CFDS® – Chartered Financial Data Scientist

Introduction to Python

Prof. Dr. Natalie Packham

10 December 2025

Table of Contents

- 6 Statistical Methods in Data Science
  - 6.1 Ridge regression, Lasso and Elastic Net
  - 6.2 Logistic regression
  - 6.3 Principal component analysis

Statistical Methods in Data Science

- In this part, we study a number of statistical methods that have become very popular in Data Science applications:
- Statistical Learning methods are often classified into:
  - Regression versus classification
  - Supervised versus unsupervised learning (versus reinforcement learning)
- The methods studied here cover these different aspects:
  - Ridge regression and Lasso (regression, supervised)
  - Logistic regression (classification, supervised)
  - Principal components analysis (regression, unsupervised)

Ridge regression, Lasso and Elastic Net

- In linear regression, we assume a linear relationship between the *target* Y and the feature vector X:

  $$Y = a + b_1 X_1 + b_2 X_2 + \cdots + b_m X_m + \epsilon,$$

  where $a, b_1, \ldots, b_m$ are constants and ϵ is the error term.

- The ordinary least squares (OLS) estimates of a, b minimise the errors

  $$\sum_{i=1}^{n} \epsilon^2 = \sum_{i=1}^{n} (Y_i - a - b_1 X_{i1} - b_2 X_{i2} - \cdots - b_m X_{im})^2.$$

- In machine learning, especially when the number of features is high and when features are highly correlated, overfitting can occur.
- One way of dealing with this is known as **regularisation**.
- The most popular regularisation methods are:
  - Ridge regression
  - Lasso
  - Elastic net

Ridge regression

- In statistics, **ridge regression** is known as **Tikhonov regularisation** or L_2 **regularisation**.

- Building on OLS, a term is added to the objective function that places a **penalty** on the size of the coefficients $b_1, \ldots, b_m$, by minimising:

  $$\sum_{i=1}^{n} (Y_i - a - b_1 X_{i1} - b_2 X_{i2} - \cdots - b_m X_{im})^2 + \lambda \sum_{j=1}^{m} b_j^2.$$

- The constant λ is called **tuning parameter** or **hyperparameter** and controls the strength of the penalty factor.

- The term $\lambda \sum_{j=1}^{m} b_j^2$ is called the **shrinkage penalty**, as it will shrink the estimates of $b_1, \ldots, b_m$ towards zero.

- Selecting a good value of λ is critial and can be achieved, for example, by **cross-validation**.

Ridge regression

- The OLS estimates do not depend on the magnitude of the independent variables: multiplying X_j by a constant c leads to a scaling of the OLS-

coefficient by $1/c$.

- This is different in ridge regression (and Lasso, see below): the estimated coefficients can change substantially when re-scaling independent variables.

- Therefore, it is custom, to *standardise* the features:

  $$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}},$$

  so that all variables are on the same scale, i.e., they all have a standard deviation of one.

Lasso

- **Lasso (Least absolute shrinkage and selection operator)**, also known as L_1 **regularisation** adds a different penalty:

  $$\sum_{i=1}^{n}(Y_i - a - b_1X_{i1} - b_2X_{i2} - \cdots - b_mX_{im})^2 + \lambda \sum_{j=1}^{m} |b_j|.$$

- This has the interesting effect that the less relevant features are completely eliminated.

- For this reason, Lasso is also often used as a feature selection or variable selection method.

Elastic net regression

- **Elastic net regression** is a mixture of ridge regression and Lasso:

  $$\sum_{i=1}^{n}(Y_i - a - b_1X_{i1} - b_2X_{i2} - \cdots - b_mX_{im})^2 + \lambda_1 \sum_{j=1}^{m} b_j^2 + \lambda_2 \sum_{j=1}^{m}$$
  .

- Combining the effects of ridge regression and Lasso means that simultaneously

  - some coefficients are reduced to zero (Lasso),
  - some coefficients are reduced in size (ridge regression).

Example

- The following application predicts house prices based on different features of the property.

- The data set is from

Hull: Machine Learning in Business. 3rd edition, independently published, 2021.

In [ ]:
```
import pandas as pd # python's data handling package
import numpy as np # python's scientific computing package
import matplotlib.pyplot as plt # python's plotting package
import seaborn as sns
sns.set()

from sklearn.metrics import mean_squared_error as mse
from sklearn.model_selection import train_test_split
# The sklearn library has cross-validation built in!
# https://scikit-learn.org/stable/modules/cross_validation.html
from sklearn.model_selection import cross_val_score
```

In [ ]:
```
# Both features and target have already been scaled: mean = 0; SD =
data = pd.read_csv('data/Houseprice_data_scaled.csv')
# data = pd.read_csv('https://raw.githubusercontent.com/packham/Pyt
```

In [ ]:
```
X = data.drop('Sale Price', axis=1)
y = data['Sale Price']
```

- `sklearn` can split training and testing data randomly.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

Linear Regression

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

In [ ]:
```
lr=LinearRegression()
lr.fit(X_train,y_train)
```

In [ ]:
```
pred = lr.predict(X_test)
mse(y_test, pred)
```

- Observe how the OLS coefficients are all non-zero.
- However, some coefficients are negative where a positive coefficient would be expected (e.g. `FullBath`).
- This is an indication that the model is struggling to fit the large number of features.

In [ ]:
```
# Create dataFrame with corresponding feature and its respective co
coeffs = pd.DataFrame([['intercept'] + list(X_train.columns),[lr.in
coeffs
```

- Indeed, some correlations are high, as the heatmap indicates, which may

cause multicollinearity and ill-fitting.

In [ ]:
```
sns.heatmap(X_train.corr())
```

Ridge regression

In [ ]:
```
# Importing Ridge
from sklearn.linear_model import Ridge
```

- Train on the training data set and use test data set to determine test MSE.

In [ ]:
```
n=int(len(X)*.75) # choose 75% of data for training
# The alpha used by Python's ridge should be the lambda above times
alphas=[0.01*n, 0.02*n, 0.03*n, 0.04*n, 0.05*n, 0.075*n,0.1*n,0.125
mses=[]
for alpha in alphas:
    ridge=Ridge(alpha=alpha)
    ridge.fit(X_train,y_train)
    pred=ridge.predict(X_test)
    mses.append(mse(y_test,pred))
np.transpose([alphas,mses])
```

- This is how to use cross-validation; just specify the number of folds (`cv`) and MSE as the `scoring` function:

In [ ]:
```
# The alpha used by Python's ridge should be the lambda above times
alphas=[0.01*n, 0.02*n, 0.03*n, 0.04*n, 0.05*n, 0.075*n,0.1*n,0.125
mses=[]
for alpha in alphas:
    scores = cross_val_score(Ridge(alpha=alpha), X, y, cv=4, scorin
    mses.append(np.mean(scores))
np.transpose([alphas, mses])
```

- Average test MSE varies with the hyperparameter α.
- The best model choice is at approximately $0.1 \cdot n = 218$.

In [ ]:
```
plt.plot(alphas, mses)
```

Lasso

In [ ]:
```
# Import Lasso
from sklearn.linear_model import Lasso
```

In [ ]:
```
# Here we produce results for alpha=0.05 which corresponds to lambd
lasso = Lasso(alpha=0.05)
lasso.fit(X_train, y_train)
```

- As Lasso acts as a variable selection method we would expect some coefficients to be set to zero:

In [ ]:
```
# DataFrame with corresponding feature and its respective coefficie
coeffs = pd.DataFrame(
    [
        ['intercept'] + list(X_train.columns),
        [lasso.intercept_] + lasso.coef_.tolist()
    ]
).transpose().set_index(0)
coeffs
```

- Now, let's find again the parameter with minimal average test MSE:

In [ ]:
```
# We now consider different lambda values. The alphas are half the
alphas=[0.0025/2, 0.005/2, 0.01/2, 0.015/2, 0.02/2, 0.025/2, 0.03/2
mses=[]
for alpha in alphas:
    scores = cross_val_score(Lasso(alpha=alpha), X, y, cv=4, scorin
    mses.append(np.mean(scores))
np.transpose([alphas,mses])
```

- The optimal parameter is at $\alpha = 0.0075$:

In [ ]:
```
plt.plot(alphas, mses)
```

Logistic regression

- In a regression setting, numerical variables are predicted.
- Another application is classification, which is about predicting the category a new observation belongs to.
- In supervised learning, and with two categories, a variation of regression, called **logistic regression** can be used.
- Given features $X_1, \ldots, X_m$, suppose there are two classes to which observations can belong.
- An example is the prediction of a loan's default risk, given characteristics of the creditor such as age, education, marital status, etc.
- Another example is the classification of e-mails into junk or non-junk e-mails.

Logistic regression

- Logistic regression can be used to calculate the probability of a positive outcome via the **sigmoid function**

$$P(Y=1|X) = \frac{1}{1+\mathrm{e}^{-X}} = \frac{\mathrm{e}^X}{1+\mathrm{e}^X},$$

where e is the Euler constant.

In [ ]:
```
x=np.arange(-7,7,0.25)
plt.plot(x, 1/(1+np.exp(-x)))
```

Logistic regression

- Setting $X = a + b_1X_1 + b_2X_2 + \cdots + b_mX_m$, the probability of a positive outcome is

  $$P(Y=1|X_1,\ldots,X_m) = \frac{1}{1+\exp(-a-\sum_{j=1}^m b_jX_j)}.$$

- The objective is to find the coefficients $a, b_1, \ldots, b_m$ that best classify the given data.
- **Maximum likelihood** is a versatile method for this type of problem, when OLS does not apply.
- Without going into detail, the **log likelihood function** is given as

  $$\ell(a, b_1, \ldots, b_m|x_1, \ldots, x_n) = \sum_{k:y_k=1} \ln(p(x_k)) + \sum_{k:y_k=0} \ln(1-p(x_k))$$

  ,

  and the parameters are chosen that maximise this function.
- (Note: The likelihood function is derived by considering the observations to be independent outcomes of a Bernoulli random variable.)

Example: Credit risk

- The dataset in this example is taken from James et al.: An Introduction to Statistical Learning. Springer, 2013.
- It contains simulated data of defaults on credit card payments, on the basis of credit card balance (amongst other things).
- An excellent tutorial and examples on logistic regression in Python is available here: https://realpython.com/logistic-regression-python/.
- We will use the `sklearn` package below. Logistic regression can also be performed with the `statsmodels.api`, in which case p-values and other statistics are calculated.

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

In [ ]:
```
data = pd.read_csv("./data/Default_JamesEtAl.csv")
# data = pd.read_csv("https://raw.githubusercontent.com/packham/Pyt
```

In [ ]:
```
x=np.array(data["balance"]).reshape(-1,1) # array must be two-dimen
y=np.array([True if x=="Yes" else False for x in data["default"]])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size
```

In [ ]:
```
model = LogisticRegression(solver='liblinear', random_state=0)
model.fit(x_train,y_train)
```

In [ ]:
```
# fitted parameters
a=model.intercept_[0]
b=model.coef_[0,0]
[a,b]
```

- Scatter plot of data and fitted logistic function:

In [ ]:
```
plt.scatter(x,y,c='orange', marker="+")
plt.xlabel('balance')
xrange=range(0,3000,10)
plt.plot(xrange,1/(1+np.exp(-a-b *xrange)))
```

- Predictions:

In [ ]:
```
model.predict_proba(x_train)[:5]
```

In [ ]:
```
model.predict(x_train)[:10]
```

- Mean accuracy of the model:

In [ ]:
```
[model.score(x_train,y_train), model.score(x_test,y_test)]
```

- Confusion matrix:

https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

In [ ]:
```
confusion_matrix(y_train,model.predict(x_train))
```

In [ ]:
```
confusion_matrix(y_test,model.predict(x_test))
```

- See link below for an explanation of the metrics:

https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

In [ ]:
```
print(classification_report(y_train, model.predict(x_train)))
```

In [ ]:
```
print(classification_report(y_test, model.predict(x_test)))
```

Principal component analysis

- **Principal Component Analysis (PCA)** summarises a large set of correlated variables by smaller number of representative variables that explain most of the variability of the original data set.
- It is a standard method for reducing the dimension of high dimensional, highly correlated systems.
- The principal components are common factors that are unobservable (latent) and directly estimated from the data.
- While being explanatory in a statistical sense, the factors do not necessarily have an economic interpretation.

PCA

PCA

- The objective is to take p random variables $X_1, X_2, \ldots, X_p$ and find (linear) combinations of these to produce random variables $Z_1, \ldots, Z_p$, the **principal components**, that are uncorrelated.
- Geometrically, expressing $X_1, \ldots, X_p$ through linear combinations $Z_1, \ldots, Z_p$ can be thought of as shifting and rotating the axes of the coordinate system (see left graph).
- Hence, the transform leaves the data points unchanged, but expresses them using different coordinates.
- The lack of correlation is a useful property because it means that the principal components are measuring different "dimensions" of the data.
- The principal components can be ordered according to their variance, that is, $\mathrm{Var}(Z_1) \geq \mathrm{Var}(Z_2) \geq \cdots \geq \mathrm{Var}(Z_p)$.
- If the variance captured in the higher dimensions is sufficiently small, then

discarding those higher dimensions will retain most of the variability, so only little information is lost.

PCA

- Let (X, Y) be are standard normally distributed random variables with a correlation of 0.7.
- The left graph above shows a scatterplot of a sample of 100 random numbers $(x_1, y_1), \ldots, (x_{100}, y_{100})$.
- By shifting and rotating the axes, new variables Z_1, Z_2, the principal components, with $Z_i = a_i X + b_i Y$ are obtained.
- The data sample expressed in terms of Z_1, Z_2 is uncorrelated.
- Also, the variance of the data contribution from the first principal component Z_1 is much greater than the variance contribution from the second principal component.
- The graph on the right shows the data points when the data are onto the first principal component, discarding the second dimensions.

PCA

- The sample variance of $(x_1, \ldots, x_n)$ is 0.9244 and the sample variance of $(y_1, \ldots, y_n)$ is 0.9226, whereas the sample variance of the data in the first principal component is 1.6014 and of the second principal component is 0.2456.
- In other words, while the original axes each account for roughly 50% of the total variance, the first principal component accounts for 87% of the sample variance.
- Neglecting the second principal component and expressing the data in the first principal component only retains 87% of the variance (see right graph.
- In practice, the number of dimensions will be higher, and one will choose the number of principal components to reflect a certain variance contribution such as 99%.
- If the data are sufficiently correlated, then only few dimensions (principal components) will be required even for high-dimensional data.

PCA example for interest rate term structure

- Interest rates of different maturities are known to exhibit large correlations.
- Interest rate term structures are therefore a good candidate for a representation by a few factors only.

- What do you think are the main ways in which an interest term structure moves over time?
- The data below is taken from

  Hull: Machine Learning in Business. 3rd edition, independently published, 2021.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import linalg
sns.set()
```

In [ ]:
```
data=pd.read_excel("./data/Treasuries_Hull.xlsx", index_col=0, pars
#data=pd.read_excelI"https://raw.githubusercontent.com/packham/Pyth
data.head()
```

- Our interest lies in movements of interest rate term structures, therefore we take first differences of the data.

In [ ]:
```
d=100*data.diff();
d.dropna(inplace=True)
d.head()
```

- Correlations of term structure movements:

In [ ]:
```
sns.heatmap(d.corr())
```

- The principal components are the eigenvectors of the correlation matrix.
- These are also called factor loadings. They express the "weight" of each factor for each maturity.

In [ ]:
```
w, vr=linalg.eig(d.corr()) # eigenvalues, eigenvectors
```

- The first PC captures changes in the level the term structure.
- The second PC captures changes in the slope.
- The third PC can be interpreted as a hump in the term structure.

In [ ]:
```
plt.figure(figsize=(7,5))
plt.xticks(range(len(d.columns)), d.columns, rotation='vertical')
plt.plot(np.real(vr[:,0:3])); # first factor loadings
plt.legend(["PC 1", "PC 2", "PC 3"])
```

- The eigenvalues express the variance captured by each PC.

In [ ]:
```
np.real(100*w/w.sum()) # percentage variances
```

- The principal component scores are the original data expressed in the PC coordinate system, dimension by dimension.
- These are obtained by multiplying each PC vector with the original data (de-meaned).
- Since the principal components are determined from maximising the variance explained by the model, this can be used to interpret the principal components.
- The correlation of each score with the original data allow for an interpretation of what each principal component represents.

In [ ]:
```
pc1 = np.transpose(vr[:,0]*(d-d.mean())).sum()
pc2 = np.transpose(vr[:,1]*(d-d.mean())).sum()
pc3 = np.transpose(vr[:,2]*(d-d.mean())).sum()
pc4 = np.transpose(vr[:,3]*(d-d.mean())).sum()
pc5 = np.transpose(vr[:,4]*(d-d.mean())).sum()
```

- This gives the same result:

In [ ]:
```
pc = np.transpose(np.matmul(np.transpose(vr), np.transpose(d.values
```

In [ ]:
```
for i in range(5):
    d.insert(i, 'pc' + str(i), pc[:,i])
```

- The plot below shows the correlations of each interest rate with the first five PC's.
- Note how the interpretation is similar to the factor loadings above.

In [ ]:
```
plt.figure(figsize=(7,5))
sns.heatmap(np.transpose(d.corr().head(5)).iloc[5:])
```