|  |  |
| --- | --- |
| **Emrah Ozpeynirci** | Amsterdam, The Netherlands  +31(0)6-24 704 287  [emrahozpeynirci@yahoo.com](mailto:emrahozpeynirci@yahoo.com) |
| **credit card**  **default estimation**  **python project** | |
| |  | | --- | | **Abstract** |   This project aims to demonstrate the 7 predictive algorithms over customer default payments in Taiwan and compares the predictive accuracy of probability of default among those methods. From the perspective of risk management, risk prediction is on the upstream for well- developed financial system where the major purpose of risk prediction is to use the financial information to predict business performance or individual customer’s credit risk and to reduc the demage of uncertainity. However, from the perspective of risk control, estimating the probability of default will be more meaningfull than classifying custimers into binary results ie. risks and non-risky. Therefore, the estimated real default probability is an important problem and an interesting challenge.     |  | | --- | | **Introduction** |   In the following context we review 7 predictive algorithms Logistic Regression, K-Nearest Neighbour, Support Vector Machine (SVM), Kernel – SVM, Naïve Bayes, Decision Tree Classification, Random forest Classification under 3 dimensionality reduction techniques PCA (Principal Component Analysis), LDA (Linear Discriminant Analysis) and Kernel PCA.   |  | | --- | | **Data Set + Business Problem Description** |   Default of Credit card Clients Dataset downloaded from UCI Machine Learning Repository Archive.  Source: <https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients>  Characteristics: Multivariate, Classification, 30.000 entries and 24 attributes  **Attribute information:**  Dataset employs a binary variable as default payment (Yes = 1, No = 0), as dependent variable  23 explanatory variables, as independent variables:  Amount of given credit in US Dollar terms, Gender, Education, Maritial Status, Age, History of past payments (payment status from April to September 2005), Measurement of the payment status (pay full, delay 1 month, delay 2 months, delay 3 months, delay 4 months, delay 5 months, delay 6 months, delay 7 months, delay 8 months, delay 9 months and above), Amount of previous payments (Amount paid in September to April) | |
|  | |

|  |
| --- |
| **Importing the dataset and applying PCA (reducing dimensionality)** |

**# Importing the libraries**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

**# Importing the dataset**

dataset = pd.read\_csv('default of credit card clients.csv', header=1)

X = dataset.iloc[:, 1:24].values

y = dataset.iloc[:, 24].values

**# Splitting the dataset into the Training set and Test set**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

From the m independent variables of our dataset PCA extracts p<=m new independent variables that explain the most variance of the dataset, regardless of the dependent variable. The fact that the dependent variable is not considered this feature makes PCA an unsupervised model.

**# Applying PCA**

from sklearn.decomposition import PCA

pca = PCA(n\_components = 2) #number of extracted features we want to get, first run with =None

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

explained\_variance = pca.explained\_variance\_ratio\_

# new extracted independent variables that describe the variance most

array([ 0.28395028, 0.1778588 , 0.0690567 , ..., 0.00181357,

0.00109744, 0.00100168])

|  |
| --- |
| **Fitting Logistic Regression** |

**# Fitting Logistic Regression to the Training set**

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4620, 83],

[1102, 195]], dtype=int64)

4620+195 / 6000 = 80.25% accuracy

|  |
| --- |
| **Fitting K-NN** |

**# Fitting K-NN to the Training set**

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4308, 395],

[ 905, 392]], dtype=int64)

(4308+392)/6000 = 78.33% accuracy

|  |
| --- |
| **Fitting SVM** |

**# Fitting SVM to the Training set**

from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4703, 0],

[1297, 0]], dtype=int64)

4703/6000 = 78.38% accuracy

|  |
| --- |
| **Fitting Kernel - SVM** |

**# Fitting Kernel SVM to the Training set**

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4485, 218],

[ 930, 367]], dtype=int64)

(4485+367)/6000 = 80.86% accuracy

|  |
| --- |
| **Fitting Naïve Bayes** |

**# Fitting Naive Bayes to the Training set**

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4552, 151],

[1041, 256]], dtype=int64)

(4552+256)/6000 = 80.13% accuracy

|  |
| --- |
| **Decision Tree Classification** |

**# Fitting Decision Tree Classification to the Training set**

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[3868, 835],

[ 831, 466]], dtype=int64)

(3868 + 466) / 6000 = 72.23% accuracy

|  |
| --- |
| **Random Forest Classification** |

**# Fitting Random Forest Classification to the Training set**

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators = 100, criterion = 'entropy', random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4319, 384],

[ 901, 396]], dtype=int64)

(4319+396) / 6000 = 78.58% accuracy

Alternative Approach - 1

|  |
| --- |
| **Linear Discriminant Analysis (LDA)** |

*Trying best three classifiers (Logistic Regression, Kernel SVM and Naïve Bayes) with LDA instead of PCA*

LDA is another feature extraction technique. From the n independent variables of our dataset, LDA extracts p<=n new independent variables that saperate the most classes of the dependent variable. Since this technique considers dependent variable, makes LDA a supervised model.

**# Applying LDA**

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n\_components = 2)

X\_train = lda.fit\_transform(X\_train, y\_train)

X\_test = lda.transform(X\_test)

**# Fitting Logistic Regression to the Training set**

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4600, 103],

[ 994, 303]], dtype=int64)

(4600+303)/6000 = 81.71% accuracy

**# Fitting Kernel SVM to the Training set**

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4501, 202],

[ 943, 354]], dtype=int64)

(4501 + 354) / 6000 = 80.91% accuracy

**# Fitting Naive Bayes to the Training set**

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4574, 129],

[1033, 264]], dtype=int64)

(4574 + 264)/6000 = 80.63% accuracy

Alternative Approach - 2

|  |
| --- |
| **Kernel PCA** |

Kernel-PCA is another feature extraction technique used when the data is not linearly separable. It is a kernelized version of PCA where we map the data to higher dimension using the kernel trick then we extract new principle components.

**# Applying Kernel PCA**

from sklearn.decomposition import KernelPCA

kpca = KernelPCA(n\_components = 2, kernel = 'rbf') #using the gaussian kernel

X\_train = kpca.fit\_transform(X\_train)

X\_test = kpca.transform(X\_test)

**# Fitting Kernel SVM to the Training set**

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random\_state = 0)

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4481, 222],

[ 975, 322]], dtype=int64)

(4481+322)/6000 = 80.05% accuracy

**# Fitting Naive Bayes to the Training set**

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

**# Predicting the Test set results**

y\_pred = classifier.predict(X\_test)

**# Making the Confusion Matrix**

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

array([[4449, 254],

[ 953, 344]], dtype=int64)

(4449+344)/6000 = 79.88% accuracy

The best prediction is achieved by applying LDA analysis to reduce dimensionality and using Logistic Regression. This regression type also the second best performer under PCA but running under Kernel PCA gave memory error. I we use this model we are able to model and predict little more than 80% of our test set.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Accuracy Matrix | | | | | | | | |
|  | Logistic Regression | K-NN | SVM | Kernel-SVM | Naïve Bayes | Decision Tree | Random Forest |
| PCA | 80.25% | 78.33% | 78.38% | 80.86% | 80.13% | 72.23% | 78.58% |
| LDA | 81.71% |  |  | 80.91% | 80.63% |  |  |
| Kernel PCA | Memory error |  |  | 80.05% | 79.88% |  |  |