Machine Learning Project: Book Classifier

January 20, 2016

I decided to write a simple command-line program that will allow a user to enter in a book manuscript in text file format. The program will then classify and assign a label to this new book as either belonging to class1(emergent reader) or class2(non-emergent reader) based on a learned classifier.

Before I begin I would like my reader(s) to know that what I present here is an idea that came to my mind as an application and exercise in machine learning and not a tutorial. I will not explain terms or concepts. There are many great tutorials on the internet and a general search will bring them up.

There are five parts to this presentation:

1. Part 1. Introduction. a general overview of the entire project.
2. Part 2. Setting Up The Machine Learning Environment. I present how I set up my ML environment.
3. Part 3. Using The Predictor Model on New Data. I show how I use the model on new data.
4. Part 4. Conclusion. general summary.
5. Part 5. ML Project: Book Classifier Source Codes

Part 1. Introduction

This is a supervised learning project that uses a classifier. We make use of a scientific Python package sckit-learn, NumPy and SciPy.

The general steps taken were obtain data, clean data, reformat data such that classifier can use it, split data into training and test sets, train classifier, store the model for future use, and then use the model on new data sets.

Parts 2 will present information on setting the ML environment.

Part 2. Setting Up The Machine Learning Environment

For this project functions were written in Python to obtain data,clean data. The source codes are not shown here. If you are interested in seeing the code they are in repository in text file format. And are only there for presentation and not meant to be run, compiled or used in any way.

Below I present the source code for classifying the data, and persisting the model for future use.

Next, in Part 3 the persisted model is used on new data.

Part 3. Using The Predictor Model on New Data

In Part 2 a ML environment was created and a model was persisted. Now I show the source code for a simple command-line program that makes use of the persisted model to classify new data, which in this case are new books.

Image of the screen output for command-line program:

```
>>> import CategorizeBook_v2 as cb
>>> cb.categorizerApplication()
manuscript file name: book_1.txt
text:  ['This', 'is', 'a', 'book', 'about', 'me.', 'Me', 'is', 'a', 'flower.',
The', 'flower', 'is', 'red', 'and', 'blue.', 'The', 'red', 'and', 'blue', 'flow
r', 'belongs', 'to', 'me.']
the book category is:  emergent reader
would you like to classify another manuscript? Y or N:
You quit the program
>>> n
```

Next in Part 4 a summary of this project is presented.

Part 4. Conclusion

This project was a simple exercise in Machine Learning using a classifier to predict new labels for unknown data. In our case the labels were class1 emergent readers and class2 non-emergent readers.

This project machine learning was done in Python. In a future project I will perform machine learning with the same data set in R.

Part 5. ML Project Book Classifier Source Codes

source code for CreateData.py

```
''' ============================================================================
=========================
    CreateData.py
    Author: Donnette Bowler
    copyright: copyright © Donnette Bowler 2016. All rights reserved. No part of this
document may be reproduced or distributed.

============================================================================
```

```
=============

    I wrote functions to collect data from text files of manuscripts.
    The manuscripts are text files in string format. Essentially these functions load
    all manuscript or book files in a directory, open and read into a string. break s
tring into a list.

    I wrote a function to categorize books based on a threshold ratio of average
    frequency of repeating words to average distance between each repeating word.
    I used this function to categorize the books. Each books average frequency and
    average distance, and book category was output to a text file.

================================================================================
==================================

'''



#scientific Python imports
import numpy as np
import pandas as pd
import os
```

```
#dependent Python imports
import MyLinguisticAnalysis_2 as MLA


def createTestData():
    input_dir='./textBooks/'
    files=os.listdir(input_dir)
    test_input_dir='./testData/'
    test_files=os.listdir(test_input_dir)
    freq_result_list=[]
    avg_dis_result_list=[]
    category_book_result_list=[]
    for f in files:
        str_list=[]
        print 'thses are the files in your directory: ', f
        full_name=os.path.join(input_dir,f )
        fr=open(full_name)#open and read files
        str=fr.read()
        #get substring

        str_list=str.split()
        print "text: ", str_list
        #perform a binary map creation on each files that maps word repeats
        analysis=MLA.gatherAnalysis(str_list)
```

```
        #print " this is the analysis on the text file: ", analysis
        # print "this is frequency: ",analysis['frequencies']
        #print "this is average distance: ", analysis['average_distance']

        freq =analysis['total_frequency']
        avg_dis=analysis['total_average_distance']
        #get category of book
        if avg_dis==0:
            category_book=1
        else:
            category_book=compareThreshValue(freq,avg_dis)
        print 'this is the book category: ',category_book
        freq_result_list.append(freq)
        avg_dis_result_list.append(avg_dis)
        category_book_result_list.append(category_book)

        #create a pandas table to hold analytics (frequency, average distance)

        #output that analysis to testData.txt file
        #get data from testData.txt file, categorize based on threshold value, and ap
pend to testData.txt
    s=pd.DataFrame.from_items([('',freq_result_list),('',avg_dis_result_list),('',cat
egory_book_result_list)])
    s.to_csv(os.path.join(test_input_dir,'testData.txt'),index=False,sep='\t')



def createData(filename):
    input_dir='./tbooks/'
    full_name=os.path.join(input_dir,filename)
    fr=open(full_name)
    str=fr.read()
    str_list=str.split()
    print 'text: ', str_list
    #perform a binary map creation on each files that maps word repeats
    analysis=MLA.gatherAnalysis(str_list)
    #print " this is the analysis on the text file: ", analysis
    # print "this is frequency: ",analysis['frequencies']
    #print "this is average distance: ", analysis['average_distance']
    freq =analysis['total_frequency']
    avg_dis=analysis['total_average_distance']
    return freq,avg_dis



def compareThreshValue(freq,dis): #threshold value is 1.6. higher the frequency and l
esser the distance the value increases
    value= freq/dis
```

```
    print 'this is the value: ',value
    if value >0.6 :
        return 1
    if value <0.6:
        return 2
    if value ==0.6:
        return 1



def mla():
    analysis=MLA.testDefinitions()
    print 'this is the result of the test: ', analysis
```

source code for MyLinguisiticAnalysis_2.py

```
''' ================================================================================
================================
    MyLinguisticAnalysis.py
    Author: Donnette Bowler
    copyright: copyright © Donnette Bowler 2016. All rights resevered. No part of thi
s document may be reproduced or distributed.

================================================================================
==============================

   This contains helper functions to perform linguisitic analysis of text files.


    Create a binary map of words in the string. Calculate frequency of each word (of
course,
    here I could have used  libraries eg. nltk- natural language toolkit,
    and other scientific libraries to get word frequencies, but I thought it would be
fun to create my own.
    I also wrote a reverse mapping function to recreate the original string.
    I used the binary map to calculate word frequencies and word distances).


================================================================================
==============================

    '''



"""this contains functions to perform linguistic analysis of interests."""
```

```
def items_same(m,n):
    """compares two items and returns a list"""
    if m==n:
        return True



def thenAddToList(n,aList):
    """returns appended list"""
    blist=aList.append(n)
    return blist



def makeDict(aList):

    #if list is empty
    if len(aList)== 0:
        return None #return none

    #else create a dictionary to store values
    else:
        aDict={}
        for item in aList:
            if item not in aDict: #if the item is not in the dictionary
                aDict[item]=[]

        return aDict




def findMatches(alist,aDict):
    """counts the number of times a key in a dictionary, aDict, appear in a list, ali
st"""
    bDict=aDict
    #if the list, alist or the dictionary, aDict is empty, then return none
    if len(alist)==0 | len(bDict)==0:
        return None

    else:
        word_loc=[] #initialize list to store word location of each item (binary)
        v=0

        #count the number of times a key appears in alist and append it to aDict[key]
        counter=0
```

```
        for key in bDict:
            counter=0
            v=0
            for item in alist:

                #print "comparing key:, %s and item:, %s " %(key,item)
                if key==item:
                    counter=counter+1
                    v=counter

                    #print "match found, %d", v
                else:
                    e=1
                    #print "no match found. counter is still: %d" % (v )
                    #v+=counter
            bDict[key]=v
            rDict=bDict

    return rDict



def keyMap(alist,aDict):
    '''this function takes a list and a map of unique items. the function creates a b
inary map of list to dictionary.
    this will make analysis easier. Each key contains its own binary map'''

    bDict=aDict
    #if the list, alist or the dictionary, aDict is empty, then return none
    if len(alist)==0 | len(bDict)==0:
        return None

    else:
        word_loc=[] #initialize list to store word location of each item (binary)
        v=0

        loc=0

        #count the number of times a key appears in alist and append it to aDict[key]
        counter=0
        for key in bDict:
            counter=0
            v=0
            word_loc=[]

            for item in alist:

                # print "comparing key:, %s and item:, %s " %(key,item)
                if key==item:
                    counter=counter+1
```

```
                    v=counter
                    # print "match found, %d", v
                    loc=1
                    word_loc.append(loc)
                else:

                    #print "no match found. counter is still: %d" % (v )
                    loc=0
                    word_loc.append(loc)
                #add v to word_loc list


            bDict[key]=word_loc
            rDict=bDict

        return rDict


def lengthDictItem(rDict):
    count=0

    for key in rDict:

        alist=rDict[key]
        for item in alist:
            count=count+1

        return count



def reverseKeyMap(rDict):
    '''this is a function to reverse or translate the keyMap with a dictionary. used
for translation and also to verify that the mapping function
        was correct'''
    fx=rDict
    slist=[""]*lengthDictItem(fx)
    rlist=[]
    if len(fx)==0:
        return None
    else:
        #for every key in fx, for every item in list
        for key in fx:
            counter=0 #keeps track of location in list
            alist=[]
            blist=slist
            alist=fx[key]
            loc_index=0
            for item in alist:
```

```
            #print "finding: ", key

        #find where item equals 1, and return its position as value of loc_index
            if item==1:
                loc_index=counter
                # print "location: ",loc_index



                if loc_index==0:
                    blist[0]=key
                # print blist
                else:
                    # print "loc_index-1: ",loc_index-1
                    blist[loc_index]=key
            #print blist
            counter=counter+1
            #print "counter: ",counter
        rlist=blist

return rlist
```

```
def calculateKeyFreq(aDict):
    '''this function keeps a count of the instances it encounters a '1' for each list
in
        the key in the dictionary. it returns a new dictionary with this count for ea
ch key'''

    bDict=aDict
    rDict={}
    if len(bDict)==0:
        #print "returning none. length of bDict: ", len(bDict)
        return None
    else:
        for key in bDict:
            alist=[]
            alist=bDict[key]
            # print "alist: ",alist
            counter=0 #initialize the counter to 0
            #for every item in the list, when item==1, increase counter
            for item in alist:
                # print "item: ",item
                if item==1:
                    counter=counter+1 #increment the counter by 1
            rDict[key]=counter #store the counter value in the dictionary at the key
        return rDict
```

```
def calDist(a_sublist):

    length=len(a_sublist)
    dist=0
    val=0
    rlist=[]
    #print "length of sublist: ", len(a_sublist)
    if len(a_sublist)==1:
        val=0
        rlist.append(val)
        return rlist

    for item in a_sublist:
        counter=0
        if counter!=len(a_sublist):
            val=(a_sublist[counter+1]-a_sublist[counter])-1
            rlist.append(val)
        else:
            rlist.append(val)
            return val
        counter=counter+1
    return rlist




def calculateItemDist(aDict):
    '''this function  calculates the distance between two items, where items==1'''
    bDict=aDict
    freq_list=[]
    occur_list=[]
    rDict={}
    rm=0
    rn=0
    if len(aDict)==0:
        return None
    else:
        #for key in bDict, get the frequency of 1's, and create a list of length freq
uency-1
        fDict={}
        fDict=calculateKeyFreq(bDict)
        for key in bDict:
            alist=[]
            alist=bDict[key]
```

```
        # print "key: ",key

        if fDict[key]==1:
            freq_list=[]
            freq_list.append(0)



        else:
            acounter=0
            bcounter=0
            m =0
            n=0
            dist=0
            #print "fdict[key]: ",fDict[key]
            val=fDict[key]
            freq_list=[]  #create a frequency list
            #print "freq list: ",len(freq_list)
            #calculate distance
            for item in alist:

                m=0
                n=0

                if item==1 :
                    m=acounter

                    rm=m
                    # print "acounter: ",acounter

                        #print "rm: ",rm

                    freq_list.append(rm)

                        #print " the same"
                acounter=acounter+1
                #print "bcounter: ",bcounter


                #print "acounter: ",acounter



            rDict[key]=freq_list
            rDict[key]=calDist(freq_list)
        return rDict



def calculateAvgDist(aDict):
```

```
    '''this function calculates the average distance between repeating words'''
    bDict=aDict
    length_list=0
    item_distance=0
    if len(bDict)==0:
        return None

    else:
        #for key in bDict, get the length of the list, and add the items (integers) i
n the list
        for key in bDict:
            alist=[]
            alist=bDict[key]
            length_list+=len(bDict[key])
            for item in alist:
                get_item=item
                item_distance+=get_item
        average_distance=item_distance/length_list
        return average_distance


def avgDist(aDict):
    rDict={}
    for key in aDict:
        counter=0
        item=0
        num=0
        alist=aDict[key]
        for item in alist:
            num=num+item
            counter=counter+1
        rDict[key]=num/counter
    return rDict




def gatherAnalysis(alist):
    #calls methods to gather analysis, and returns a dictionary containing all analys
is

    blist=alist

    summary_analysis={}
    a=makeDict(blist)
    b=findMatches(blist,a)
    c=keyMap(alist,b)
    # print "map: ", c
```

```
    d=calculateKeyFreq(c)
    #print "frequencies: ",d
    e=calculateItemDist(c )
    #print "word distances: ",e
    f=avgDist(e)
    summary_analysis['map']=c
    summary_analysis['frequencies']=d
    summary_analysis['distances']=e
    summary_analysis['average_distance']=f

    sumthis=0
    sumDist=0
    count_1=0
    count_2=0
    #print "before loop in dict"

    sumanl=summary_analysis['frequencies']
    for key in sumanl:

        val=sumanl.get(key,0)
        #print "this is the value: ",val
        sumthis+=val
        count_1+=1

    for key in summary_analysis['average_distance']:
        val=summary_analysis['average_distance'].get(key,0)
        #print "this is the value for average distance: ",val
        sumDist+=val
        count_2+=1

# print "sumthis: ",sumthis
# print "count: ",count_1
#print "sumdis: ",sumDist

    eval_1=float(sumthis)/float(count_1)
    eval_2=float(sumDist/count_2)
    summary_analysis['total_frequency']=eval_1
    summary_analysis['total_average_distance']=eval_2




    #summary_analysis= reverseKeyMap(c)

    return summary_analysis


def testDefinitions():
```

```
    T=['hello','are','you','okay','you','okay']
    analysis=gatherAnalysis(T)

    return analysis
```

source code for BookLabelPredictor_v3.py

```
''' ================================================================================
====================
    BookLabelPredictor.py
    Author: Donnette Bowler
    copyright: copyright © Donnette Bowler 2016. All rights reserved. No part of this
document may be reproduced or distributed.

================================================================================

    The data that we are interested in is stored in a text file.
    First we load the data into numpy array of form (n-samples,features),
    where features are a 2D numpy array, and n-samples is the number of lines
    in the file. The last column of the file represents the target labels of the data
.

    This predictor uses k-nearest neighbours to predict a label
    on a new book.



================================================================================
========

'''


#standard scientific Python imports
import matplotlib.pyplot as plt
import numpy as np

#import classifiers and performance metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn import linear_model,neighbors,svm


import os
```

```
# a function that loads the data from a text file into arrays
def loadData():

    input_dir="./testData"
    filename="testData.txt"
    full_name=os.path.join(input_dir,filename)
    fr=open(full_name)
    number_of_lines=len(fr.readlines()) #get the number of lines in text file for the
number of observations
    book_X=zeros((number_of_lines,2)) # initialize an 2D array and fill with zeros
    book_Y=zeros((number_of_lines,)) #initialize a 1D array and fill with zeroes

    index=0
    fr=open(full_name)
    for line in fr.readlines():

        line=line.strip()
        list_from_line=line.split('\t')
        book_X[index,:]=list_from_line[0:2] #array of features

        book_Y[index:,]=list_from_line[-1] #array of category or target values

        index+=1

    return book_X,book_Y



# a function to create an array of randomly ordered indices
def permutateData(input_list):
    np.random.seed(0) #set the random seed
    indices=np.random.permutation(len(input_list)) #randomly order the indices
    return indices


#create a classifier: nearest-neighbor classifier
def classify0():
    bookX,bookY=loadData()

    book_indices=permutateData(bookX) #create a random list of indices


    n_samples=len(bookX)

    #take a subset of the data for training the classifier
    book_X_train=bookX[:0.5*n_samples]
    book_Y_train=bookY[:0.5*n_samples]
```

```
    #take a subset of the data for testing the classifier
    book_X_test=bookX[0.5*n_samples:]
    book_Y_test=bookY[0.5*n_samples:]


    #create a classifier
    knn=KNeighborsClassifier(n_neighbors=5)

    #train the classifier with training data
    v=knn.fit(book_X_train,book_Y_train)
    #use the trained classifier on the test data set to predict target values
    prediction=knn.predict(book_X_test)
    actual=book_Y_test

    #compare the predicted target values with the actual target values of the data
    print "predicted values: ", prediction
    print "actual values: ", actual
    print v


def modelBook():
    bookX,bookY=loadData() #load the data

    book_indices=permutateData(bookX) #create a random list of indices

    n_samples=len(bookX)

    #take a subset of the data for training the classifier
    book_X_train=bookX[:0.5*n_samples]
    book_Y_train=bookY[:0.5*n_samples]
    #take a subset of the data for testing the classifier
    book_X_test=bookX[0.5*n_samples:]
    book_Y_test=bookY[0.5*n_samples:]

    #create a classifier
    logistic=linear_model.LogisticRegression()
    #score the performance of the KNN classifier
    print ('KNN score: %f' %knn.fit(book_X_train,book_Y_train).score(book_X_test,book
_Y_test))

    #fit the data to the regression line and score its performance.
    print ('logisticsRegression score: %f' % logistic.fit(book_X_train,book_Y_train).
score(book_X_test,book_Y_test))

    #linear regression on data
    regr=linear_model.LinearRegression()

    #fit training data
    regr.fit(book_X_train,book_Y_train)
```

```
    #score the performance of classifier and print the result to the screen
    print "regression score: (note a variance score of 1 is perfect prediction and 0
means no linear relationship): ", regr.score(book_X_test,book_Y_test)
```

source code for dataBookViz.py

```
''' ============================================================================
==

    dataBookViz.py
    Author: Donnette Bowler
    copyright: copyright ©Donnette Bowler 2016. All rights reserved. No part of this
document may be reproduced or distributed.

============================================================================

    Use pandas dataframe to store data. We create a scatter plot with Matplotlib to
    analyze our data.



============================================================================
=

    '''



#scientific Python imports
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt

def createViz():

    input_dir="./testData/"
    filename="testData.txt"

    full_name=os.path.join(input_dir,filename)
    df=pd.read_csv(full_name,'\t')
    print df
```

```
    print type(df)
    df.columns=['freq','distance','category']


    df.plot(kind="scatter",x='freq',y='distance',c='category',s=50)
    plt.show()
```

code for CategorizeBook_v2.py

```
''' ============================================================================
====
    CategorizeBook_v2.py
    Author: Donnette Bowler
    copyright: copyright ©Donnette Bowler. All rights reserved. No part of this docum
ent may be reproduced or distributed.

============================================================================

    A command-line program that gets a text file from a user, processes the file, use
s
    a trained classifier to categorize the data, and returns the category to the user
.

    The program categorizes the manuscript as "emergent reader" or "other".



============================================================================

    '''



#import files

import CreateData
from sklearn.externals import joblib
from numpy import *
import numpy as np
```

```
#program function
def categorizerApplication():
    startCategorizer()
    answer=raw_input("would you like to classify another manuscript? Y or N: ")
    if (answer=='Y')| (answer=='y'):
        categorizerApplication()
    else:
        print "You quit the program"
        return

#function to get user input, parse file, classify the data, and return result to user
def startCategorizer():
    result_list=['emergent reader','other']
    book_filename=raw_input('manuscript file name: ')
    freq,avg_dis=CreateData.createData(book_filename)
    X=zeros((1,2))
    X[0]=[freq,avg_dis]

    #classify unknown data with our model
    clf=joblib.load('model_svm.pkl')
    result=clf.predict(X[0])


    print 'the book category is: ',result_list[(int(result[0]))-1]
```