from sklearn import preprocessing

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.cross\_validation import train\_test\_split

import tensorflow.contrib.learn as skflow

from sklearn import svm, datasets

from sklearn.metrics import confusion\_matrix

import shutil

import os

# Encode text values to dummy variables(i.e. [1,0,0],[0,1,0],[0,0,1] for red,green,blue)

def encode\_text\_dummy(df,name):

dummies = pd.get\_dummies(df[name])

for x in dummies.columns:

dummy\_name = "{}-{}".format(name,x)

df[dummy\_name] = dummies[x]

df.drop(name, axis=1, inplace=True)

# Encode text values to indexes(i.e. [1],[2],[3] for red,green,blue).

def encode\_text\_index(df,name):

le = preprocessing.LabelEncoder()

df[name] = le.fit\_transform(df[name])

return le.classes\_

# Encode a numeric column as zscores

def encode\_numeric\_zscore(df,name,mean=None,sd=None):

if mean is None:

mean = df[name].mean()

if sd is None:

sd = df[name].std()

df[name] = (df[name]-mean)/sd

# Convert all missing values in the specified column to the median

def missing\_median(df, name):

med = df[name].median()

df[name] = df[name].fillna(med)

# Convert a Pandas dataframe to the x,y inputs that TensorFlow needs

def to\_xy(df,target):

result = []

for x in df.columns:

if x != target:

result.append(x)

# find out the type of the target column. Is it really this hard? :(

target\_type = df[target].dtypes

target\_type = target\_type[0] if hasattr(target\_type, '\_\_iter\_\_') else target\_type

print(target\_type)

# Encode to int for classification, float otherwise. TensorFlow likes 32 bits.

if target\_type in (np.int64, np.int32):

# Classification

return df.as\_matrix(result).astype(np.float32),df.as\_matrix([target]).astype(np.int32)

else:

# Regression

return df.as\_matrix(result).astype(np.float32),df.as\_matrix([target]).astype(np.float32)

# Nicely formatted time string

def hms\_string(sec\_elapsed):

h = int(sec\_elapsed / (60 \* 60))

m = int((sec\_elapsed % (60 \* 60)) / 60)

s = sec\_elapsed % 60

return "{}:{:>02}:{:>05.2f}".format(h, m, s)

# Regression chart, we will see more of this chart in the next class.

def chart\_regression(pred,y):

t = pd.DataFrame({'pred' : pred.flatten(), 'y' : y\_test.flatten()})

t.sort\_values(by=['y'],inplace=True)

a = plt.plot(t['y'].tolist(),label='expected')

b = plt.plot(t['pred'].tolist(),label='prediction')

plt.ylabel('output')

plt.legend()

plt.show()

def plot\_confusion\_matrix(cm, names, title='Confusion matrix', cmap=plt.cm.Blues):

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

plt.title(title)

plt.colorbar()

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

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

plt.yticks(tick\_marks, names)

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

path = "./data/"

filename = os.path.join(path,"crx.csv")

df = pd.read\_csv(filename,na\_values=['NA','?'])

# Encode feature vector

encode\_numeric\_zscore(df,'a2')

encode\_numeric\_zscore(df,'s3')

encode\_numeric\_zscore(df,'a8')

encode\_numeric\_zscore(df,'a11')

encode\_numeric\_zscore(df,'a14')

encode\_numeric\_zscore(df,'a15')

encode\_text\_dummy(df, 'a1')

encode\_text\_dummy(df, 'a4')

encode\_text\_dummy(df, 'a5')

encode\_text\_dummy(df, 'a6')

encode\_text\_dummy(df, 'a7')

# encode\_text\_dummy(df, 'a9')

encode\_text\_dummy(df, 'a10')

encode\_text\_dummy(df, 'a12')

encode\_text\_dummy(df, 'a13')

encode\_text\_dummy(df, 'a9')

missing\_median(df, 'a2')

missing\_median(df, 'a14')

#species = encode\_text\_index(df,"a16")

a16 = encode\_text\_index(df,'a16')

num\_classes = len(a16)

# Create x & y for training

# Create the x-side (feature vectors) of the training

x, y = to\_xy(df,'a16')

# Split into train/test

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

x, y, test\_size=0.25, random\_state=42)

# Create a deep neural network with 3 hidden layers of 10, 20, 10

##############class 12, 用model = Sequential() 附近的部分？

classifier = skflow.DNNClassifier(hidden\_units=[10, 20, 10], n\_classes=num\_classes,

epochs=10000)

# Early stopping

early\_stop = skflow.monitors.ValidationMonitor(x\_test, y\_test,

early\_stopping\_rounds=200, print\_steps=50, n\_classes=num\_classes)

# Fit/train neural network

classifier.fit(x\_train, y\_train, early\_stop)

# Measure accuracy

score = metrics.accuracy\_score(y, classifier.predict(x))

print("Final accuracy: {}".format(score))

pred = classifier.predict(x\_test)

# Compute confusion matrix

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

np.set\_printoptions(precision=2)

print('Confusion matrix, without normalization')

print(cm)

plt.figure()

plot\_confusion\_matrix(cm, a16)

# Normalize the confusion matrix by row (i.e by the number of samples

# in each class)

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

print('Normalized confusion matrix')

print(cm\_normalized)

plt.figure()

plot\_confusion\_matrix(cm\_normalized, a16, title='Normalized confusion matrix')

plt.show()