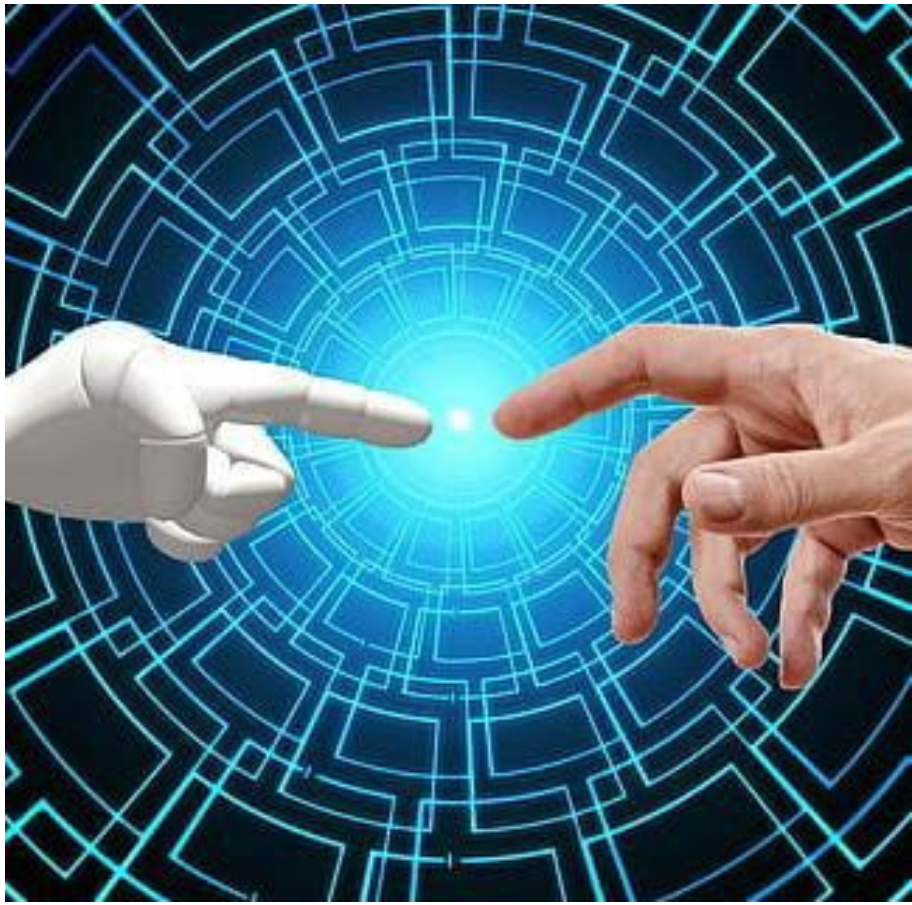
Artificial Intelligence (AI) for Engineering

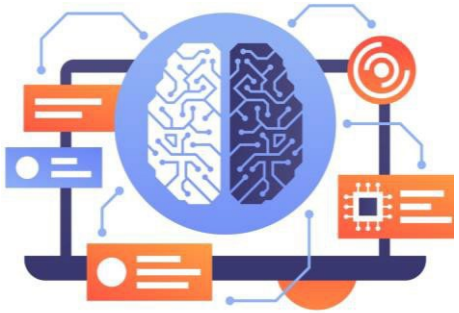


COS40007

Dr. Afzal Azeem Chowdhary

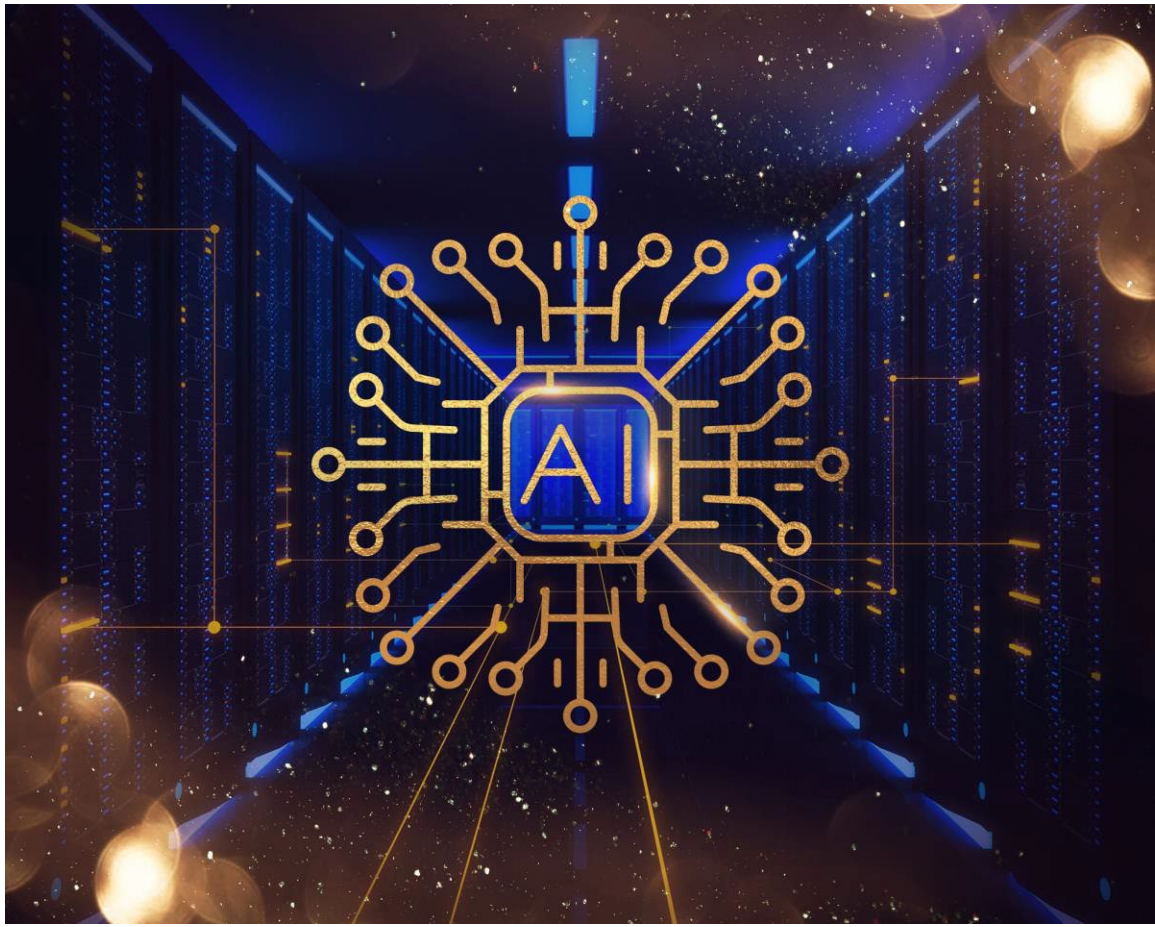
Lecturer, SoCET, Swinburne University of Technology

Seminar 4: 25th March 2025



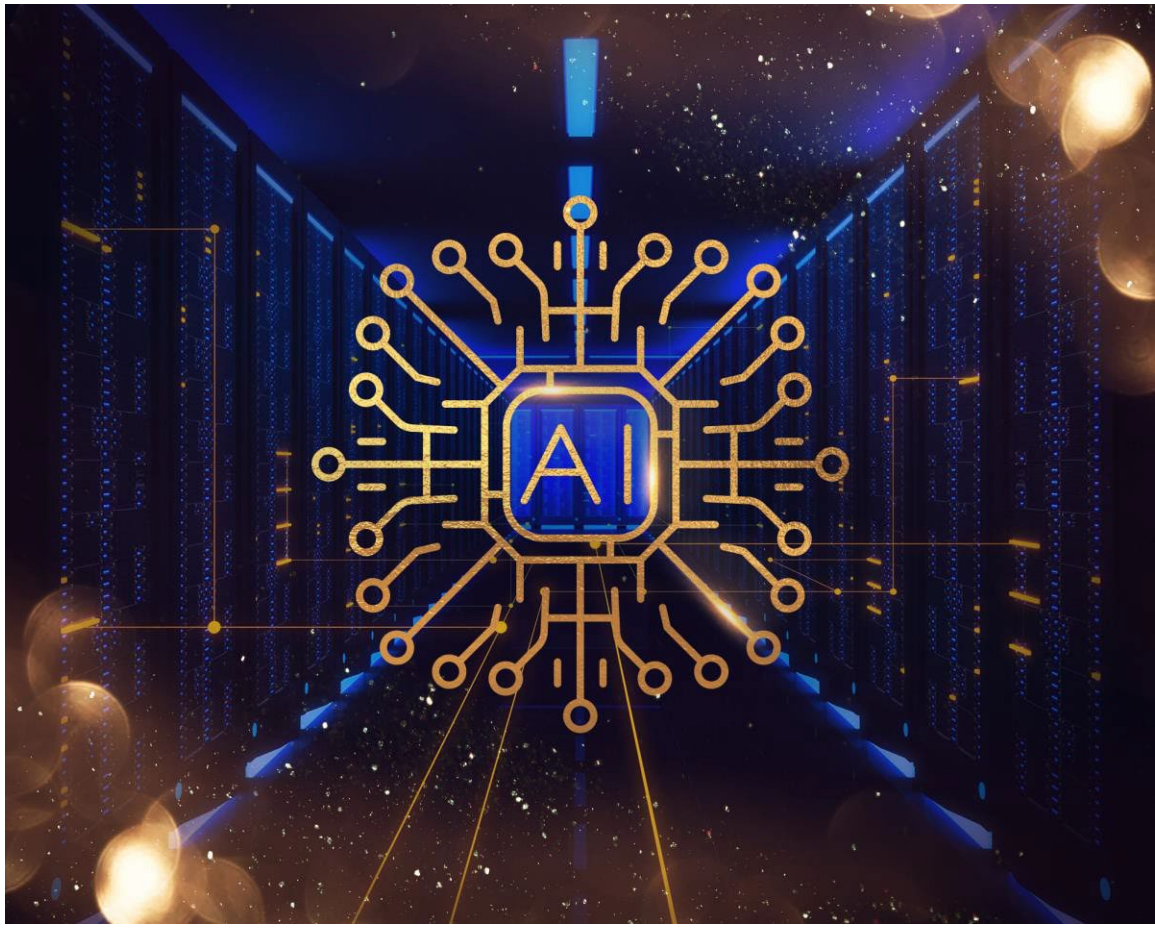
Overview

- Scikit-learn
- Examples of scikit-learn for Machine Learning
- Basics of Clustering



Required Reading

- Chapter 1-6 of “Applied Machine Learning and AI for Engineers”
- Chapter 2-6 of “Machine Learning with Pytorch and Scikit-Learn”



At the end of this you should be able to

- Understand what functions are available in Scikit-learn
- Understand how to perform training and validation in Scikit-learn
- Understand how to do evaluation using scikit-learn
- Understand how to do clustering using scikit-learn

Python

Simple Python Programs

```
str = "1 4 3  5 7"

a = str.split();

l = len(a)
for i in range(0, l, 1):
      if(int(a[i]) > 3):
            print(a[i])
print('\n')
```

```
l = []
l2 = [1, 10]
l.append(l2)
l2 = [10, 100]
l.append(l2)
l2 = [20, 200]
l.append(l2)

rows = 3;
cols = 2;

for i in range(0, rows, 1):
      for j in range(0, cols, 1):
            print(l[i][j])
            print(" ")
      print('\n')
```

```
f = open("inputdata")

data = []
i=0;
l = f.readline()
while(l != ''):
      a = l.split()
      l2 = []
      for j in range(0, len(a), 1):
            l2.append(a[j])
      data.append(l2)
      l = f.readline()

rows = len(data)
cols = len(data[0])

print("row=", rows, "cols=", cols)

for i in range(0, rows, 1):
      print(data[i])
```

scikit-learn: Machine Learning with Python

- Simple and efficient tools for machine learning and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Supervised learning, identifying to which category an input data point or object belongs.

Applications: Activity Recognition, Image Recognition, Fake New Detection, Cancer-cell Detection.

Algorithms: SVM, *k*-Nearest Neighbors *(k*NN*),* Random Forests, Decision Trees, Naïve Bayes, Logistic Regresssion

Regression

Supervised learning, predicting a continuous-valued attribute associated with an object based on independent features. The ultimate goal of the regression algorithm is to plot a best-fit line or a curve between the data.

Applications: Drug Response, Stock Prices, Forecasting Sales.

Algorithms: Support Vector Regression (SVR) extends the principles of Support Vector Machines (SVM), Linear Regresssion , Ridge Regression, Lasso Regression.

Clustering

Unsupervised learning, automatic grouping of similar objects into sets and reveals hidden structures.

Applications: Customer segmentation, Grouping experiment outcomes, Anomaly Detection.

Algorithms: Centroid-based Clustering (*k*-Means), Spectral Clustering, Density based Scan (DBScan), Hierarchical Based

Dimensionality Reduction

Reducing the number of random independent variables to consider so that the low-dimensional representation retains some meaningful properties of the original data.

Applications: Visualization, Increased efficiency

Algorithms: PCA, Linear discriminant analysis (LDA), T-distributed stochastic neighbor embedding (t-SNE), feature selection, non-negative matrix factorization

Model selection

Comparing, validating and choosing the best parameters and the most appropriate model for a given issue.

Goal: Improved accuracy via Parameter tuning.

Modules: Grid search, Cross validation, Randomised Search, Train-Test Split, Performance metrics, Model Comparison.

Preprocessing & Dataset

Dataset Loading and Dataset Transformation is the first and most important step.
We already have gone in detail in Seminar 2 but briefly written below again: Feature extraction, standardization and normalization.

Goal: Transforming input data such as text for use with machine learning algorithms.

Modules: Preprocessing, Feature Extraction.

Most importantly

- Ease of use
- Good comprehensive documentation
- Good Community Support

These points contribute to its popularity and widespread adoption of Good Documentation.

Scikit-Learn API

- Object-oriented interface centered around the concept of an Estimator:
- “An estimator is any object that learns from data; it maybe
  - a classification,
  - regression or
  - clustering algorithm or
  - a transformer that extracts/filters useful features from raw data.”

Scikit-learn

Scikit learn models follow a simple, shared pattern

1. Import the model that you need to use
2. Build the model, setting its hyperparameters
3. Train model parameters on your data: Using the fit method
4. Use the model to make predictions
    - Using the predict/transform methods
    - Sometimes fit and predict/transform are implemented within the same class method

Scikit-learn

- **fit()**: learn model parameters from input data. E.g. train a classifier
- **predict()**: apply model parameters to make predictions on data. e.g. predict class labels
- **fit_predict()**: fit model and make predictions to data e.g. apply clustering to data
- **fit_transform()**: fit model and transform data transform data E.g. apply PCA to transform data

Estimator API

```
class Estimator(object):
def fit(self, X, y=None):
"""Fits estimator to data. """ #
set state of ``self``
return self
def predict(self, X):
"""Predict response of ``X``. """
# compute predictions ``pred``
return pre
```

Estimators

- `fit(X,y)` sets the state of the estimator.
- `X` is usually a 2D numpy array of `shape (num_samples, num_features)`.
- `y` is a 1D array with `shape (n_samples,)`
- `predict(X)` returns the class or value
- `predict_proba()` returns a 2D array of `shape (n_samples, n_classes)`

```
from sklearn import svm
estimator = svm.SVC(gamma=0.001)
estimator.fit(X, y)
estimator.predict(x)
```

Transformer

```
class Transformer(Estimator):
def transform(self, X):
"""Transforms the input data. """ # transform ``X``
to ``X_prime``
      return X_prime

from sklearn import preprocessing

Xt = preprocessing.normalize(X) # Normalizer
Xt = preprocessing.scale(X) # StandardScaler

imputer =Imputer(missing_values='Nan’,strategy='mean') Xt =
imputer.fit_transform(X)
```

Classification

```
from sklearn import metrics
from sklearn import cross_validation as cv

splits = cv.train_test_split(X, y, test_size=0.2)
X_train, X_test, y_train, y_test = splits

model = ClassifierEstimator()
model.fit(X_train, y_train) expected =
y_test
predicted = model.predict(X_test)

print metrics.classification_report(expected, predicted) print
metrics.confusion_matrix(expected, predicted) print
metrics.f1_score(expected, predicted)
```

MSE and Coefficient of determination

```
from sklearn import metrics
from sklearn import cross_validation as cv

splits = cv.train_test_split(X, y, test_size=0.2)
X_train, X_test, y_train, y_test = splits

model = RegressionEstimator() model.fit(X_train,
y_train)

expected = y_test
predicted = model.predict(y_test)
print metrics.mean_squared_error(expected, predicted)
print metrics.r2_score(expected, predicted)
```

Decision Tree and Random Forest

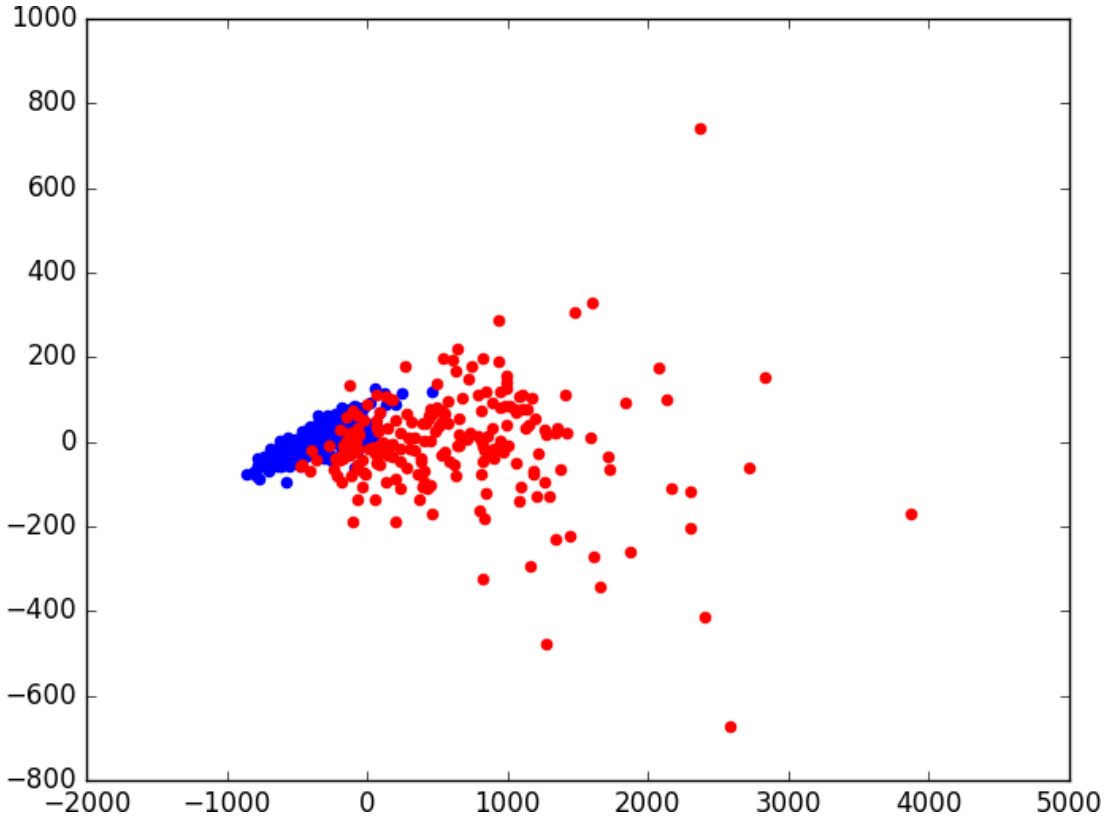
```
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
import numpy as np

f = open("bc.train.0")
data = np.loadtxt(f)
train = data[:,1:]
trainlabels = data[:,0]

f = open("bc.test.0")
data = np.loadtxt(f)
test = data[:,1:]
testlabels = data[:,0]

clf = tree.DecisionTreeClassifier()
clf.fit(train,trainlabels)
prediction = clf.predict(test)

rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(train,trainlabels)
prediction2 = rfc.predict(test)

err = 0
err2 = 0
for i in range(0, len(prediction), 1):
    if(prediction[i] != testlabels[i]):
        err += 1
    if(prediction2[i] != testlabels[i]):
        err2 += 1
err = err/len(testlabels)
err2 = err2/len(testlabels)
print(err,err2)
```

Dimensionality reduction and visualization with PCA

```
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA

import numpy as np

f = open("bc")
data = np.loadtxt(f)

X = data[:,1:]
Y = data[:,0]

pca = PCA(n_components=2)
pca.fit(X)
newdata = pca.transform(X)

y1 = []
y2 = []
for i in range(0, len(newdata), 1):
        if(Y[i] == 1):
                y1.append(newdata[i][0])
                y2.append(newdata[i][1])

plt.scatter(y1, y2, color='blue')

y1 = []
y2 = []
for i in range(0, len(newdata), 1):
        if(Y[i] == -1):
                y1.append(newdata[i][0])
                y2.append(newdata[i][1])

plt.scatter(y1, y2, color='red')

plt.show()
```

Dimensionality reduction and visualization with PCA

PCA plot of breast cancer data (output of program in previous slide)



k-means clustering

```
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import numpy as np

f = open("bc")
data = np.loadtxt(f)

X = data[:,1:]
Y = data[:,0]

clustering = KMeans(n_clusters=2,init='random').fit(X)

err = 0
for i in range(0, len(X), 1):
        if(clustering.labels_[i] != Y[i]):
                err += 1
err /= len(X)
print(err)
```

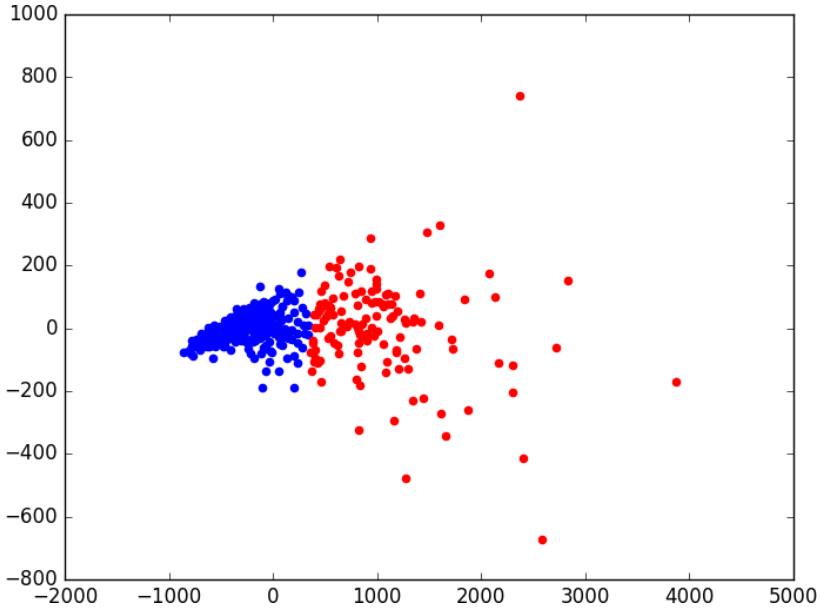
k-means PCA plot in scikit-learn

```
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
import numpy as np
from sklearn.decomposition import PCA

f = open("bc")
data = np.loadtxt(f)
X = data[:,1:]
Y = data[:,0]

pca = PCA(n_components=2)
pca.fit(X)
newdata = pca.transform(X)
clustering = KMeans(n_clusters=2,init='random').fit(X)

x = []
y = []
for i in range(0, len(newdata), 1):
        if(clustering.labels_[i] == 0):
                x.append(newdata[i][0])
                y.append(newdata[i][1])

plt.scatter(x, y, color='blue')

x = []
y = []
for i in range(0, len(newdata), 1):
        if(clustering.labels_[i] == 1):
                x.append(newdata[i][0])
                y.append(newdata[i][1])

plt.scatter(x, y, color='red')
plt.show()
```

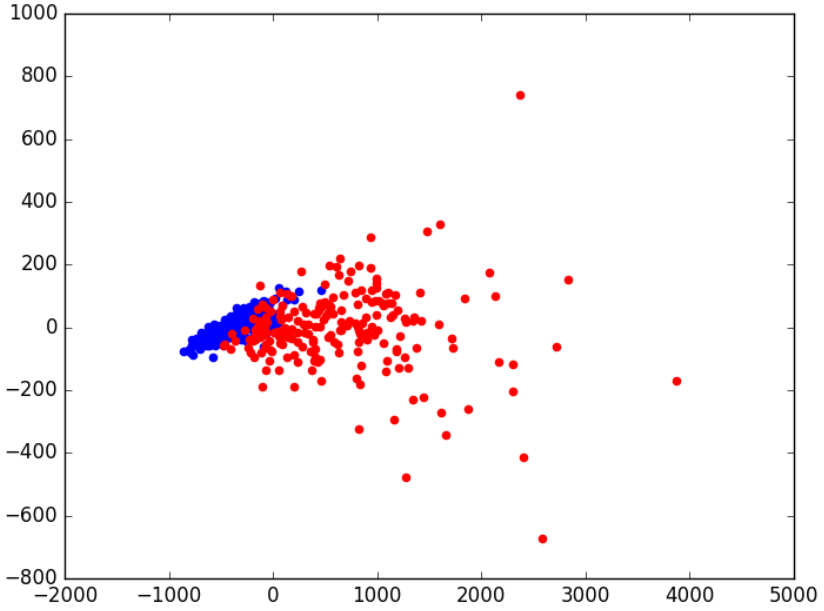
SWINBURNE UNIVERSITY OF TECHNOLOGY

k-means PCA plot in scikit-learn

PCA plot of breast cancer data coloured by *k*-means labels



PCA plot of breast cancer data coloured by true labels



Learn, Practice and Enjoy the AI journey