//

Machine Learning IV

maXbox Starter 66 - Data Science with Max

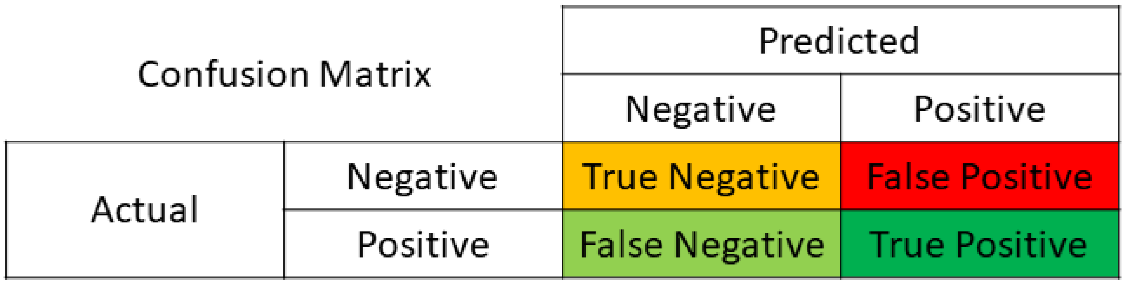
There are two kinds of data scientists:
 1) Those who can extrapolate from incomplete data.

This tutor makes a comparison of a several classifiers in scikit-learn on synthetic datasets. The dataset is very simple as a reference of understanding. We do have 6 samples of 4 features [A,B,C,D] with 2 classes [0,1]:

```
list2 = [[1,2,3,4,0],[3,4,5,6,0], [5,6,7,8,1],
        [7,8,9,10,1],[10,8,6,4,0],[9,7,5,3,1]]

arr2 = np.array(list2, dtype='float')
print(arr2,'\n')
```

```
   A   B   C   D
[[ 1.  2.  3.  4.  0.]
 [ 3.  4.  5.  6.  0.]
 [ 5.  6.  7.  8.  1.]
 [ 7.  8.  9. 10.  1.]
 [10.  8.  6.  4.  0.]
 [ 9.  7.  5.  3.  1.]]
```

We simply score a classifier with a confusion matrix. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.



Confusion Matrix		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

There are two possible predicted classes: 1 as "yes" and 0 as "no". If we were predicting for example the presence of a disease, "yes" would mean they have the disease, and "no" would mean they don't have the disease.
If you want to learn to carry out tasks **and** concepts by themselves, so here **is** an overview of the confusion matrix and a general overview:

http://www.softwareschule.ch/decision.jpg
http://www.softwareschule.ch/examples/machinelearning.jpg

OK., lets start with a first classifier, we split the dataset in **y** (target or label) and **X** (predictors) for the 4 features[1]:

```
y = arr2[0:,4]
X = arr2[0:,0:4]
features = ['A','B','C','D']
```

1 For sake of simplicity we don't split data in a train and test set

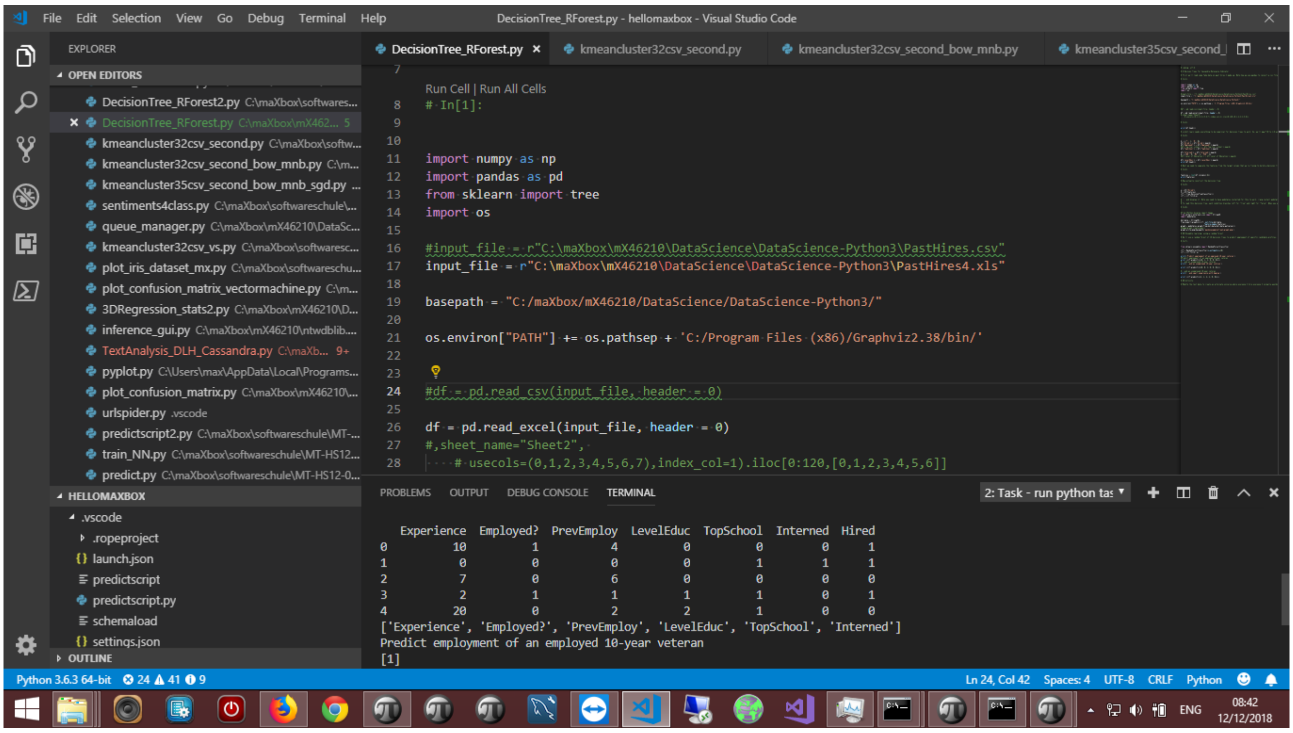
```
print(y,'\n',X,'\n')

[0. 0. 1. 1. 0. 1.]

[[ 1.  2.  3.  4.]
 [ 3.  4.  5.  6.]
 [ 5.  6.  7.  8.]
 [ 7.  8.  9. 10.]
 [10.  8.  6.  4.]
 [ 9.  7.  5.  3.]]
```

Our first classifier is a linear support vector machine.

```
from sklearn.svm import LinearSVC

svm = LinearSVC(random_state=100)
y_pred = svm.fit(X,y).predict(X)    # fit and predict in one line

print('linear svm score: ',accuracy_score(y, y_pred))
linear svm score:  0.8333333333333334
print(confusion_matrix(y, y_pred))

[[2 1]
 [0 3]]

print("Numbs of mislabeled points out of total %d points : %d"
                    % (X.shape[0],(y != y_pred).sum()))

Numbs of mislabeled points out of total 6 points : 1
```

So now we interpret the missing one on the next page!

Note: Linear SVC is similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples, not only 6 in our case.


DecisionTree_RForest.py - hellomaxbox - Visual Studio Code
DecisionTree_RForest.py
kmeancluster32csv_second.py
kmeancluster32csv_second_bow_mnb.py
Run Cell | Run All Cells
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
2: Task - run python tas
Ln 24, Col 42 Spaces: 4 UTF-8 CRLF Python
08:42 12/12/2018

```
# In[1]:


import numpy as np
import pandas as pd
from sklearn import tree
import os

#input_file = r"C:\maXbox\mX46210\DataScience\DataScience-Python3\PastHires.csv"
input_file = r"C:\maXbox\mX46210\DataScience\DataScience-Python3\PastHires4.xls"

basepath = "C:/maXbox/mX46210/DataScience/DataScience-Python3/"

os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'


#df = pd.read_csv(input_file, header = 0)

df = pd.read_excel(input_file, header = 0)
#,sheet_name="Sheet2",
    # usecols=(0,1,2,3,4,5,6,7),index_col=1).iloc[0:120,[0,1,2,3,4,5,6]]
```

```
   Experience  Employed?  PrevEmploy  LevelEduc  TopSchool  Interned  Hired
0          10          1           4          0          0         0      1
1           0          0           0          0          1         1      1
2           7          0           6          0          0         0      0
3           2          1           1          1          1         0      1
4          20          0           2          2          1         0      0
['Experience', 'Employed?', 'PrevEmploy', 'LevelEduc', 'TopSchool', 'Interned']
Predict employment of an employed 10-year veteran
[1]
```

The confusion matrix has the form:

```
print(confusion_matrix(y, y_pred))
[[2 1]
 [0 3]]
```

The first row belongs to the 0 class and the second the 1 class:

```
    0 1 predict
0 [[2 1]
1  [0 3]]
```

As we can see there's one false positive predicted!

Confusion Matrix		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

As we compare y != y_pred:

```
print((y, y_pred))

[0. 0. 1. 1. 0. 1.] = y      (actual)
[0. 0. 1. 1. 1. 1.] = y_pred (predicted)
```

That means we predicted yes [1], but they don't actually have the disease; we can also say the false positive is like a false alarm.
What can we learn from this matrix?

- There are two possible predicted classes: "yes" and "no". If we were predicting the presence of a disease, for example, "yes" would mean they have the disease, and "no" would mean they don't have the disease after a diagnosis.
- The classifier made a total of 6 predictions (e.g., 6 patients were being tested for the presence of that disease).
- Out of those 6 cases, our classifier predicted "yes" 4 times, and "no" 2 times (no=0, yes=1).
- In reality and best case, 3 patients in the sample have the disease, and 3 patients do not (only true negative & true positive):
- [[3 0]
- [0 3]]

We can conclude the result with a report:

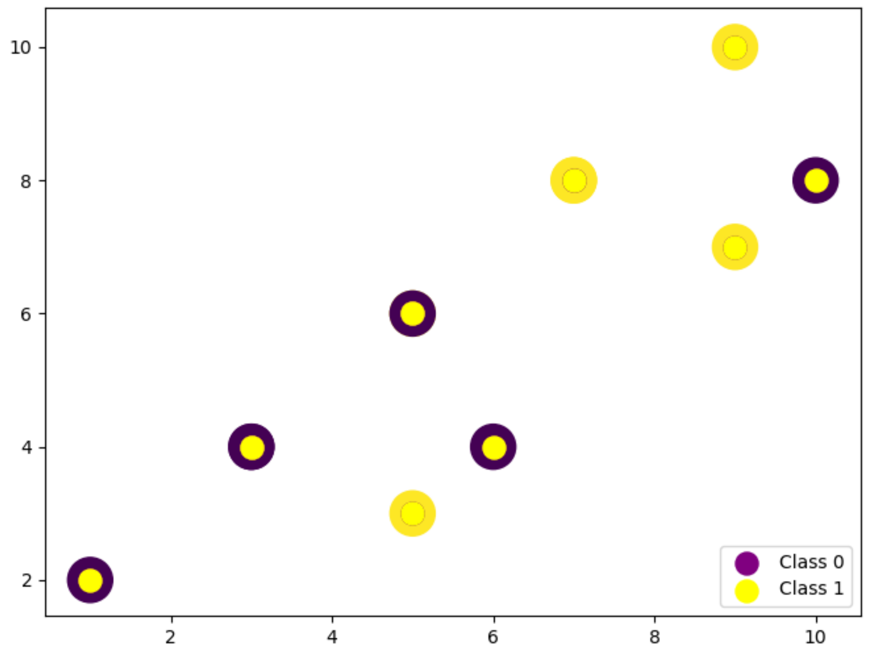
```
from sklearn.metrics import classification_report
classification report:
              precision    recall  f1-score   support

    class 0       1.00      0.67      0.80         3
    class 1       0.75      1.00      0.86         3

avg / total       0.88      0.83      0.83         6
```

The precision mean when it predicts (yes or no), how often is it correct?
TP/predicted yes = 3/4 = 0.75
The recall mean when it's actually yes, how often does it predict yes?

Note that in binary classification, recall of the positive class is also known as “sensitivity”; recall of the negative class is “specificity”.
More details of that topic at:

https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/
or EKON 22 - November 2018 at Düsseldorf Session

So our data has 4 features & 3 duplicates, easy to find with a classification:



Our next classification test is another support vector machine of supervised learning, but with a non-linear kernel:

```
clf = SVC(random_state=100)
y_pred = clf.fit(X,y).predict(X)
print('supportvectormachine score1: ',clf.score(X,y))
print('score2: ',accuracy_score(y, y_pred))
print(confusion_matrix(y, y_pred))
#plotPredictions(clf)
```

I initialize the constructor with a defined random state, so our tests will have always the same result to reproduce. The seed of the pseudo random number generator used when shuffling the data for probability estimates.
This classification has no mislabeled data, the score is 1:

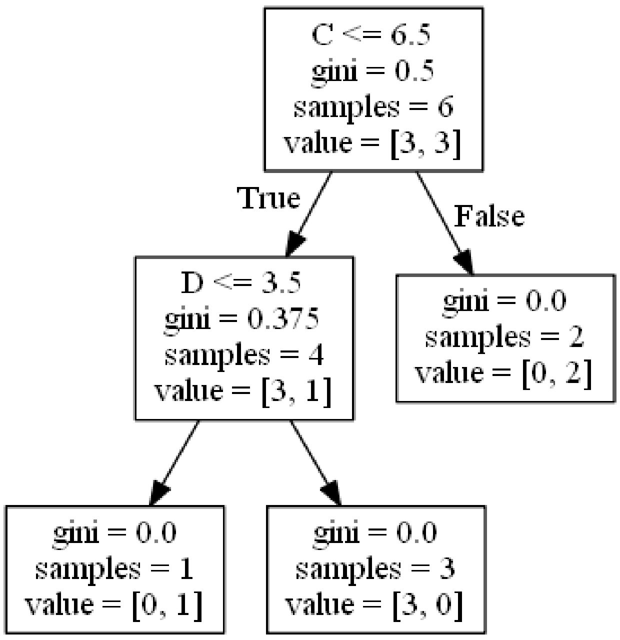
```
>>> supportvectormachine score1:  1.0
score2:  1.0
[[3 0]
 [0 3]]
classification report:
              precision    recall  f1-score   support

    class 0       1.00      1.00      1.00         3
    class 1       1.00      1.00      1.00         3

avg / total       1.00      1.00      1.00         6
```

Another 4 classifier with scores on the same script[2]:

```
clf = GaussianNB()
y_pred = clf.fit(X,y).predict(X)
print('gaussian nb score2: ',accuracy_score(y, y_pred))
print(confusion_matrix(y, y_pred))
>>>gaussian nb score2:  0.8333333333333334
[[2 1]
 [0 3]]

clf = MLPClassifier(alpha=1, random_state=100)
y_pred = clf.fit(X,y).predict(X)
print('mlperceptron score2: ',accuracy_score(y, y_pred))
print(confusion_matrix(y, y_pred))
>>> mlperceptron score2:  0.8333333333333334
[[2 1]
 [0 3]]

clf = KNeighborsClassifier(n_neighbors=3)
y_pred = clf.fit(X,y).predict(X)
print('kneighbors score2: ',accuracy_score(y, y_pred))
print(confusion_matrix(y, y_pred))
#plotPredictions(clf)
[[2 1]
 [0 3]]

clf = DecisionTreeClassifier(random_state=100,max_depth=5)
y_pred = clf.fit(X,y).predict(X)
print('decision tree score2: ',accuracy_score(y, y_pred))
print(confusion_matrix(y, y_pred))
>>> decision tree score2:  1.0
[[3 0]
 [0 3]]
```

Decision Trees and Random Forest are very interesting, cause you can visualize the implicit knowledge to a explicit decision map with the help of pydotplus:

```
import pydotplus;
```



2 http://www.softwareschule.ch/examples/classifier_compare2confusion.py.txt

```
from sklearn.externals.six import StringIO
import pydotplus
from sklearn import tree

dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
                         feature_names=features)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
print(graph,dot_data, basePath)
#Image(graph.create_png())

graph.write_png(basePath+'\maxboxdecisiontree_graph2.png')

os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
```

The features of a decision tree are always randomly permuted at each split. Therefore, the best found split may vary, even with the same training data and max_features=n_features, if the improvement of the criterion is identical for several splits enumerated during the search of the best split.
To obtain a deterministic behavior during fitting, random_state has to be fixed.

```
clf = DecisionTreeClassifier(random_state=100,max_depth=5)
```

https://en.wikipedia.org/wiki/Decision_tree_learning

Of course, machine learning (often also referred to as Artificial Intelligence, Artificial Neural Network, Big Data, Data Mining or Predictive Analysis) is not that new field in itself as they want to believe us. For most of the cases you do experience 5 steps in different loops in a artificial neural network:

• Data Shape	(Write one or more dataset importing functions)
• Modelclass	(Pre-made Estimators encode best practices,)
• Cost Function	(cost or loss, and then try to minimize error)
• Optimizer	(optimization to constantly modify vars to reduce costs.)
• Classify	(Choosing a model and classify algorithm - supervised)
• Accuracy Score	(Predict or report precision and drive data to decision)

At its core, most algorithms should have a proof of classification and this is nothing more than keeping track of which feature gives evidence to which class. The way the features are designed determines the model that is used to learn. This can be a confusion matrix, a certain confidence interval, a T-Test statistic, p-value or something else used in hypothesis[3] testing.

http://www.softwareschule.ch/examples/decision.jpg

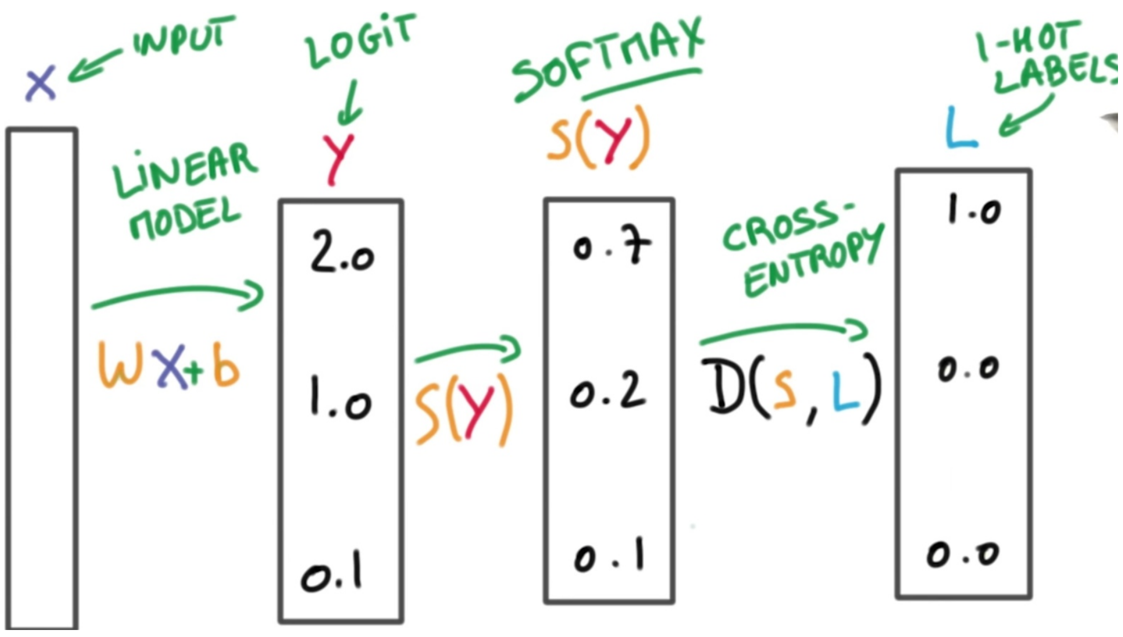
Note about the Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent. MLPClassifier trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters.

It can also have a regularization term added to the loss function (e.g. cross entropy) that shrinks model parameters to prevent over-fitting (learn by heart).

This implementation works with data represented as dense numpy arrays or sparse scipy arrays of floating point values.

3 A thesis with evidence

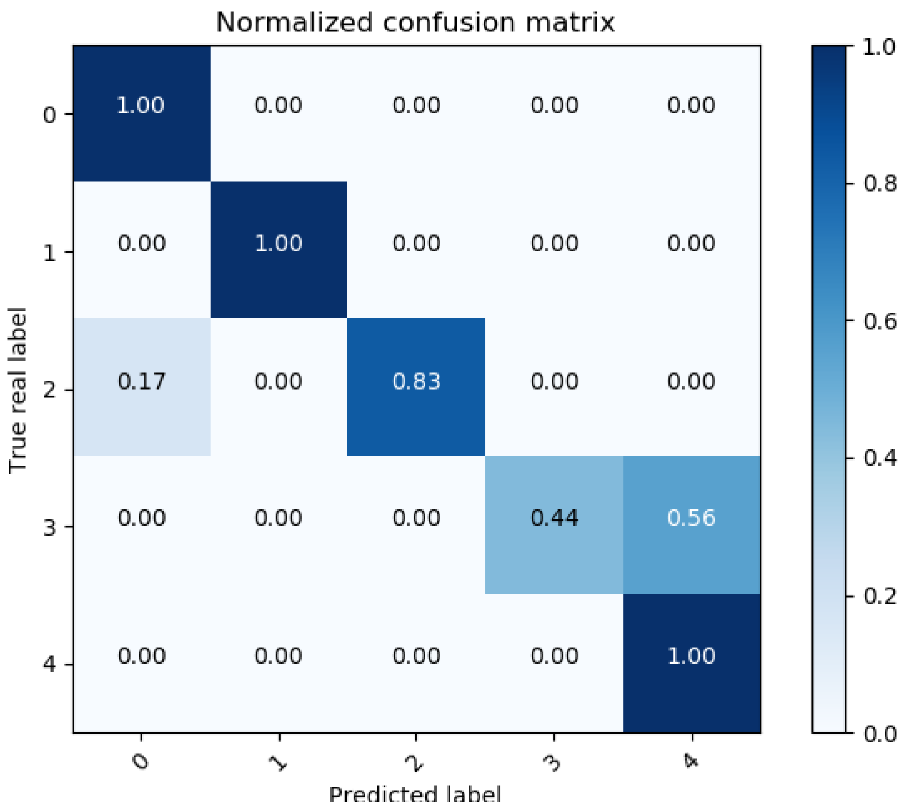


"Classification with Cluster"

```
Now we will use linear SVC to partition our graph into clusters and split the
data into a training set and a test set for further predictions.

 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Run classifier, using a model that is too regularized (C too low) to see
# the impact on the results

 classifier = svm.SVC(kernel='linear', C=0.01)
 y_pred = classifier.fit(X_train, y_train).predict(X_test)

By setting up a dense mesh of points in the grid and classifying all of them, we
can render the regions of each cluster as distinct colors:

def plotPredictions(clf):
        xx, yy = np.meshgrid(np.arange(0, 250000, 10),
                        np.arange(10, 70, 0.5))
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        plt.figure(figsize=(8, 6))
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
        plt.scatter(X[:,0], X[:,1], c=y.astype(np.float))
        plt.show()
```

A simple CNN architecture was trained on MNIST dataset using TensorFlow with 1e-3 learning rate and cross-entropy loss using four different optimizers: SGD, Nesterov Momentum, RMSProp and Adam.
We compared different optimizers used in training neural networks and gained intuition for how they work. We found that SGD with Nesterov Momentum and Adam produce the best results when training a simple CNN on MNIST data in TensorFlow.

https://sourceforge.net/projects/maxbox/files/Docu/EKON_22_machinelearning_slides_scripts.zip/download



Last note concerning PCA and Data Reduction or Factor Analysis:

```
As PCA simply transforms the input data, it can be applied both to
classification and regression problems. In this section, we will use a
classification task to discuss the method.
The script can be found at:
  http://www.softwareschule.ch/examples/811_mXpcatest_dmath_datascience.pas
  ..\examples\811_mXpcatest_dmath_datascience.pas

from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QA

clf = QA()
y_pred = clf.fit(X,y).predict(X)
print('\n QuadDiscriminantAnalysis score2: ',accuracy_score(y, y_pred))
print(confusion_matrix(y, y_pred))

  It may be seen that:
• High correlations exist between the original variables, which are
  therefore not independent
• According to the eigenvalues, the last two principal factors may be
  neglected since they represent less than 11 % of the total variance. So,
  the original variables depend mainly on the first two factors
• The first principal factor is negatively correlated with the second and
  fourth variables, and positively correlated with the third variable
• The second principal factor is positively correlated with the first variable
• The table of principal factors show that the highest scores are usually
  associated with the first two principal factors, in agreement with the
  previous results
Const
  N    = 6;  { Number of observations }
  Nvar = 4;   { Number of variables }
```

Of course, its not always this and that simple. Often, we don't know what number of dimensions is advisable in upfront. In such a case, we leave n_components or Nvar parameter unspecified when initializing PCA to let it calculate the full transformation. After fitting the data, explained_variance_ratio_ contains an array of ratios in decreasing order: The first value is the ratio of the basis vector describing the direction of the highest variance, the second value is the ratio of the direction of the second highest variance, and so on.

```
print('pearson correlation, coeff:, p-value:')
for i in range(3):
  print (pearsonr(X[:,i],X[:,i+1]))

corr = np.corrcoef(X, rowvar=0)  # correlation matrix
w, v = np.linalg.eig(corr)        # eigen values & eigen vectors
print('eigenvalues & eigenvector:')
print(w)
print(v)
```

```
>>> eigenvalues & eigenvector:

[ 2.66227922e+00  1.33772078e+00 -4.33219131e-18  6.51846049e-17]

[[-0.46348627  0.56569851 -0.11586592  0.19557765]
 [-0.5799298   0.27966438  0.53986421  0.24189689]
 [-0.5724941  -0.30865195 -0.71438049 -0.75459654]
 [-0.34801208 -0.71169306  0.42986304  0.57777101]]
C:\maXbox\mX46210\DataScience\confusionlist
```

You can detect high-multi-collinearity by inspecting the *eigen values* of *correlation matrix*. A very low eigen value shows that the data are collinear, and the corresponding *eigen vector* shows which variables are collinear. If there is no collinearity in the data, you would expect that none of the eigen values are close to zero. Multicollinearity does not reduce the predictive power or reliability of the model as a whole, at least within the sample data set; it only affects calculations regarding individual predictors.

Being a linear method, PCA has, of course, its limitations when we are faced with strange data that has non-linear relationships. We wont go into much more details here, but its sufficient to say that there are extensions of PCA.

All slides and scripts of the above article:

https://sourceforge.net/projects/maxbox/files/Docu/EKON_22_machinelearning_slides_scripts.zip/download

The script with 7 classifiers can be found:

http://www.softwareschule.ch/examples/classifier_compare2confusion.py.txt

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QA
```

    http://www.softwareschule.ch/box.htm

    https://scikit-learn.org/stable/modules/

    https://packaging.python.org/tutorials/managing-dependencies/

https://towardsdatascience.com/understanding-data-science-classification-metrics-in-scikit-learn-in-python-3bc336865019

Doc:
    http://fann.sourceforge.net/fann_en.pdf
    http://www.softwareschule.ch/examples/datascience.txt
    https://maxbox4.wordpress.com

Last Note:

Pipenv is a dependency manager for Python projects. If you're familiar with Node.js' npm , Composer of PHP, or Ruby's bundler, it is similar in spirit to those tools. While pip alone is often sufficient for personal use, Pipenv is recommended for collaborative projects as it's a higher-level tool that simplifies dependency management for common use cases.

Use pip to install Pipenv:

```
pip install --user pipenv
```

Keep in mind that Python is used for a great many different purposes, and precisely how you want to manage your dependencies may change based on how you decide to publish your software.