گزارش پروژه اول قسمت دوم

پروژه رگرسیون خطی چندمتغیره

درس: مبانی و کاربردهای هوش مصنوعی

استاد راهنما: دکتر حسین کارشناس نجف آبادی

اعضای گروه:

علی‌اکبر احراری- ۴۰۰۳۶۱۳۰۰۱

مهرآذین مزروق- ۴۰۰۳۶۱۳۰۵۵

پاییز ۱۴۰۲

فهرست

گزارش کار الگوریتم و قسمت‌های مختلف کد

df_creator

```
def df_creator():
    file_name = './Flight_Price_Dataset_Q2.csv'
    data_frame = pd.read_csv(file_name)
    encoded_df = pd.DataFrame()

    mapping = {'zero': 3,
               'one': 2,
               'two_or_more': 1}
    encoded_df['stops_mapping'] = data_frame['stops'].map(mapping)

    mapping = {'Economy': 1,
               'Business': 2}
    encoded_df['class_mapping'] = data_frame['class'].map(mapping)

    mapping = {'Morning': 'morning_departure',
               'Early_Morning': 'early_morning_departure',
               'Evening': 'evening_departure',
               'Night': 'night_departure',
               'Afternoon': 'afternoon_departure',
               'Late_Night': 'late_night_departure'}
    data_frame['departure_mapping'] =
data_frame['departure_time'].map(mapping)
    departure_dummies =
pd.get_dummies(data_frame['departure_mapping'])

    mapping = {'Morning': 'morning_arrival',
               'Early_Morning': 'early_morning_arrival',
               'Evening': 'evening_arrival',
               'Night': 'night_arrival',
               'Afternoon': 'afternoon_arrival',
               'Late_Night': 'late_night_arrival'}
    data_frame['arrival_mapping'] =
data_frame['arrival_time'].map(mapping)
    arrival_dummies = pd.get_dummies(data_frame['arrival_mapping'])

    encoded_df = pd.concat([encoded_df, departure_dummies,
arrival_dummies, data_frame['duration'],
                            data_frame['days_left'],
data_frame['price']], axis=1)
    return encoded_df
```

این تابع، ابتدا دیتافریمی از دیتاست سوال می‌سازد.

از این دیتاست، داده‌ها ستون stops را به اعداد ۱و۲و۳ مپ می‌کند.

ستون class را به اعداد ۱ و ۲ مپ می‌کند.

ستون‌های departure_time و arrival_time را نیز به روش One-Hot مپ می‌کند.

سپس این ستون‌های مپ شده، به‌علاوه‌ی ستون‌های duration و days_left به یک دیتافریم جدید تبدیل می‌شوند و به عنوان دیتافریم اصلی پروژه معرفی می‌شوند.

define_and_normalize_xs

```
def define_and_normalize_xs():
    a1 = df['stops_mapping']
    a2 = df['class_mapping']
    a3 = df['duration']
    a4 = df['days_left']
    a5 = df['afternoon_departure']
    a5 = a5.astype(int)
    a6 = df['early_morning_departure']
    a6 = a6.astype(int)
    a7 = df['evening_departure']
    a7 = a7.astype(int)
    a8 = df['late_night_departure']
    a8 = a8.astype(int)
    a9 = df['morning_departure']
    a9 = a9.astype(int)
    a10 = df['night_departure']
    a10 = a10.astype(int)
    a11 = df['afternoon_arrival']
    a11 = a11.astype(int)
    a12 = df['early_morning_arrival']
    a12 = a12.astype(int)
    a13 = df['evening_arrival']
    a13 = a13.astype(int)
    a14 = df['late_night_arrival']
    a14 = a14.astype(int)
    a15 = df['morning_arrival']
    a15 = a15.astype(int)
    a16 = df['night_arrival']
    a16 = a16.astype(int)
    # Normalize features
    a1 = (a1 - a1.mean()) / a1.std()
    a2 = (a2 - a2.mean()) / a2.std()
    a3 = (a3 - a3.mean()) / a3.std()
    a4 = (a4 - a4.mean()) / a4.std()
    a5 = (a5 - a5.mean()) / a5.std()
    a6 = (a6 - a6.mean()) / a6.std()
    a7 = (a7 - a7.mean()) / a7.std()
    a8 = (a8 - a8.mean()) / a8.std()
    a9 = (a9 - a9.mean()) / a9.std()
    a10 = (a10 - a10.mean()) / a10.std()
    a11 = (a11 - a11.mean()) / a11.std()
    a12 = (a12 - a12.mean()) / a12.std()
    a13 = (a13 - a13.mean()) / a13.std()
    a14 = (a14 - a14.mean()) / a14.std()
    a15 = (a15 - a15.mean()) / a15.std()
    a16 = (a16 - a16.mean()) / a16.std()
```

```
    return np.c_[a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12,
a13, a14, a15, a16]
```

در این تابع، متغیر های دیتاست خود را مشخص می‌کنیم. به دلیل ترتیبی بودن متغیر های اسمی Morning، Night و ... که دارای مقادیری از جنس Boolean هستند، نیاز به تبدیل آن‌ها به مقادیری عددی است. لذا از تابع astype(int) استفاده می‌کنیم.

پس از مشخص کردن متغیرها، برای اینکه متغیرها در یک مقیاس قرار گیرند و راحتتر بتوانیم روی آن‌ها پردازش انجام دهیم، روی آن‌ها عمل عادی سازی (Normalizing) را پیاده سازی می‌کنیم. این کار همچنین از به وجود آمدن خطای nan type جلوگیری می‌کند.

gradiant_descent

```
def gradiant_descent(X, Y):
    learning_rate = 0.001
    epochs = 2700
    N = Y.size
    coeff = np.random.rand(17)
    past_costs = []
    PAST_COEFF = [coeff]
    for i in range(epochs):
        prediction = np.dot(X, coeff)
        error = prediction - Y
        cost = 1 / (2 * N) * np.dot(error.T, error)
        past_costs.append(cost)
        der = (1 / N) * learning_rate * np.dot(X.T, error)
        coeff = coeff - der
        PAST_COEFF.append(coeff)
    return PAST_COEFF, past_costs
```

این تابع یک الگوریتم بهینه‌سازی به نام Gradiant descent را برای مساله رگرسیون خطی پیاده‌سازی می‌کند. این الگوریتم، با به‌روزرسانی تدریجی ضرایب، مدل رگرسیون را با هدف کمینه کردن تابع هزینه انجام می‌دهد. در هر تکرار از الگوریتم به تعداد epochs، مقادیر پیش‌بینی شده بر اساس ضرایب فعلی محاسبه شده و خطای میانگین مربعات بین پیش‌بینی‌ها و مقادیر واقعی محاسبه می‌شود. سپس با محاسبه مشتق تابع هزینه نسبت به ضرایب و استفاده از نرخ یادگیری، ضرایب به‌روزرسانی می‌شوند. این عمل تا رسیدن به تعداد تعیین شده ادامه پیدا کرده و به اتمام برسد. در آخر، تغییرات ضرایب و مقادیر تابع هزینه در هر تکرار به تحلیل و بهینه‌سازی مدل بعدی کمک می‌کنند.

generate_errors

```python
def generate_errors():
    # Predictions
    predictions = np.dot(x_test, coeffi)
    # Mean Squared Error (MSE)
    mse = np.mean((y_test - predictions) ** 2)
    # Root Mean Squared Error (RMSE)
    rmse = np.sqrt(mse)
    # R-squared (R²)
    mean_y = np.mean(y_test)
    ss_total = np.sum((y_test - mean_y) ** 2)
    ss_residual = np.sum((y_test - predictions) ** 2)
    r_squared = 1 - (ss_residual / ss_total)
    # Mean Absolute Error (MAE)
    mae = mean_absolute_error(y_test, predictions)
    return mse, rmse, mae, r_squared
```

در این تابع ، به منظور ارزیابی عملکرد مدل رگرسیون، چندین معیار ارزیابی را محاسبه می‌کنیم. این معیارها شامل میانگین خطای مربعات (MSE)، میانگین مطلق خطا (MAE)، خطای میانگین مربعات جذر شده (RMSE)، و ضریب تعیین (R-squared)هستند. در ابتدا، با استفاده از ضرایب مدل و داده‌های آزمایشی x_testو y_test، پیش‌بینی‌ها محاسبه می‌شوند. سپس، معیار MSE به عنوان میانگین از مربع اختلافات بین مقادیر پیش‌بینی شده و واقعی محاسبه می‌شود. RMSE نیز به عنوان جذر میانگین خطای مربعات به دست می‌آید و میزان میانگین مطلق خطا را اندازه‌گیری می‌کند. همچنین، ضریب تعیین R-squared به عنوان نسبت واریانس قابل پیش‌بینی متغیر وابسته از متغیرهای مستقل محاسبه می‌شود. این معیارها به صورت یک تاپلMSE، RMSE، MAE و R-squared را بازگردانده می‌شوند و ارزیابی دقیقی از کیفیت و دقت مدل رگرسیون فراهم می‌کنند.

generate_file

```
def generate_file():
    st = "PRICE = "
    for k in range(16):
        st = st + f' ({coeffi[k]}) * [{df.columns[k]}] +'

    ans = st[:-1]
    t = f'\nTraining Time: {round(end_time - start_time)}s'
    MSE, RMSE, MAE, R_SQUARED = generate_errors()
    errors = (f'\n\nLogs: '
              f'\nMSE: {MSE}'
              f'\nRMSE: {RMSE}'
              f'\nMAE: {MAE}'
              f'\nR2: {R_SQUARED}')

    file = open('[2]-UIAI4021-PR1-Q2.txt.txt', 'w')
    file.write(ans)
    file.write(t)
```

```
    file.write(errors)
    file.close()
```

این تابع، تابع خطی به‌دست آمده از الگوریتم را به صورت متن، وارد فایل می‌کند.

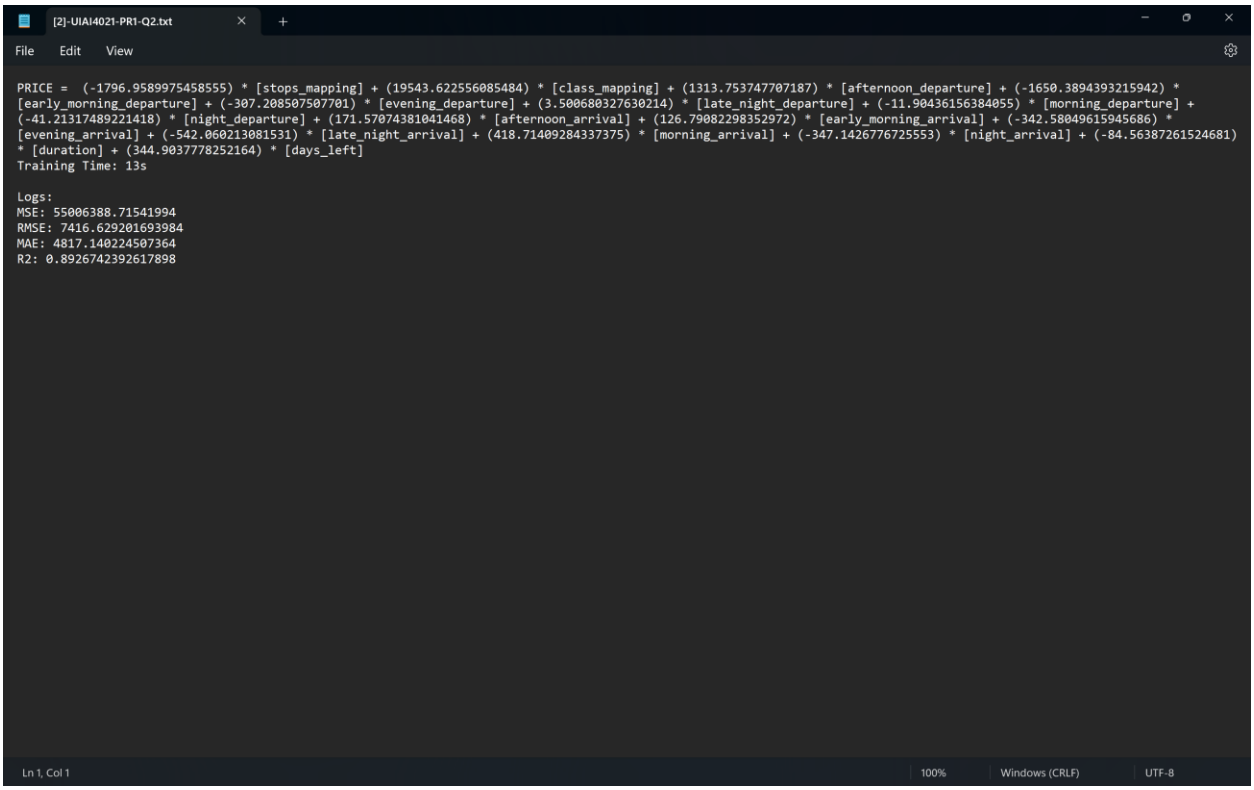
main codes

```
df = df_creator()
# Define "y"
y = df['price']
x = np.c_[define_and_normalize_xs(), np.ones(270138)]
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, shuffle=True)
start_time = time.time()
past_coeff, past_cost = gradiant_descent(x_train, y_train)
end_time = time.time()
coeffi = past_coeff[-1]

generate_file()
```

در شروع برنامه، دیتافریم اصلی پروژه ساخته می‌شود. سپس همه‌ی x ها ساخته و یک ستون که فقط حاوی ۱ می‌باشد ساخته می‌شود؛ سپس الگوریتم کاهش شیب اجرا می‌شود و در نهایت فایل مورد نیاز ساخته می‌شود.

نمونه خروجی


[2]-UIAI4021-PR1-Q2.txt
File
Edit
View
PRICE = (-1796.9589975458555) * [stops_mapping] + (19543.622556085484) * [class_mapping] + (1313.753747707187) * [afternoon_departure] + (-1650.3894393215942) * [early_morning_departure] + (-307.208507507701) * [evening_departure] + (3.500680327630214) * [late_night_departure] + (-11.90436156384055) * [morning_departure] + (-41.21317489221418) * [night_departure] + (171.57074381041468) * [afternoon_arrival] + (126.79082298352972) * [early_morning_arrival] + (-342.58049615945686) * [evening_arrival] + (-542.060213081531) * [late_night_arrival] + (418.71409284337375) * [morning_arrival] + (-347.1426776725553) * [night_arrival] + (-84.56387261524681) * [duration] + (344.9037778252164) * [days_left]
Training Time: 13s
Logs:
MSE: 55006388.71541994
RMSE: 7416.629201693984
MAE: 4817.140224507364
R2: 0.8926742392617898
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

کتابخانه‌های استفاده شده

time: برای به‌دست آوردن زمان اجرای الگوریتم کاهش شیب

pandas: برای استفاده از داده‌های سوال و تبدیل آن به دیتافریم و استفاده از داده‌های دیتافریم‌ها

numpy:

sklearn.metrics: برای به‌دست آوردن MAE

sklearn.model_selection: جداسازی داده‌ها به دو قسمت train و test

منابع

Bing AI - Search

https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent

https://youtu.be/lCOHri09YmM?si=GIPhdwRy-fxqrr_M