mohitKhanna1411 / COMP9321_20T1_UNSW

Code Issues Pull requests Actions Projects Wiki Security Insights

master

COMP9321_20T1_UNSW / Assignment_3 / z5266543.py / Jump to

mohitKhanna1411 ready to push History

1 contributor

Raw Blame

226 lines (188 sloc) 9.85 KB

```
# Author: Mohit Khanna
# Student ID: z5266543
# Platform: Mac

import os
import ast
import sys
import pandas as pd
from sklearn import metrics
from scipy.stats import pearsonr
from sklearn.ensemble import RandomForestRegressor, GradientBoostingClassifier


zid = os.path.splitext(os.path.basename(__file__))[0]

##################################################################################################
#                                                Downloaded List
holiday_month = list(['04', '05', '06', '11', '12'])
elite_list_production_companies = list(['Warner Bros', 'Sony Pictures', 'Walt Disney', 'Universal Pictures',
                                        'Pixar Animation', 'New Line Cinema', 'Twentieth Century Fox Film Cor
                                        'Paramount Pictures', 'Marvel Studios', 'Columbia Pictures'])
elite_list_cast = list(['Samuel L. Jackson', 'Harrison Ford', 'Robert Downey Jr', 'Tom Hanks', 'Morgan Freema
                        'Scarlett Johansson', 'Andy Serkis', 'Anthony Daniels', 'Tom Cruise', 'Eddie Murphy',
                        'Stanley Tucci', 'Idris Elba', 'Johnny Depp', 'Ian McKellan', 'Bradley Cooper', 'Don
                        'Vin Diesel', 'Michael Caine', 'Gary Oldman', 'Dwayne Johnson', 'Zoe Saldana', 'Stell
                        'Robin Williams', 'Woody Harrelson', 'Cate Blanchett', 'Robert DeNiro', 'Bruce Willis
                        'Emma Watson', 'Will Smith', 'Matt Damon', 'Chris Pratt', 'Chris Evans', 'Chris Hemsw
                        'Cameron Diaz', 'Liam Neeson', 'Orlando Bloom', 'Steve Carell', 'Simon Pegg',  'Mark
                        'Helena Bonham Carter', 'Mark Wahlberg', 'Owen Wilson', 'Julia Roberts', 'Ralph Fienn
                        'Carrie Fisher', 'Elizabeth Banks', 'Forest Whitaker', 'Ben Stiller', 'Adam Sandler',
elite_list_crew = list(['Gore Verbinski', 'David Yates', 'Ridley Scott', 'J.J. Abrams', 'George Lucas', 'Chri
                        'Clint Eastwood', 'Tim Burton', 'Ron Howard', 'Christopher Nolan', 'James Cameron', '
                        'Peter Jackson', 'Michael Bay', 'Steven Spielberg', 'Kevin Feige', 'Kathleen Kennedy'
                        'Jerry Bruckheimer', 'Neal H. Moritz', 'Frank Marshall', 'Charles Roven', 'Lorenzo di
                        'Ian Bryce', 'Lauren Shuler Donner', 'Avi Arad', 'Christopher Meledandri', 'Janet Hea
```

```
                        'Steven Spielberg', 'Christopher Markus', 'Stephen McFeely', 'Andrew Stanton', 'Steve
                        'David Koepp', 'Lawrence Kasdan', 'Fran Walsh', 'Terry Rossio', 'Ted Elliott'])
##################################################################################################


##################################################################################################
#                                        Data Pre-processing
def normalize_columns(data):
    if isinstance(data, list):
        res = list()
        for i in range(len(data)):
            datum = data[i]
            res.append(datum['name'])
        return ','.join(res)


def normalize_production_countries(data):
    if "United States of America" in data:
        if len(data.split(",")) == 1:
            return "USA"
        else:
            return "USA,Other_countries"
    else:
        return "Other_countries"


def normalize_spoken_languages(data):
    if "English" in data:
        if len(data.split(",")) == 1:
            return "English"
        else:
            return "English,Other_lang"
    else:
        return "Other_lang"


def normalize_keywords(data):
    length = len(data.split(','))
    if length <= 5:
        return 1
    elif length >= 6 and length <= 15:
        return 2
    elif length >= 16:
        return 3


def normalize_runtime(data):
    if data <= 60:
        return 1
    elif data >= 61 and data <= 100:
        return 2
    elif data >= 101 and data <= 140:
        return 3
    elif data >= 141 and data <= 180:
        return 4
    elif data >= 181:
```

```
        return 5


def normalize_production_companies(data):
    if any(e in data for e in elite_list_production_companies):
        if len(data.split(',')) == 1:
            return "Elite Group"
        else:
            return "Elite Group,Other_companies"
    else:
        return "Other_companies"


def normalize_gender(data):
    male = 0
    female = 0
    neutral = 0
    if isinstance(data, list):
        for i in range(len(data)):
            datum = data[i]
            if datum['gender'] == 1:
                female += 1
            elif datum['gender'] == 2:
                male += 1
            else:
                neutral += 1

    return pd.Series([male, female, neutral], index=['Male', 'Female', 'Neutral'])


def normalize_cast_crew(data, elite_list):
    ctr = 0
    for e in elite_list:
        if e in data:
            ctr += 1
    return ctr


def preprocess(dataframe):
    df = dataframe.copy()
    columns_to_normalize = ['genres', 'production_countries', 'cast', 'crew',
                            'keywords', 'production_companies', 'spoken_languages']
    for c in columns_to_normalize:
        dataframe[c] = dataframe[c].apply(lambda data: ast.literal_eval(data))
    for c in columns_to_normalize:
        df[c] = dataframe[c].apply(normalize_columns)
    df['runtime'] = df['runtime'].apply(normalize_runtime)
    df['production_countries'] = df['production_countries'].apply(
        normalize_production_countries)
    df['production_companies'] = df['production_companies'].apply(
        normalize_production_companies)
    df['n_genres'] = df['genres'].apply(
        lambda data: len(data.split(',')))
    df['spoken_languages'] = df['spoken_languages'].apply(
        normalize_spoken_languages)
    df['keywords'] = df['keywords'].apply(normalize_keywords)
```

```
    df['n_cast'] = df['cast'].apply(lambda data: len(data.split(',')))
    df['n_crew'] = df['crew'].apply(lambda data: len(data.split(',')))
    df['cast'] = df['cast'].apply(normalize_cast_crew, args=(elite_list_cast,))
    df['crew'] = df['crew'].apply(normalize_cast_crew, args=(elite_list_crew,))
    df[['Male', 'Female', 'Others']
       ] = dataframe['cast'].apply(normalize_gender)
    df['release_date'] = df['release_date'].apply(
        lambda data: 1 if data[5:7] in holiday_month else 0)
    df['homepage'] = df['homepage'].apply(
        lambda data: 0 if(data != data) else 1)
    df['tagline'] = df['tagline'].apply(
        lambda data: 0 if(data != data) else 1)
    dummies_for_columns = ['genres', 'production_countries',
                           'production_companies', 'spoken_languages']
    for c in dummies_for_columns:
        df = pd.concat([df.drop(c, axis=1), df[c].str.get_dummies(
            sep=",")], axis=1)

    return df, dataframe
#######################################################################################


if __name__ == "__main__":

    df_train = pd.read_csv(sys.argv[1])
    df_train, o_df_train = preprocess(df_train)
    df_test = pd.read_csv(sys.argv[2])
    df_test, o_df_test = preprocess(df_test)

    columns_to_drop = ['revenue', 'movie_id', 'rating',
                       'original_title', 'original_language', 'overview', 'status']

#######################################################################################
#                                     PART - 1 REGRESSION

    x_train = df_train.drop(columns_to_drop, axis=1).values
    y_train_revenue = df_train['revenue'].values
    x_test = df_test.drop(columns_to_drop, axis=1).values
    y_test_revenue = df_test['revenue'].values
    movie_ids = df_test['movie_id'].values

    rfr_model = RandomForestRegressor(random_state=0)
    rfr_model.fit(x_train, y_train_revenue)
    y_predicted_revenue = rfr_model.predict(x_test)

    msr = metrics.mean_squared_error(y_test_revenue, y_predicted_revenue)
    pcc, _ = pearsonr(y_predicted_revenue, y_test_revenue)

    pd.DataFrame({'movie_id': movie_ids, 'predicted_revenue': y_predicted_revenue}, columns=[
                 'movie_id', 'predicted_revenue']).to_csv(zid + '.PART1.output.csv', index=False)
    pd.DataFrame([[zid, round(msr, 2), round(pcc, 2)]], columns=['zid', 'MSR', 'correlation']).to_csv(
        zid + '.PART1.summary.csv', index=False)
#######################################################################################

#######################################################################################
#                                     PART - 2 CLASSIFICATION
```

```
    df_train['runtime'] = o_df_train['runtime']
    df_test['runtime'] = o_df_test['runtime']

    x_train = df_train[['runtime', 'budget', 'cast', 'crew']].values
    x_test = df_test[['runtime', 'budget', 'cast', 'crew']].values
    y_train_rating = df_train['rating'].values
    y_test_rating = df_test['rating'].values

    gbc_classifier = GradientBoostingClassifier()
    gbc_classifier.fit(x_train, y_train_rating)
    y_predicted_rating = gbc_classifier.predict(x_test)

    reports = metrics.classification_report(
        y_test_rating, y_predicted_rating, output_dict=True)
    average_precision = round(reports['macro avg']['precision'], 2)
    average_recall = round(reports['macro avg']['recall'], 2)
    accuracy = round(reports['accuracy'], 2)

    pd.DataFrame({'movie_id': movie_ids, 'predicted_rating': y_predicted_rating}, columns=[
                 'movie_id', 'predicted_rating']).to_csv(zid + '.PART2.output.csv', index=False)
    pd.DataFrame([[zid, average_precision, average_recall, accuracy]], columns=[
                 'zid', 'average_precision', 'average_recall', 'accuracy']).to_csv(zid + '.PART2.summary.csv'
##########################################################################################################
```