**Install scikit-learn**

Create isolated Python environment:

virtualenv ckd-ml -p python3.5

Activate virtualenv:

source ckd-ml/bin/activate

Run the following to install scikit-learn and mpl (to plot the result) in your virtualenv:

pip install google-api-python-client==1.6.2

pip install scikit-learn==0.19.1

pip install pandas==0.22.0

pip install scipy==1.0.0

pip install matplotlib

pip install statistics

pip install graphviz

pip install IPython

pip install joblib

**The data for this ML Model**

The [Chronic Kidney Disease Data Sets](https://archive.ics.uci.edu/ml/datasets/chronic_kidney_disease)that this sample uses for training is hosted by the [UC Irvine Machine Learning Repository](https://archive.ics.uci.edu/ml/datasets/). Citation: Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science.

Create a directory to hold the data:

mkdir ckd\_data

Download and extract the data we’ll use for this lab:

curl https://archive.ics.uci.edu/ml/machine-learning-databases/00336/Chronic\_Kidney\_Disease.rar

unrar x Chronic\_Kidney\_Disease.rar

Load the data into a pandas DataFrame to use it with Sklearn classifier.  
Train a simple model in Save the model to a file that can be used later for prediction.

First, we need to create a scikit-learn model and train it. Once we've done that, we can save the model by joblib.

Create a new file called **train.py**.

Add this to set up the imports:

import pandas as pd

import numpy as np

from statistics import median

import os

####Convert Dataset (ARFF to CSV)

Now define the format of your input data:  
Convert ARFF file to CSV including only required data under @data tag

files = [arff for arff in os.listdir('.') if arff.endswith(".arff")]

csvfun for converting arff list to csv list with header as @attribute tag

def csvfun(content):

data = False

header = ""

newContent = []

for line in content:

if not data:

if "@attribute" in line:

attri = line.split()

columnName = attri[attri.index("@attribute")+1]

header = header + columnName + ","

elif "@data" in line:

data = True

header = header[:-1]

header += '\n'

newContent.append(header)

else:

newContent.append(line)

return newContent

Main loop for reading and writing files

for f in file:

with open(f , "r") as inFile:

content = inFile.readlines()

name,ext = os.path.splitext(inFile.name)

new = csvfun(content)

with open(name+".csv", "w") as outFile:

outFile.writelines(new))

Generated csv is chronic\_kidney\_disease.csv and chronic\_kidney\_disease\_full.csv

chronic\_kidney\_disease.csv file contains 400 rows of data and 24 columns, which are then used as features and labels. To fit in different classifiers.

dataset = pd.read\_csv('chronic\_kidney\_disease.csv', header=0)

####Data Preprocessing

X = dataset.iloc[:, 0:24].values

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

Binarize Data (Make Binary)  
And replacing string values with numeric values for computation

for i in range(0,399):

if y[i] == 'ckd':

y[i] = 1

else:

y[i] = 0

y = y.astype(int)

for a in range(0, 399):

if X[a][5] == 'normal':

X[a][5] = 0

if X[a][5] == 'abnormal':

X[a][5] = 1

for a in range(0, 399):

if X[a][6] == 'normal':

X[a][6] = 0

if X[a][6] == 'abnormal':

X[a][6] = 1

for a in range(0, 399):

if X[a][7] == 'notpresent':

X[a][7] = 0

if X[a][7] == 'present':

X[a][7] = 1

for a in range(0, 399):

if X[a][8] == 'notpresent':

X[a][8] = 0

if X[a][8] == 'present':

X[a][8] = 1

for a in range(0, 399):

for b in range(18, 24):

if X[a][b] == 'yes' or X[a][b] == 'good':

X[a][b] = 0

if X[a][b] == 'no' or X[a][b] == 'poor':

X[a][b] = 1

def is\_float(input):

try:

num = float(input)

except ValueError:

return False

return True

for a in range(0,399):

for b in range(0, 24):

if(isinstance(X[a][b], int)):

X[a][b] = float(X[a][b])

elif(isinstance(X[a][b], str)):

if(is\_float(X[a][b])):

X[a][b] = float(X[a][b])

####Handling missing data

There are missing values in datasets. So here we are replacing the the value “?” with the average of the feature for the computation.

totals = [0] \* 24

added = [0] \* 24

for a in range(0, 399):

for b in range(0, 24):

if(isinstance(X[a][b], float)):

totals[b] += X[a][b]

added[b] += 1

averages = [0] \* 24

for a in range(0, 24):

averages[a] = totals[a] / added[a]

c = 0

for a in range(0, 399):

for b in range(0, 24):

if(isinstance(X[a][b], float) == 0):

X[a][b] = averages[b]

c += 1

replaceing all the missing values of the dataset with the average of nonMising value in feature

X = X.astype(float)

The data must be converted to numerical values before it gets passed to the model.  
using [pandas.to\_numeric](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_numeric.html) to convert argument(dataset) to a numeric type.

dataset[["age", "bp", "sg", "al", "su", "bgr", "bu", "sc", "sod", "pot", "hemo", "pcv", "wbcc", "rbcc"]] = dataset[["age","bp", "sg", "al", "su", "bgr", "bu", "sc", "sod", "pot", "hemo", "pcv", "wbcc", "rbcc"]].apply(pd.to\_numeric)

####Splitting datasets into (Training set-70% & Testing set-30%)

First importing [sklearn.model\_selection.train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3)

**test\_size : *float, int or None, optional (default=None)***

If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If train\_size is also None, it will be set to 0.25.

Here test\_size is 30% of the dataset and train\_size (complement of test\_size) will be 70% of the dataset.

####Train machine learning model and save the classifier to a file using joblib  
first import the classifier from sklearn.linear\_model

lg\_classifier is used assign the LogisticRegression Classifier to create ckdModel.

from sklearn.linear\_model import LogisticRegression

import joblib

lg\_classifier = LogisticRegression()

Here we are using LogisticRegression as dependent variable is binary in nature.

(ckd and notckd)  
joblib is used to save the model to a file later can be used to deploy the model with Userinterface.

Finally, create, train, and save the classifier to a file:

''' CLASSIFICATION BY LogisticRegression'''

print("Logistic Regression")

lg\_classifier.fit(X\_train, y\_train)

# Export the model to a file

joblib.dump(lg\_classifier, 'lg\_classifier.joblib')

print('Model trained and saved')

####Testing data

Model Accuracy with Confusion Matrix, Accuracy Score, f1 Score and Classification Report

print('Testing And Model Accuracy')

predictions = lg\_classifier.predict(X\_test)

from sklearn.metrics import classification\_report,confusion\_matrix,accuracy\_score,f1\_score

print("Confusion Matrix")

print(pd.crosstab(y\_test, predictions, rownames=['Label'], colnames=['Predicted'], margins=True))

print("Classification Report")

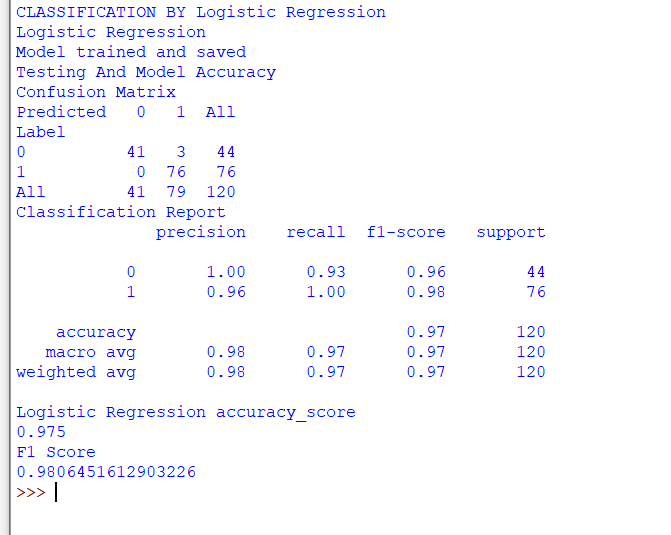
print(classification\_report(y\_test,predictions))

print("Logistic Regression accuracy\_score")

print(accuracy\_score(y\_test, predictions))

print("F1 Score")

print(f1\_score(y\_test, predictions, average='binary'))



Now creating CkdModel with k-Nearest Neighbors

First taking k = 1

''' CLASSIFICATION BY KNN '''

print("KNeighborsClassifier where k =1 ")

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors = 1)

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

knn\_y\_pred = knn.predict(X\_test)

print("KNeighborsClassifier accuracy\_score where k =1 ")

from sklearn.metrics import accuracy\_score,f1\_score

print('Accuracy Score',accuracy\_score(y\_test, knn\_y\_pred))

print('f1 Score',f1\_score(y\_test, knn\_y\_pred, average='binary'))

# finding optimum algorithm with K-Folds cross-validator

print(" K-Folds cross-validator ")

print(" K-Folds test with k =1 ")

from sklearn.model\_selection import cross\_val\_score

knn\_test\_kfolds = KNeighborsClassifier(n\_neighbors = 1)

scores\_test\_kfolds = cross\_val\_score(knn\_test\_kfolds,X,y,cv=10, scoring='accuracy')

print(scores\_test\_kfolds)

print("Mean = ",scores\_test\_kfolds.mean())

''' TEST K FOLDS'''

k\_range = range(1,50)

k\_scores =[]

for k in k\_range:

knnk = KNeighborsClassifier(n\_neighbors = k)

scores = cross\_val\_score(knnk,X,y,cv=10, scoring='accuracy')

k\_scores.append(scores.mean())

print(k\_scores)

import matplotlib.pyplot as plt

plt.plot(k\_range,k\_scores)

plt.show()

''' BEST CASE = 82.5'''

