PythonDecisionTree.md

Pythoh Code of Decision Tree

Rae YIngqi Zhang

```
import re  ## for regular expressions
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pylab as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import tree
import graphviz
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Load text data and make a word count dataframe using CountVectorizer

```
DF_dairy = pd.read_csv("pubAg/content_dairy.csv")
DF_corn = pd.read_csv("pubAg/content_corn.csv")
DF_farms = pd.read_csv("pubAg/content_farms.csv")
DF_wheat = pd.read_csv("pubAg/content_wheat.csv")
DF_organic_farms = pd.read_csv("pubAg/content_organic_farms.csv")
DF_dairy.insert(0, "Label", "Dairy")
DF_corn.insert(0, "Label", "Corn")
DF_farms.insert(0, "Label", "Farms")
DF_wheat.insert(0, "Label", "Wheat")
DF_organic_farms.insert(0, "Label", "OrganicFarms")


CSV_DF = pd.concat([DF_dairy, DF_corn, DF_farms, DF_wheat, DF_organic_farms], ignore_index=True)
labels = CSV_DF["Label"]
```

CLEAN the content

Replace punctuation with space

Accept one or more copies of punctuation plus zero or more copies of a space and replace it with a single space

```
AbstractLIST = []
for items in CSV_DF["abstract"]:
    line = str(items)
    # print(Title)
    line = re.sub(r'[,.;@#?!&$()%^*\-\']+', ' ', str(line), flags=re.IGNORECASE)
    line = re.sub(' +', ' ', str(line), flags=re.IGNORECASE)
```

```
    line = re.sub(r'\"', ' ', str(line), flags=re.IGNORECASE)
    print(line)
    line = re.sub(r'[^a-zA-Z]', " ", str(line), flags=re.VERBOSE)
    line = line.replace(',', '')
    line = ' '.join(line.split())
    line = re.sub("\n|\r", "", line)
    # remove words of a given length.....
    line = ' '.join([wd for wd in line.split() if len(wd) > 3])
    AbstractLIST.append(line)

print("The abstract list is:\n")
print(AbstractLIST)
```

Remove all words that match the topics.

```
topic = ["Dairy", "Corn", "Farms", "Wheat", "OrganicFarms"]
NewAbstractLIST = []

for element in AbstractLIST:
    print(element)
    element=str(element)
    print(type(element))
    ## make into list
    AllWords = element.split(" ")
    print(AllWords)

    ## Now remove words that are in your topics
    NewWordsList = []
    for word in AllWords:
        print(word)
        word = word.lower()
        if word in topic:
            print(word)
        else:
            NewWordsList.append(word)

    ##turn back to string
    NewWords = " ".join(NewWordsList)
    ## Place into NewHeadlineLIST
    NewAbstractLIST.append(NewWords)
```

Instantiate CV

```
MyCV = CountVectorizer(stop_words='english')
```

It has to be the code below since it encountered several errors

```
MyCV_content = MyCV.fit_transform(NewAbstractLIST)
```

Convert DTM to a DF

```
print(MyCV.vocabulary_)
vocab_dict = MyCV.vocabulary_
dict_key = vocab_dict.keys()
print("The vocab is: ", dict_key, "\n\n")
```

NEXT - Use pandas to create data frames

```
Word_count = pd.DataFrame(MyCV_content.toarray(), columns=dict_key)
Word_count_labeled = Word_count
## Add the label back to the text data
Word_count_labeled.insert(0, "Labels", labels)
# Word_count.to_csv('pubAg_5content_word_count.csv', index=False)
```

Create Training and Testing Data. Then model and test the Decision Tree

Before start modeling, let's visualize and explore.

word clouds for each of the topics.

```
List_of_WC = []
for mytopic in topic:
    tempdf = Word_count_labeled[Word_count_labeled['Labels'] == mytopic]
    print(tempdf)

    tempdf = tempdf.sum(axis=0, numeric_only=True)
    NextVarName = str("wc" + str(mytopic))
    NextVarName = WordCloud(width=1000, height=600, background_color='#fbeed1',
                            min_word_length=4,
                            max_words=10000).generate_from_frequencies(tempdf)

    ## All three wordclouds
    List_of_WC.append(NextVarName)
```

Create the wordclouds

```
fig = plt.figure(figsize=(30, 30))
# figure, axes = plt.subplots(nrows=2, ncols=2)
NumTopics = len(topic)
for i in range(NumTopics):
    print(i)
    ax = fig.add_subplot(NumTopics, 1, i + 1)
    plt.imshow(List_of_WC[i], interpolation='bilinear')
    plt.axis("off")
    plt.savefig("decisiontree/NewClouds.pdf")
```

Decision Tree

STEP 1 Create Training and Testing Data

Write the dataframe to csv

```
Word_count_labeled.to_csv("decisiontree/Word_count_labeled.csv")
TrainDF, TestDF = train_test_split(Word_count_labeled, test_size=0.3)
# print(TrainDF)
# print(TestDF)
```

STEP 2: Separate LABELS

TEST

```
TestLabels = TestDF["Labels"]
print(TestLabels)
TestDF = TestDF.drop(["Labels"], axis=1)
print(TestDF)
```

TRAIN

```
TrainLabels = TrainDF["Labels"]
print(TrainLabels)
## remove labels
TrainDF = TrainDF.drop(["Labels"], axis=1)
```

STEP 3: Run DT

1 Instantiate - entropy

```
MyDT = DecisionTreeClassifier(criterion='entropy',  ##"entropy" or "gini"
                              splitter='best',  ## or "random" or "best"
                              max_depth=None,
                              min_samples_split=2,
                              min_samples_leaf=1,
                              min_weight_fraction_leaf=0.0,
                              max_features=None,
                              random_state=None,
                              max_leaf_nodes=None,
                              min_impurity_decrease=0.0,
                              class_weight=None)

MyDT.fit(TrainDF, TrainLabels)

tree.plot_tree(MyDT)

feature_names = TrainDF.columns
Tree_Object = tree.export_graphviz(MyDT, out_file=None,
                                   ## The following creates TrainDF.columns for each
                                   ## which are the feature names.
                                   feature_names=feature_names,
                                   class_names=topic,
                                   filled=True, rounded=True,
                                   special_characters=True)

graph = graphviz.Source(Tree_Object)

graph.render("MyTree1")
```

Create a function to generate confusion matrix and heatmap

```
def get_confusionmatrix_and_heatmap(mapname, decisiontree):
    print("Prediction\n")
    DT_pred = decisiontree.predict(TestDF)
    print(DT_pred)

    bn_matrix = confusion_matrix(TestLabels, DT_pred)
```

```
    print("\nThe confusion matrix is:")
    print(bn_matrix)
    matrix_df = pd.DataFrame(bn_matrix)
    matrix_df.index = topic
    matrix_df.columns = topic
    print(matrix_df)

    sns.set(rc={'axes.facecolor':'#fbeed1', 'figure.facecolor':'#fbeed1'})
    fig = sns.heatmap(matrix_df, cmap="YlGnBu")
    plt.title(mapname, fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
    plt.xlabel("Prediction Labels", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
    plt.ylabel("Known Labels", fontname="ITC Officina Sans", fontweight="bold", color="#726abb")
    plt.xticks(fontname="ITC Officina Sans", color="#726abb")
    plt.yticks(fontname="ITC Officina Sans", color="#726abb")
    path = mapname + ".png"
    plt.savefig(path)

    FeatureImp = decisiontree.feature_importances_
    indices = np.argsort(FeatureImp)[::-1]
    ## print out the important features.....
    for f in range(TrainDF.shape[1]):
        if FeatureImp[indices[f]] > 0:
            print("%d. feature %d (%f)" % (f + 1, indices[f], FeatureImp[indices[f]]))
            print("feature name: ", feature_names[indices[f]])

get_confusionmatrix_and_heatmap('Confusion Matrix Heatmap 1 (Entropy mss=2 msl=1)',MyDT)
```

2 Instantiate - gini

```
MyDT1 = DecisionTreeClassifier(criterion='gini',  ##"entropy" or "gini"
                               splitter='best',  ## or "random" or "best"
                               max_depth=None,
                               min_samples_split=10,
                               min_samples_leaf=7,
                               min_weight_fraction_leaf=0.0,
                               max_features=None,
                               random_state=None,
                               max_leaf_nodes=None,
                               min_impurity_decrease=0.0,
                               class_weight=None)

MyDT1.fit(TrainDF, TrainLabels)
feature_names = TrainDF.columns
Tree_Object = tree.export_graphviz(MyDT1, out_file=None,
                                   ## The following creates TrainDF.columns for each
                                   ## which are the feature names.
                                   feature_names=feature_names,
                                   class_names=topic,
                                   filled=True, rounded=True,
                                   special_characters=True)

graph = graphviz.Source(Tree_Object)

graph.render("MyTree2")
```

Generate confusion matrix and heatmap

```
get_confusionmatrix_and_heatmap('Confusion Matrix Heatmap 2 (Gini mss=10 sml=7)', MyDT1)
```

3 Instantiate - gini

```
MyDT2 = DecisionTreeClassifier(criterion='gini',  ##"entropy" or "gini"
                               splitter='random',  ## or "random" or "best"
                               max_depth=None,
                               min_samples_split=15,
                               min_samples_leaf=45,
                               min_weight_fraction_leaf=0.0,
                               max_features=None,
                               random_state=None,
                               max_leaf_nodes=None,
                               min_impurity_decrease=0.0,
                               class_weight=None)

MyDT2.fit(TrainDF, TrainLabels)

tree.plot_tree(MyDT2)

feature_names = TrainDF.columns
Tree_Object = tree.export_graphviz(MyDT2, out_file=None,
                                   ## The following creates TrainDF.columns for each
                                   ## which are the feature names.
                                   feature_names=feature_names,
                                   class_names=topic,
                                   filled=True, rounded=True,
                                   special_characters=True)

graph = graphviz.Source(Tree_Object)

graph.render("MyTree3")
```

Generate confusion matrix and heatmap

```
get_confusionmatrix_and_heatmap('Confusion Matrix Heatmap 3 (Gini mss=15 sml=45 )', MyDT2)
```