#################################

# Week 3.1: K Nearest Neighbors #

#################################

# importing libraries

import itertools

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.ticker import NullFormatter

import pandas as pd

import matplotlib.ticker as ticker

from sklearn import preprocessing

##################

# Importing Data #

##################

# importing library

import wget

# downloading csv file using wget

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/teleCust1000t.csv'

# wget.download(url, 'teleCust1000t.csv')

# opening csv file and reading it into a variable called df

df = pd.read\_csv("teleCust1000t.csv")

# checking dataset head

print(df.head())

###################################

# Data Visualization and Analysis #

###################################

# checking frequency tabs for one of our columns

print(df['custcat'].value\_counts())

# category 3 has the highest service members

# plus service, that is

# as listed above here are the value labels for each value

# custcat is our depedent var for which we want to predict values for a new entrant

'''

1- Basic Service

2- E-Service

3- Plus Service

4- Total Service

'''

# time to plot a histogram of our customers' income groups

df.hist(column='income', bins=50)

# plt.show()

# ok, it's evident this distribution is skewed right, as always

###############

# Feature Set #

###############

# printing off some of our columns

# to hone in on key features

print(df.columns)

# converting pd df to np array for scikitlearn usage

X = df[['region', 'tenure', 'age', 'marital', 'address',

'income', 'ed', 'employ', 'retire', 'gender', 'reside']].values # .astype(float)

print(X[0:5])

# subsetting and selecting away my variable of choice

y = df['custcat'].values

print(y[0:5])

########################

# Normalizing our Data #

########################

# literally normalizes - mean 0, var 1

X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))

print(X[0:5])

# creating a test train split suited to our problem

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

# test size is 20 percent

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=4)

print ('Train set:', X\_train.shape, y\_train.shape)

print ('Test set:', X\_test.shape, y\_test.shape)

##################

# Classification #

##################

# K nearest neighbor (KNN)

from sklearn.neighbors import KNeighborsClassifier

# Training the algorithm, with k = 4

k = 4

# training model and predicting

neigh = KNeighborsClassifier(n\_neighbors = k).fit(X\_train, y\_train)

print(neigh)

# Predicting estimated(y-values) using testset(x-values) as input

yhat = neigh.predict(X\_test)

yhat[0:5]

# Evaluating model accuracy using inbuilt sklearn functions

from sklearn import metrics

print("Train set Accuracy: ", metrics.accuracy\_score(y\_train, neigh.predict(X\_train)))

print("Test set Accuracy: ", metrics.accuracy\_score(y\_test, yhat))

############

# Practice #

############

# building algorithm with k = 6

# Training the algorithm, with k = 4

k = 6

# training model and predicting

neigh = KNeighborsClassifier(n\_neighbors = k).fit(X\_train, y\_train)

print(neigh)

# Predicting estimated(y-values) using testset(x-values) as input

yhat = neigh.predict(X\_test)

yhat[0:5]

print("Train set Accuracy: ", metrics.accuracy\_score(y\_train, neigh.predict(X\_train)))

print("Test set Accuracy: ", metrics.accuracy\_score(y\_test, yhat))

# appears that the algorithm is more accurate when k = 4

Ks = 10

mean\_acc = np.zeros((Ks-1))

std\_acc = np.zeros((Ks-1))

ConfustionMx = [];

for n in range(1, Ks):

# Training our Model and Predicting

neigh = KNeighborsClassifier(n\_neighbors = n).fit(X\_train, y\_train)

yhat=neigh.predict(X\_test)

mean\_acc[n-1] = metrics.accuracy\_score(y\_test, yhat)

std\_acc[n-1]=np.std(yhat==y\_test)/np.sqrt(yhat.shape[0])

print(mean\_acc)

# plotting model-accuracy for different numbers of neighbors

plt.plot(range(1,Ks), mean\_acc,'g')

plt.fill\_between(range(1,Ks),mean\_acc - 1 \* std\_acc,

mean\_acc + 1 \* std\_acc,

alpha=0.10)

plt.legend(('Accuracy ', '+/- 3xstd'))

plt.ylabel('Accuracy ')

plt.xlabel('Number of Nabors (K)')

plt.tight\_layout()

plt.show()

print( "The best accuracy was with", mean\_acc.max(), "with k=", mean\_acc.argmax()+1)

# in order to display plot within window

# plt.show()

############################

# Week 3.2: Decision Trees #

############################

# importing libraries

import numpy as np

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

##################

# Importing Data #

##################

# importing library

import wget

# downloading csv file using wget

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/drug200.csv'

# wget.download(url, 'drug200.csv')

# reading in our csv on pharmaceutical

my\_data = pd.read\_csv("drug200.csv", delimiter=",")

# checking our dataset, high level

my\_data[0:5]

############

# Practice #

############

# checking dataset dimensions

my\_data.size

my\_data.shape

##################

# Pre-processing #

##################

# subsetting

X = my\_data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na\_to\_K']].values

X[0:5]

# preprocessing our data

from sklearn import preprocessing

##############labeling################

# creating and applying label for column 1

le\_sex = preprocessing.LabelEncoder()

le\_sex.fit(['F', 'M'])

X[:,1] = le\_sex.transform(X[:,1])

# creating and applying label for column 2

le\_BP = preprocessing.LabelEncoder()

le\_BP.fit([ 'LOW', 'NORMAL', 'HIGH'])

X[:,2] = le\_BP.transform(X[:,2])

# creating and applying label for column 3

le\_Chol = preprocessing.LabelEncoder()

le\_Chol.fit([ 'NORMAL', 'HIGH'])

X[:,3] = le\_Chol.transform(X[:,3])

X[0:5]

#############target#var###############

# declaring target variable

y = my\_data["Drug"]

y[0:5]

################################

# Setting up our Decision Tree #

################################

# importing preprocessing package

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

# creating a test train split using python package

X\_trainset, X\_testset, y\_trainset, y\_testset = train\_test\_split(

X, y, test\_size=0.3, random\_state=3)

# Practice

# 1: display shapes and size of trainsets

# size means num of cells, wheras shape shows dimension

X\_trainset.size

X\_trainset.shape

y\_trainset.size

y\_trainset.shape

# 2: display shapes and size of testsets

X\_testset.size

X\_testset.shape

y\_testset.size

y\_testset.shape

# naturally, both testsets and both trainsets have the same

# number of rows

# naturally, both x datasets and both y dataset have the same

# number of columns

#####################

# Modeling our Data #

#####################

# creating decision tree object

drugTree = DecisionTreeClassifier(criterion="entropy", max\_depth = 4)

drugTree

# fitting decision tree classifications to our training data

drugTree.fit(X\_trainset, y\_trainset)

###############################

# Prediction on our test data #

###############################

# using decision tree object to predict test data classification

predTree = drugTree.predict(X\_testset)

# printing outcome

predTree [0:5]

y\_testset [0:5]

# model performs fairly well, predicts all values correctly

###################################

# Evaluation of the Decision Tree #

###################################

# importing scoring mechanism

from sklearn import metrics

import matplotlib.pyplot as plt

# super high accuracy as we can see

# print("DecisionTree's Accuracy: ", metrics.accuracy\_score(y\_testset, predTree))

#########################

# Visualization Tactics #

#########################

from sklearn.externals.six import StringIO

import pydotplus

import matplotlib.image as mpimg

from sklearn import tree

# from sklearn.tree import export\_graphviz

import sklearn

print(sklearn.\_\_file\_\_)

dot\_data = StringIO()

# filename = "drugtree.png"

'''

featureNames = my\_data.columns[0:5]

targetNames = my\_data["Drug"].unique().tolist()

out = tree.export\_graphviz(drugTree, feature\_names=featureNames,

out\_file=dot\_data, class\_names= np.unique(y\_trainset),

filled=True, special\_characters=True,rotate=False)

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png(filename)

img = mpimg.imread(filename)

plt.figure(figsize=(100, 200))

plt.imshow(img,interpolation='nearest')

'''

# skipping this funky part because graphviz refuses

# to operate in python 3.6

# i'm sure it's just some funk to do with adding

# the path to the environmental variables.

# nonetheless, only the graphing tool is off.

# the decision tree predictor still works a ok :) just fine!

# so technically i skipped the decision tree graphing portion, big deal.

# i retained the crux of the matter.

# also, tried plotting online, on ibm. took wayy too long to load.

# can return if necessary.

# in order to display plot within window

# plt.show()

#################################

# Week 3.3: Logistic Regression #

#################################

# importing libraries

import pandas as pd

import pylab as pl

import numpy as np

import scipy.optimize as opt

from sklearn import preprocessing

import matplotlib.pyplot as plt

##################

# Importing Data #

##################

# importing library

import wget

# downloading Customer Churn project data using wget

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/ChurnData.csv'

# wget.download(url, 'ChurnData.csv')

# opening churn data and reading it into a variable called churn\_df

churn\_df = pd.read\_csv("ChurnData.csv")

# checking dataset head

churn\_df.head()

###########################

# Pre-processing our data #

###########################

# subsetting to releveant columns

churn\_df = churn\_df[['tenure', 'age', 'address', 'income', 'ed',

'employ', 'equip', 'callcard', 'wireless',

'churn']]

# changing var type

churn\_df['churn'] = churn\_df['churn'].astype('int')

churn\_df.head()

##############

# Practicing #

##############

# checking for dimensions

churn\_df.shape

# X is the feature variables set, y is the target output

X = np.asarray(churn\_df[['tenure', 'age', 'address', 'income',

'ed', 'employ', 'equip']])

X[0:5]

# making churn the target variable

y = np.asarray(churn\_df['churn'])

y[0:5]

# also normalizing the dataset

X = preprocessing.StandardScaler().fit(X).transform(X)

X[0:5]

#############################

# Creating Test Train Split #

#############################

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

# 20 - 80 split, 4 folds

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X,

y, test\_size=0.2, random\_state=4)

# printing test sets, and train sets side by side

# print ('Train set:', X\_train.shape, y\_train.shape)

# print ('Test set:', X\_test.shape, y\_test.shape)

###################################

# Modeling: Logit w/ Scikit Learn #

###################################

# importing libraries

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

# iteration 1: inverse regularization = .01, solver = liblinear

# fitting regression model to our training dataset

LR = LogisticRegression(C=0.01, solver='liblinear').fit(X\_train,y\_train)

LR

# predicting outcome variable of interest

yhat = LR.predict(X\_test)

yhat

# this returns probabilities of all binary outcomes yhat

yhat\_prob = LR.predict\_proba(X\_test)

yhat\_prob

############################################

# Evaluating our Logistic Regression model #

############################################

# importing scoring metric

from sklearn.metrics import jaccard\_similarity\_score

#################

# jaccard index #

#################

print(jaccard\_similarity\_score(y\_test, yhat))

# so the model performs pretty well - scoring a .750,

# or 75% similarity levels

###################################

# constructing a confusion matrix #

###################################

from sklearn.metrics import classification\_report, confusion\_matrix

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

'''printing and plotting the confusion matrix

can normalize using option, normalize=True

'''

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normlized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

# creating plot features

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

# formatting it to our liking

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

confusion\_matrix(y\_test, yhat, labels=[1,0])

# computing confusion matrix, to predict false positives, and false negatives

cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[1,0])

np.set\_printoptions(precision=2)

# plotting non-normalized confusion matrix

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['churn=1',

'churn=0'], normalize= False, title='Confusion matrix')

# plt.show()

# beautiful display of false positives and false negatives

# printing out our vals for comparison

classification\_report(y\_test, yhat)

# log loss calculations

from sklearn.metrics import log\_loss

print(log\_loss(y\_test, yhat\_prob))

############

# practice #

############

# iteration 2: inverse regularization = .05, solver = sag

# fitting new regression model to our training dataset

LR2 = LogisticRegression(C=0.05, solver='sag').fit(X\_train,y\_train)

LR2

# predicting outcome variable of interest

yhat2 = LR2.predict(X\_test)

yhat2

# this returns probabilities of all binary outcomes yhat

yhat\_prob2 = LR2.predict\_proba(X\_test)

yhat\_prob2

# evaluating using jacard index

print(jaccard\_similarity\_score(y\_test, yhat2))

# so the model performs fairly well - scoring a .725, though less well than 1st iteration model

# or 72.5% similarity levels

# evaluating using confusion matrix

confusion\_matrix(y\_test, yhat2, labels=[1,0])

# confusion matrix object, for plotting

cnf\_matrix2 = confusion\_matrix(y\_test, yhat2, labels=[1,0])

np.set\_printoptions(precision=2)

# plotting non-normalized confusion matrix

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['churn=1',

'churn=0'], normalize= False, title='Confusion matrix')

plt.show()

# printing out fp's, and fn's for comparison

classification\_report(y\_test, yhat2)

# evaluating using log loss

print(log\_loss(y\_test, yhat\_prob2))

# the non-normalized confusion matrixes show

# that the two models perform essentially the same on classification

# though jaccard indices show one is closer to the true values

# while the other isn't

# in order to display plot within window

# plt.show()

####################################

# Week 3.4: Support Vector Machine #

####################################

# importing libraries

import pandas as pd

import pylab as pl

import numpy as np

import scipy.optimize as opt

from sklearn import preprocessing

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

import matplotlib.pyplot as plt

##########################

# Loading Cancer dataset #

##########################

# importing library

import wget

# download, and save

# url = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/cell\_samples.csv'

# wget.download(url, 'cell\_samples.csv')

# read in, print

cell\_df = pd.read\_csv("cell\_samples.csv")

cell\_df.head()

# Step 1:

'''

since class (benign, or malignant) is what we want to predict

we set it aside as our target variable

'''

# plotting our data points as an overlayed scatterplot

# plot 1: class vs unifsize

'''

ax = cell\_df[cell\_df['Class'] == 4][0:50].plot(kind='scatter', x='Clump',

y = 'UnifSize', color='DarkBlue', label='malignant');

'''

# plot 2: class vs clump, superimposed

'''

cell\_df[cell\_df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize',

color='Yellow', label='benign', ax=ax);

'''

# plt.show()

##############################

# Step 2: Data preprocessing #

##############################

# getting an estimate of data types

cell\_df.dtypes

# dropping non numeric rows from barenuc

cell\_df = cell\_df[pd.to\_numeric(cell\_df['BareNuc'], errors='coerce').notnull()]

# converting bare nuclei to an integer.

cell\_df['BareNuc'] = cell\_df['BareNuc'].astype('int')

cell\_df.dtypes

# subsetting for predictor vars

feature\_df = cell\_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh',

'SingEpiSize', 'BareNuc', 'BareNuc', 'BlandChrom',

'NormNucl', 'Mit']]

# converting to arrays

X = np.asarray(feature\_df)

X[0:5]

# doing the same for outcome variables

cell\_df['Class'] = cell\_df['Class'].astype('int')

y = np.asarray(cell\_df['Class'])

y[0:5]

#############################

# Step 3: Modeling our Data #

#############################

# creating a test and training split

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.2, random\_state=4)

# good way to size up the shapes and sizes of split in 1 line of code.

# print ('Train set:', X\_train.shape, y\_train.shape)

# print ('Test set:', X\_test.shape, y\_test.shape)

# using default rbf for modeling in svm: radial basis function

from sklearn import svm

# declaring and fitting object

clf = svm.SVC(kernel='rbf')

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

# predicting outcome values

yhat = clf.predict(X\_test)

yhat [0:5]

# there is no way around choosing different models,

# and comparing results, to then choose the best performing model

##############################

# Step 3: Evaluation metrics #

##############################

from sklearn.metrics import classification\_report, confusion\_matrix

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

the current function print and plots confusion matrix

normalization can be applied by setting parameter normalize = true

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2

for i, j in itertools.product(range(cm.shape[0]),

range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

# computing confusion matrix for current data

cnf\_matrix = confusion\_matrix(y\_test, yhat, labels=[2,4])

np.set\_printoptions(precision=2)

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

# plotting our non normalized matrix

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['Benign (2)',

'Malignant (4)'], normalize= False,

title='Confusion matrix')

plt.show()

# using the f1\_score for scoring performance

from sklearn.metrics import f1\_score

print(f1\_score(y\_test, yhat, average='weighted'))

# using the jaccard index for scoring performance

from sklearn.metrics import jaccard\_similarity\_score

print(jaccard\_similarity\_score(y\_test, yhat))

########################################

# Step 4: Practice metrics, iterations #

########################################

# iterating with a different functional type: linear

clf2 = svm.SVC(kernel='linear')

clf2.fit(X\_train, y\_train)

# predicting outcome values

yhat2 = clf2.predict(X\_test)

print(yhat2[0:5])

# computing confusion matrix with new function fitted

cnf\_matrix2 = confusion\_matrix(y\_test, yhat2, labels=[2,4])

np.set\_printoptions(precision=2)

print(classification\_report(y\_test, yhat2))

# plotting our non normalized matrix

plt.figure()

plot\_confusion\_matrix(cnf\_matrix2, classes=['Benign (2)',

'Malignant (4)'], normalize= False,

title='Confusion matrix 2')

plt.show()

# using the f1\_score for scoring linear model performance

print(f1\_score(y\_test, yhat2, average='weighted'))

# using the jaccard index for scoring linear model performance

print(jaccard\_similarity\_score(y\_test, yhat2))

# in order to display plot within window

# plt.show()