k-Fold was built on top of the traditional Train-Test Split approach. The main disadvantage of Train-Test Split is that considering the small sample of data doesn't have the same distribution, this technique will have a high possibility of bias because we would miss the information that is not used for training. To overcome this kind of issue, k-Fold proposes the new way to validate data as below:

1. Shuffle the entire dataset to split it into k folds (k-1 folds for training, and one-fold for testing purposes). The k value is chosen large enough to be statistically representative of the broader dataset. Normally, it's from 5 to 10 as a rule of thumbs.
    - The lower value of k will lead us to a similar bias problem in Train Test Split.

- The higher value of k might cause over-fitting because we over-train the model in the same training data over and over again.

2. Fit the machine learning model using the training set, then test it with the testing set. Record the scores.
3. Repeat steps 1 and 2 until all the folds are used as a testing set. Record the scores.
4. Calculate the performance metric of the machine learning model by taking the average of recorded scores in previous steps.

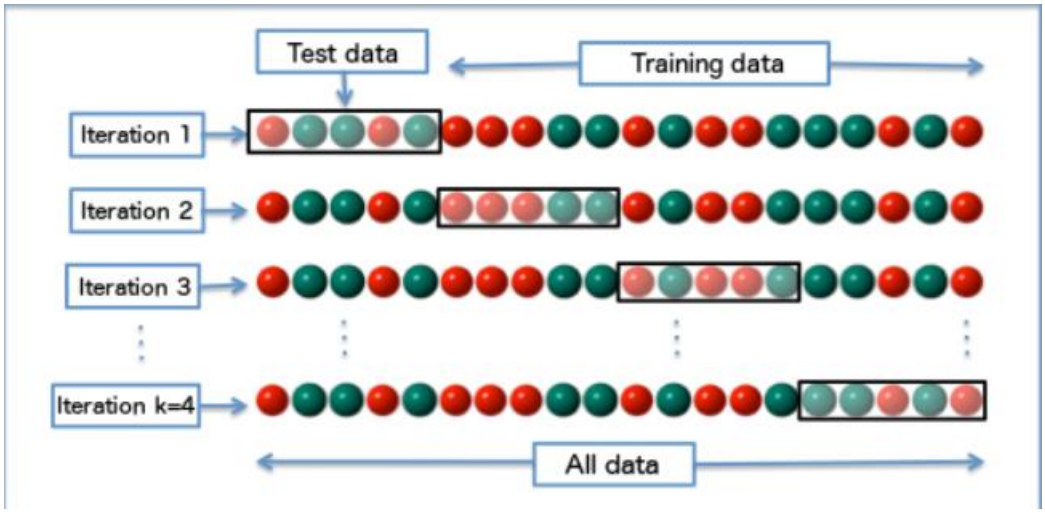


Figure 3. K-Fold Cross-Validation Process

2.3. Walk forward analysis

The book of the Evaluation and Optimization of Trading Strategies was the first to cover the walk forward analysis ("WFA"), which was written by Robert E. Pardo. Walk forward analysis provides an opportunity to optimize an existing trading strategy by dividing the data into in sample data (for initial testing) and out of sample data (unknown and non-optimized data for validation) and taking a segment of the data to optimize a strategy or validate the strategy. It can also be utilized to gain more complete understanding of its forecasting ability.

The basic principle of the walk forward analysis is intuitive. Analysts first use a set of in-sample data (or past returns) to perform optimization tests and run the best optimized values of parameters. Normally these parameters will influence the trading performance and optimal ones would produce largest gain or favorable return. The next step is to apply the parameters to the out-of-sample testing and verify the performance by applying it forward on data following the optimization part, and the process will be repeated over subsequent time segments.

The table below shows how walk forward analysis works (e.g. 5 periods in the following example). Each period consists of optimization segment and running process with the following steps:

- Identify the "in-sample" data (e.g. the historical stock price) in a certain timeframe (dark blue section below) and define the "out-of-sample" data in a period
- Use "in-sample" data to perform optimization tests on *optimization period 1* to get the best parameters based on specifications
- Apply the parameters to the *run period 1*, which is also the "out-of-sample" period data (light blue below)
- Put this "out-of-sample" result to the overall WFA result
- Move to the *optimization period 2* and find the best set of parameters used for *running period 2* data
- Continue the above process until the period 5 to obtain the relative realistic results from out of sample data

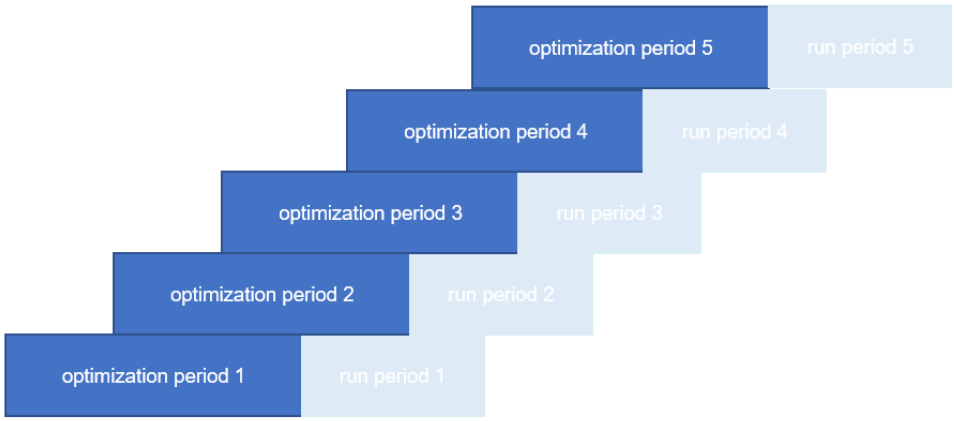


Figure 4. How Walk-Forward optimization works

Walk forward analysis is a powerful and reliable backtesing method to analyze, evaluate and improve the long-term performance or profitability of a trading strategy since this method optimize the trading rule parameters based on existing historical data. In addition, the walk forward analysis employs multiple smaller backtests to split the optimization periods to produce precise optimization segment to test the predictability and robustness of the model.

Walk Forward Analysis results can also be viewed in spreadsheet or a 3D picture to present the correlations between in and out sample variations. However, since the essential of walk forward technique is to optimize the parameters on a past segment of data, it may encounter overfitting problems.

2.4. Purged k-fold cross validation

One of the main issues with inappropriate work of cross validation in finance is than given time-series observations are time-ordered and not independently and identically distributed. In reality, past outcomes used to predict future outcomes, while randomization in cross-validation disrupts order and sometimes trains model on future outcomes. Moreover, there is increase of errors created by selection bias as well as multi-testing because of iterative usage of same folds to train and test.

Thus, it leads to "leakage" that occurs as a result of common information both in test and train fold in cross-validation process. It happens only when both independent variable X at time T and T+1 has serial correlation, and labels of overlapping data Y at time T and T+1 are used. In such case, when this observation is placed in different sets, predictive power of model trained at T to compute T+1 is increased in any situation and may result in false discoveries, when independent variable is inappropriate, or overestimate impact of appropriate independent variable.

In order to decrease the probability of leakage, we need to drop train set observations that have overlapping periods with test set observations or/and try to prevent overfitting of classifier.

Purged K-Fold Cross-Validation, created by Lopez de Prado, deals with leakages problem by processes of "purging" and "embargo". The former implies removing (or "purging") time overlapping labels within observations of training set with labels these of testing set. It relates to both left(beginning) and right(end) side overlapping of train set with test set. In most scenarios, implementation of purging is enough to avoid leakage.

Nevertheless, in scenarios when usage of purging is not sufficient to avoid all leakages, Lopez de Prado suggest to implement embargo, which is simply removing of observations of training set that take place directly after all test sets, i.e. right(end) side of test sets. It is justified because we get rid of signals created by test and transferred to train set. Embargo process do not affect left(beginning) of train set as our training labels Y included already available information before testing, and, thus, are valid.

Finally, in any train and test split performed on overlapping training observations, "purge" and "embargo" processes are suggested to be implemented. The reason is that both processes are valid for evaluation of model performance, back testing, as well as hyper-parameter fitting.

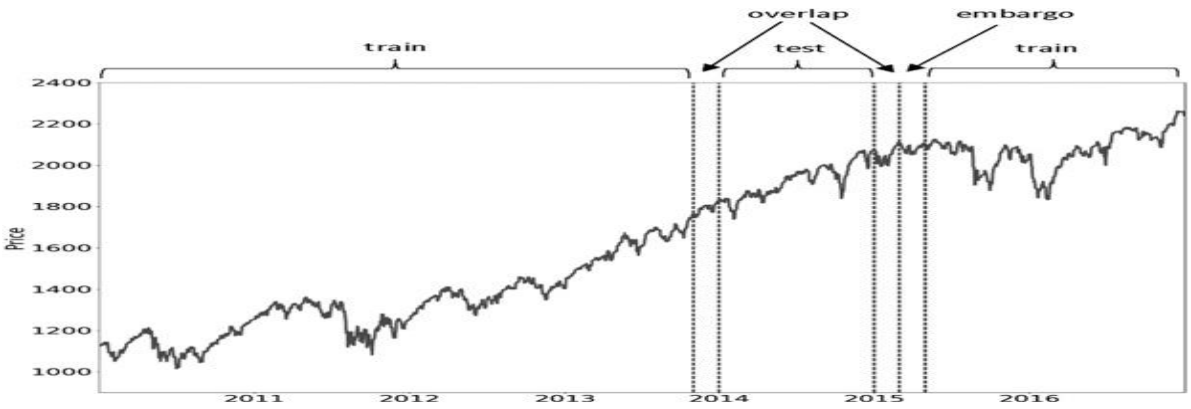


Figure 5. Purging and embargo processes illustrated by Lopez de Prado.

3. Results

Import vital libraries.

```
[1]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import yfinance as yf
     from sklearn.decomposition import PCA
     from sklearn.preprocessing import StandardScaler
```

In this report, we are going to make use of the following 4 explanatory variables that can possibly explain the movement of any US stock, moving average of the nasdaq index, rolling volatility of nasdaq, federal interest rate and mortgage loan rates. The federal interest rate and mortgage loan rates are retrieved from Quandl.com.

Download Nasdaq index data, then calculate 50 days rolling of moving average and volatility.

```
[2]: start_date = '2014-05-08'
     end_date = '2020-05-08'

     inner_start = '2017-01-01'
     inner_end = '2017-10-01'

     nasdaq = yf.download('^IXIC', start=start_date, end=end_date, progress=False).rename(columns={'Adj Close': 'Adj_Close'})
     nasdaq.sort_index(inplace=True)

     ma = nasdaq.Adj_Close.rolling(window=50).mean().fillna(method='pad')
     vol = nasdaq.Adj_Close.rolling(window=50).std().fillna(method='pad')
```

Read federal interest rate and mortgage loan rate data. We only use these rate data in 10 moth which is ranging from 2017-01-01 to 2017-10-01.

[3]:
```
# Read federal interest rate
rates = pd.read_csv("FRED-DTB3.csv")
rates['Date'] = pd.to_datetime(rates['Date'])
mask = (rates['Date'] > inner_start) & (rates['Date'] <= inner_end)
rates = rates.loc[mask]
# Read mortgage loan rate
mortgage_rates = pd.read_csv("mortgage.csv")
mortgage_rates['Date'] = pd.to_datetime(mortgage_rates['Date'])
mask = (mortgage_rates['Date'] > inner_start) & (mortgage_rates['Date'] <= inner_end)
mortgage_rates = mortgage_rates.loc[mask]
```

After we load all the data required, we need to do some formatting. Firstly, we need to restrict the data to only a time period which we will be using to train the model. In this case, we will be using a 10-month period from 2017-01-01 to 2017-10-01.

Additionally, because not all the dates will be consistent among the different data sources, we need to filter all 4 datasets such that every dataset will contain exactly the same dates.

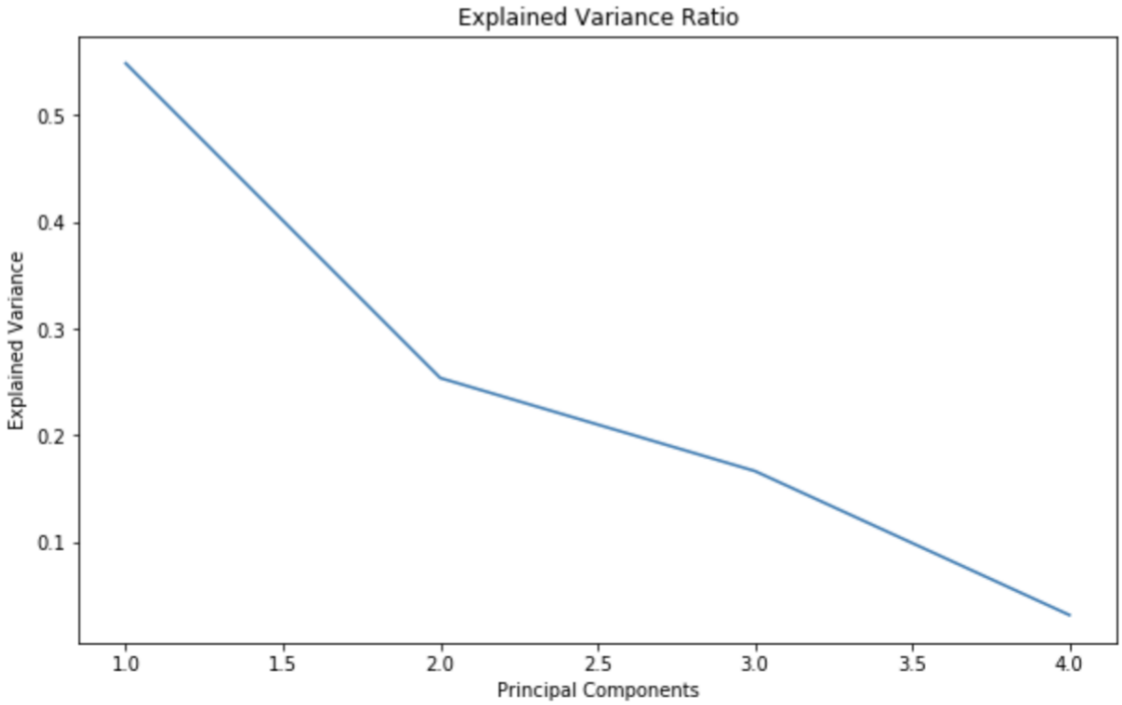
Finally, we have to scale each dataset to prevent biasness as the dataset with larger values will dominate over the others. We will be using the StandardScaler for this.

[4]:
```
ma_windowed = ma.loc[(ma.index > inner_start) & (ma.index <= inner_end)]
vol_windowed = vol.loc[(vol.index > inner_start) & (vol.index <= inner_end)]

filtered_mortgage_rates = mortgage_rates[mortgage_rates['Date'].isin(ma_windowed.index)]
filtered_ma = ma_windowed[ma_windowed.index.isin(mortgage_rates['Date'])]
filtered_vol = vol_windowed[vol_windowed.index.isin(mortgage_rates['Date'])]
filtered_rates = rates[rates['Date'].isin(mortgage_rates['Date'])]

train = np.empty([4, len(filtered_mortgage_rates)])

train[0] = StandardScaler().fit_transform(filtered_ma.values.reshape(-1, 1))[:, 0]
train[1] = StandardScaler().fit_transform(filtered_vol.values.reshape(-1, 1))[:, 0]
train[2] = StandardScaler().fit_transform(filtered_rates['Value'].values.reshape(-1, 1))[:, 0]
train[3] = StandardScaler().fit_transform(filtered_mortgage_rates['Value'].values.reshape(-1, 1))[:, 0]
```

Now, we are ready to fit our 4 datasets to a PCA to analyse the principal components. The 4 components are joined together in a matrix and transposed. The resulting matrix consists of n number of rows with 4 columns. Each row refers to a specific time in the time series, while each of the 4 columns represents the data from each respective dataset of explanatory variables.

After fitting with PCA, the plot of the explained variance ratio of each principal component is shown below. From the graph, we see that the first pricinpal component explains about 55% of the variation, while the second principal component explains about 25%.

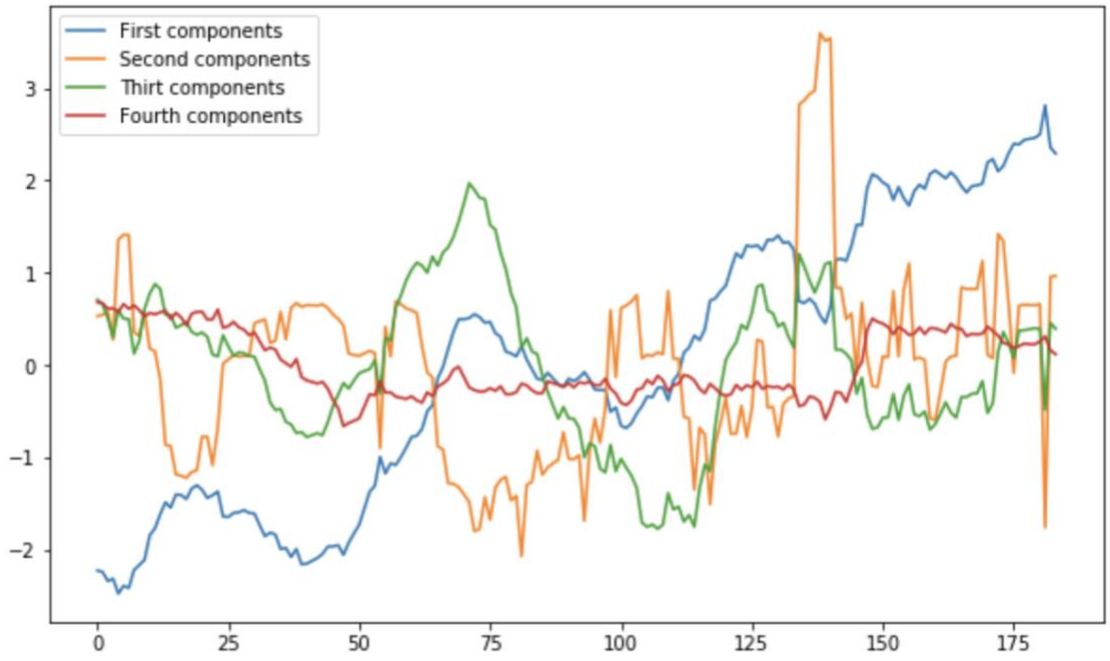
[5]:
```
transposed_train = train.T

# Fit PCA
pca = PCA(n_components=4)
pca.fit(transposed_train)
reduced_data = pca.transform(transposed_train)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 5), pca.explained_variance_ratio_)
plt.ylabel('Explained Variance')
plt.xlabel('Principal Components')
plt.title('Explained Variance Ratio')
plt.show()
```



We can then transform each of the original data by projecting them onto each of the principal components. The plot of the series on each projected component is then shown in the diagram below.

[6]:
```
projected_pc1 = reduced_data[:,0]
projected_pc2 = reduced_data[:,1]
projected_pc3 = reduced_data[:,2]
projected_pc4 = reduced_data[:,3]
```

[7]:
```
plt.figure(figsize=(10, 6))
plt.plot(projected_pc1, label='First components')
plt.plot(projected_pc2, label='Second components')
plt.plot(projected_pc3, label='Thirt components')
plt.plot(projected_pc4, label='Fourth components')
plt.legend(loc='best');
```



The blue line refers to the first principal component while the orange line refers to the second principal component. We can see that the blue line captures the long-term drift, while the orange line, which is mean-reverting, captures the variance.

We are now ready to use the principal components for training a machine learning model for trading. By utilizing the first 2 principal components for training, we can already retain 80% of the information and reduce the number of dimensions by half.

4. Conclusion

- In this report, we introduced Principle Component Analysis which is simplest method in dimensionality reduction techniques. Besides, backtesting methods such as k-fold, purged k-fold cross validation and walk forward analysis are also reviewed.
- In the explosion of the internet, the curse of dimensionality problems are unavoidable, we need preprocessing approaches to handle it, hence, a lot of dimensionality techniques

were born such as SVD, PCA, LDA, Autoencoders, ... therein, PCA is a simple method that based on linear model that produces high performance and efficiency. In our example that applied for nasdag index, a simple two components PCA retains 80% information from 4 features, this clearly shows that large amounts of information appear only in a few important features. With 80% retaining insights, we can boldly model the data with acceptable accuracy and remove overfitting.

- While k-fold cross validation is one of most backtesting methods for optimizing parameters with the purpose reduce overfitting, it reveals many drawbacks when deal with time series data. A biggest disadvantage is breaking serial correlation in time series. Therefore, purged k-fold cross validation was introduced.
- Walk forward analysis is another technique in backtesing family that using for tuning hyper parameters. Whereas each method has its own strengths and weaknesses, we should be careful with overfitting when this method.

References

Tiep, V. H. (2018). Machine Learning co ban.

Robert Pardo (2018).The Evaluation and Optimization of Trading Strategies

Wiley (2018). López de Prado, Marcos, (2018).) Advances in Financial Machine Learning: Chapter 7. Cross-Validation in Finance.

Chad Gray (2018 July 18). Stock Prediction with ML: Walk-forward Modeling

Sanjay.M. (2018, November 13). Why and how to Cross Validate a Model?

Brownlee, J. (2019, August 8). A Gentle Introduction to k-fold Cross-Validation.