Project_3_Jupyter_Notebook

May 7, 2020

The Analyzer
Jacob Duvall
train_analyzer_from_yummly.py

[5]:
```
# This code builds out models for pickle files
import json
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.svm import SVC
import pickle
import os

g_cv = ""
g_model = ""


# controls workflow of this executable
def main():
    os.chdir("C:\\Users\\jdale\\OneDrive\\School\\Text Analytics\\the_analyzer")
    df = parse_yummly()
    create_model_from_df_cuisine(df)


# opens and parses local yummly json file
def parse_yummly():
    with open('yummly.json', 'rb') as file:
        file_json = json.load(file)
        df = create_dataframe(file_json)
        file.close()
        return df


# creates a pandas dataframe from the yummly json file that contains three␣
 ↪columns: 1. id 2. cuisine 3. ingredients
def create_dataframe(file):
    id_list = list()
```

```
    cuisine_list = list()
    ingredient_list = list()
    for recipe in file:
        id_list.append(recipe['id'])
        cuisine_list.append(recipe['cuisine'])
        ing_string = ""
        for ing in recipe['ingredients']:
            ing_string = ing_string + " " + ing
        ingredient_list.append(ing_string)
    data = {'id': id_list,
            'cuisine': cuisine_list,
            'ingredients': ingredient_list}
    df = pd.DataFrame(data=data)
    return df


# not used in final implementation
def format_cuisine(cuisine_dictionary, cuisine_list):
    ingredient_dictionary = dict()
    for cuisine in cuisine_list:
        ingredient_string = ""
        ingredient_list = cuisine_dictionary[cuisine]
        for ingredient in ingredient_list:
            ingredient_string = ingredient_string + ingredient
        ingredient_dictionary[cuisine] = ingredient_string
    return ingredient_dictionary


# not used in final implementation
def label(df):
    cuisine_list = df.cuisine.unique()
    count_label = 0
    id_list = list()
    cuisine_list2 = list()
    ingredient_list = list()
    label_list = list()
    cuisine_label_dict = dict()
    for cuisine in cuisine_list:
        cuisine_label_dict[cuisine] = count_label
        count_label = count_label + 1
    for index, row in df.iterrows():
        id_list.append(row['id'])
        cuisine_list2.append(row['cuisine'])
        ingredient_list.append(row['ingredients'])
        label_list.append(cuisine_label_dict[row['cuisine']])
    data = {'id': id_list,
            'cuisine': cuisine_list2,
```

```
            'ingredients': ingredient_list,
            'label': label_list}
    df = pd.DataFrame(data=data)
    return df


# tokenizes the ingredients and creates a SVC model for the cuisine types
def create_model_from_df_cuisine(df):
    x_train = df['ingredients']
    y_train = df['cuisine']
    cv = CountVectorizer()
    x_train_cv = cv.fit_transform(x_train)
    ttt = TfidfTransformer()
    x_train_ttt = ttt.fit_transform(x_train_cv)
    clf = SVC(gamma = 'auto', probability=True).fit(x_train_ttt, y_train)

    pickle_save1 = 'pickle_cv.pkl'
    pickle.dump(cv, open(pickle_save1, 'wb'))
    g_cv = cv
    pickle_save = 'pickle_model.pkl'
    g_model = clf
    pickle.dump(clf, open(pickle_save,  'wb'))
```

[ ]:
```
main()
```

project_3.py

[2]:
```
import pickle
import pandas as pd
from train_analyzer_from_yummly import parse_yummly
import jellyfish
import sys

# provides support for all functions called within main.py


# takes argparse of ingredients and appends them all into one string that can␣
 ↪be analyzed by learning models
def ingredients_to_string(ingredients_list):
    ingredients_string = ""
    for ingredient in ingredients_list.ingredient:
        ingredients_string = ingredients_string + ingredient + ' '
    return ingredients_string


# takes ingredients string and shows the predicted cuisine and the predicted␣
 ↪value of the cuisine type
def predict_cuisine(ingredients):
```

```
    clf = pickle.load(open('pickle_model.pkl', 'rb'))
    cv = pickle.load(open('pickle_cv.pkl', 'rb'))
    values = clf.predict_proba(cv.transform([ingredients]))
    df = create_probability_dataframe(values, clf.classes_)
    cuisine_df = get_top_n_from_df(df, 1)
    cuisine_confidence = float(round(cuisine_df.value * 100, 2))
    recipe_confidence = recipe_finder(ingredients, 5)
    print('Cuisine: ', cuisine_df.cuisine.iat[0].capitalize(), ' (',␣
 ↪cuisine_confidence, '%)', sep='')
    print(recipe_confidence)


# formats value list and class list into a pandas dataframe that can more␣
 ↪easily be analyzed for top values
def create_probability_dataframe(value_list, class_list):
    v_list = list()
    c_list = list()
    data = {'value': value_list[0],
            'cuisine': class_list}
    df = pd.DataFrame(data=data)
    return df


# retrieve the top n best columns based on model score
def get_top_n_from_df(df, n):
    largest = df.nlargest(n, 'value')
    return largest


# given the input ingredients, finds the top n most similar recipes from yummly
def recipe_finder(ingredients, n):
    df = parse_yummly()
    ingredients_list = set(ingredients.split())
    match_number = 0
    id_score_dict = dict()
    for index, row in df.iterrows():
        row_i_list = set(row['ingredients'].split())
        for ingredient in ingredients_list:
            for i in row_i_list:
                if ingredient == i:
                    match_number += 1
        match_percentage = match_number / len(ingredients_list)
        match_number = 0
        id_score_dict[row['id']] = match_percentage
    final_string = "Closest " + str(n) + " recipes: "
    for key in {key: id_score_dict[key] for key in sorted(id_score_dict,␣
 ↪key=id_score_dict.get, reverse=True)[:n]}:
```

```
        value = ({key: id_score_dict[key] for key in sorted(id_score_dict,␣
↪key=id_score_dict.get, reverse=True)[:n]}[key])
        final_string = final_string + str(key) + ' ' + '(' + str(round(value *␣
↪100, 2)) + '%) '
    return final_string
```

main.py

[8]:
```
import argparse
#import project_3
from sklearn.svm import SVC



# takes arguments from argparse coming from --ingredient to process ingredients␣
↪into a list
# that can be analyzed by learning models to predict cuisine type and N closest␣
↪recipes
def main(arguments):
    ingredients = ingredients_to_string(arguments)
    predict_cuisine(ingredients)


# runs via: "py main.py --ingredient granola --ingredient rice"

parser = argparse.ArgumentParser()
# all possible arguments defined with help
parser.add_argument("--ingredient", action='append', type=str, required=True,
                    help="ingredient for cooking!")
args = parser.parse_args(["--ingredient", "duck",
                          "--ingredient", "peanuts",
                          "--ingredient", "wine",
                          "--ingredient", "bread"])
main(args)
```

```
Cuisine: French (95.19%)
Closest 5 recipes: 34488 (75.0%) 17993 (75.0%) 21198 (75.0%) 37826 (75.0%) 3641
(50.0%)
```