دانشگاه صنعتي اميركبير
(پلی‌تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

پروژه مباحثی در ریاضیات و کاربرد ها:
تجزیه QR برای تشخیص دست نوشته

نگارش

آریان فتحی

ارسلان تازیکه

گروه ۱۰

استاد درس
دکتر فاطمه شاکری

فروردین ۱۴۰۴

چکیده

در این پروژه، به بررسی و پیاده‌سازی روش‌های تجزیه QR برای تشخیص دست نوشته پرداخته شد. با استفاده از ماتریس‌های هاوس هولدر ابتدا تجزیه QR بر روی مجموعه داده‌های Letters EMNISTانجام شد. داده‌های آموزشی به‌عنوان ورودی برای تشکیل ماتریس‌های مربوط به هر کلاس انتخاب شدند و سپس با استفاده از حل مسئله کمترین مربعات، برچسب‌گذاری داده‌های تست انجام شد. در بخش دوم پروژه، تجزیه QR به‌صورت افزایشی- با افزودن داده‌های جدید و استفاده از ماتریس‌های دوران گیونز به‌روزرسانی گردید. نتایج این بخش با نتایج اولیه مقایسه شد و کارایی مدل بررسی شد.

فهرست مطالب

صفحه

فصل اول: تئوری مسئله کمترین مربعات

در این قسمت سعی می کنیم به طور خلاصه تئوری مسئله کمترین مربعات را بیان کنیم.(دقت کنیم مطالب این بخش از جزوه درس دکتر شـــاکری الهام گرفته شـــده و از مطالب آن نیز اســـتفاده می شود).

در حالت کلی مسئله کمترین مربعات به شکل زیر بیان می شود:

مسألهی کم‌ترین مربعات:

فرض کنید $A \in \mathbb{R}^{m\times n}(m > n)$ و $b \in \mathbb{R}^{m\times 1}$. می‌خواهیم X ای را بیابیم که $\|b - AX\|_2$ را مینیمم کند.

$$\min_X \|b - AX\|_2$$

در واقع هدف ما حل دسـتگاهی به شـکل AX = b اسـت که A و b در بالا تعریف شـده اند. اما می دانیم که الزاما این دســـتگاه جواب ندارد. در واقع یکی از تعبیر هایی که برای جواب داشـــتن این دسـتگاه اسـتفاده می شـود، این اسـت که بتوان بردار b را به صـورت ترکیبی خطی از سـتون های ماتریس A نوشـت. حال اگر چنین نباشـد، هدف ما یافتن برداری مانند X اسـت به شـکلی که بردار AX کمترین فاصـله را تا بردار b داشـته باشـد. در این حالت یافتن برداری مانند X معادل اسـت با جواب معادله نرمال زیر:

$$A^T b = A^T AX$$

که می دانیم این معادله حتما جواب دارد(ممکن است الزاما یکتا نباشد)

اما حل این دستگاه برای حل مسئله کمترین مربعات دارای مشکلاتی مثل از دست رفتن اطلاعات و بدوضع تر شدن مسئله نسبت به مسئله اولیه می باشد. از این رو تلاش می کنیم از ماتریس A به

ماتریسی- مانند B برسیم به طوری که فضای ستونی آنها یکسان باشد و ماتریس ضرایب مسئله نرمال نیز خوش حالت تر شود به عبارتی $B^T B = I$. حال به حل این مسئله و جستجو برای بردار مطلوب در این فضا بپردازیم. یکی از کاربرد های مسئله کمترین مربعات در تشخیص اعداد دست نوشته است که در این پروژه به آن پرداخته شده است.

فصل دوم : نکات پیاده سازی

2-1- بخش اول پروژه

در این بخش ابتدا درباره نحوه خواندن دیتا صحبت می کنیم. همان طور که می دانیم از دیتاست Emnist استفاده می کنیم. در تکه کد زیر دیتاست از Kaggle دانلود می شود و مسیر آن را ذخیره می کنیم:

```
[ ] path = kagglehub.dataset_download("crawford/emnist")

    print("Path to dataset files:", path)
```

همچنین در تکه کد های زیر، دیتافریم را از فایل دانلود شده مربوطه می خوانیم و ستون های آن را با توجه به خود دیتاست تشکیل می دهیم:

```
[ ] columns = ["label"] + [f"pixel_val_{i}" for i in range(1, 785)]

[ ] df_letters_train = pd.read_csv(f"{path}/emnist-letters-train.csv", header=None)
    df_letters_train.columns = columns
```

در خروجی زیر، می توانیم به ۵ داده(دست نوشته) اول دیتاست نگاهی بندازیم:

	label	pixel_val_1	pixel_val_2	pixel_val_3	pixel_val_4	pixel_val_5	pixel_val_6	pixel_val_7	pixel_val_8	pixel_val_9	...	pixel_val_775	pixel_val_776	pixel_val_777	pixel_val_778	pixel_val_779	pixel_val_780	pixel_val_781	pixel_val_782	pixel_val_783	pixel_val_784
0	23	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	16	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	15	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	23	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

در بخش بعدی، دیتاست را نرمالایز می کنیم:

```
# Normalizing dataset leads to better results
scaler = StandardScaler()
df_letters_train.iloc[:, 1:] = scaler.fit_transform(df_letters_train.iloc[:, 1:])
```

با نرمالسازی دیتاست، چون مقادیر تمام پیکسل ها بین ۰ و ۱ است، در نتیجه در فرایند تشکیل مجموعه آموزشی و یادگیری از آن، فرایند robust تر می شود و تاثیر پیکسل های خیلی بزرگ یا خیلی کوچک دیگر زیاد نیست. از طرفی فرایند یادگیری نیز سریع تر می شود چون محاسبات پیچیدگی کمتری خواهند داشت.

در این بخش ابتدا از هر کلاس اعداد، ۲۰۰ نمونه اولیه را به عنوان داده های اموزشی درنظر می گیریم و ماتریس آموزشی متناظر با هر کلاس را تشکیل می دهیم. سپس ۲۰ داده دیگر از هر کلاس را به عنوان ستون به ماتریس تست اضافه می کنیم و ماتریس تست حاصل از کلاس ها را می سازیم:

```
# Creating train and test matrices from letter dataset

def creating_train_and_test_matrices(train_matrices, test_matrices):
  for label in range(1, 27):
    class_samples = df_letters_train[df_letters_train['label'] == label]

    selected_train_samples = class_samples.iloc[:200, 1:].values
    selected_test_samples = class_samples.iloc[200:220, 1:].values

    train_matrices[label] = selected_train_samples.T

    test_matrices.append(selected_test_samples.T)

  test_matrices = np.concatenate(test_matrices, axis=1)

  class_1_matrix_train = train_matrices[1]

  print("Shape of class 1 (A) train matrix:", class_1_matrix_train.shape)
  print("Type of class 1 train matrix:", type(class_1_matrix_train))
  print("Shape of concatenated test matrices:", test_matrices.shape)

  return train_matrices, test_matrices
```

نکته ای که باید به آن توجه کنیم این است که داده های ما(بردار شامل مقادیر پیکسل ها) باید به عنوان بردار های ستونی در ماتریس ها ظاهر شوند نه سطرها. از این رو از عملیات ترانهاده استفاده می کنیم.

در ادامه از هر کلاس ۲۰ نمونه درنظر می گیریم و آرایه با ۵۲۰ عنصر زیر را می سازیم:

```
# Global Variables

true_labels = np.repeat(np.arange(1, 27), 20)
```

در تکه کد زیر هم به روش تجزیه QR با استفاده از ماتریس های هاوس هولدر می پردازیم. به طور خلاصـــه هم می دانیم نحوه صـــفر کردن عناصر ماتریس ورودی، ســـاختن ماتریس بالامثلثی R، همچنین ماتریس متعامد Q که از ضرب ماتریس های متعامدی به دســت می آید که وارون ماتریس های هاوس هولدر هستند که از قضا آن ها هم متعامدند:

```
def qr_decomposition_with_householder(A):
    [m, n] = A.shape
    R = A.copy()
    Q = np.identity(m)

    for i in range(min(m, n)):
        x_i = R[i:, i]
        s = np.sign(x_i[0]) if x_i[0] != 0 else 1
        normx = np.linalg.norm(x_i)
        e1 = np.zeros_like(x_i)
        e1[0] = 1
        v_i = x_i + s * normx * e1
        v_i = v_i / np.linalg.norm(v_i)  # Normalize the Householder vector

        H_i = np.identity(m)
        H_i[i:, i:] -= 2 * np.outer(v_i, v_i)

        R = np.dot(H_i, R)
        Q = np.dot(Q, H_i.T)

    return Q, R
```

در ادامه هم چک می کنیم که آیا ماتریس های R و Q بدســـت آمده، خاصـــیت های بالامثلثی و تعامد را دارند:

```
def is_upper_triangular(R):
    return np.allclose(R, np.triu(R))

def is_orthogonal(Q):
    identity = np.eye(Q.shape[0])
    QTQ = np.dot(Q.T, Q)
    return np.allclose(QTQ, identity)
```

در تابع بعدی ماتریس های Q و R ای که از تجزیه هر یک از ماتریس های آموزشی کلاس ها به دست می آیند را محاسبه می کنیم و ذخیره می کنیم:

```python
def qr_decomposition_of_all_matrices(train_matrices, Q, R):
    for i in tqdm(range(1, len(train_matrices.keys()) + 1), desc="Generating Q and R matrices for all classes of training matrices"):
        Q_i, R_i = qr_decomposition_with_householder(train_matrices[i])
        Q.append(Q_i)
        R.append(R_i)
        if not is_orthogonal(Q_i):
            print(f"Warning: Q{i} is not Orthogonal")
            print("-" * 30)
        if not is_upper_triangular(R_i):
            print(f"Warning: Q{i} is not Upper Triangle")
            print("-" * 30)

    return Q, R
```

در تکه کد زیر هم با استفاده از کمترین مربعات، برچسب نمونه های تست ماتریس تست را مشخص می کنیم:

```python
def predict_labels_using_least_square(test_matrices, Q, R):
    m, n = test_matrices.shape
    predicted_labels = np.zeros(n, dtype=int)
    for i in tqdm(range(n), desc="Predicting Labels"):
        min_score = float('inf')
        for j in range(26):
            # Solve the least squares problem using QR decomposition
            y = np.matmul(Q[j].T, test_matrices[:, i])
            x, residuals, rank, s = np.linalg.lstsq(R[j], y, rcond=1e-5)
            score = np.linalg.norm(np.matmul(R[j], x) - y, 2)
            if score < min_score:
                min_score = score
                predicted_labels[i] = j + 1

    return predicted_labels
```

به طور خلاصه برای هر نمونه آن را به فضای کلاس‌ها تصویر می کنیم.(با استفاده از ماتریس Q دست آمده از تجزیه QR ماتریس آن کلاس)

حل معادله کمترین مربعات :با استفاده از ماتریس R بهترین برآورد برای برچسب کلاس پیدا می‌شود.

محاسبه خطا :خطایی محاسبه می‌شود که نشان‌دهنده‌ی چقدر بردار بدست آمده به بردار واقعی نزدیک است. خطای کمتر به معنای تطابق بهتر است.

انتخاب برچسب بهتر :اگر خطا برای کلاسی از خطای کلاس های قبلی کمتر شد، برچسب کلاس مربوطه به عنوان پیش‌بینی برای آن نمونه انتخاب می‌شود.

در تکه کد زیر هم به طور خلاصه دقت بدست آمده در پیش بینی برچسب داده های مربوط به هر کلاس و همچنین دقت کلی(نسبت پیش بینی ها درست به کل نمونه های تست)را بدست می آوریم:

```python
def calculate_accuracy(predicted_labels, true_labels, num_classes=26, samples_per_class=20):
    correct_per_class = np.zeros(num_classes, dtype=int)
    total_per_class = np.zeros(num_classes, dtype=int)

    for i, label in enumerate(predicted_labels):
        true_label = true_labels[i]  # Get the corresponding true label
        class_index = true_label - 1  # Assuming labels are 1 to 26

        total_per_class[class_index] += 1  # Count the sample for this class
        if label == true_label:
            correct_per_class[class_index] += 1  # Count correct prediction

    accuracy_per_class = correct_per_class / total_per_class

    overall_accuracy = np.sum(correct_per_class) / np.sum(total_per_class)

    print("Accuracy per class:")
    for class_id in range(num_classes):
        print(f"Class {class_id + 1}: {accuracy_per_class[class_id] * 100:.2f}%")

    print(f"\nOverall Accuracy: {overall_accuracy * 100:.2f}%")
    return accuracy_per_class, overall_accuracy
```

در ادامه هم پایپ لاین مربوط به این قسمت را تشکیل می دهیم و آن را اجرا می کنیم. در ادامه می توانیم نتایج و دقت های بدست آمده از این روش را با هم مشاهده کنیم:

```python
def pipline_without_updating():
  train_matrices, test_matrices = creating_train_and_test_matrices({}, [])
  Q, R = qr_decomposition_of_all_matrices(train_matrices, [], [])
  predicted_labels = predict_labels_using_least_square(test_matrices, Q, R)
  accuracy_per_class, overall_accuracy = calculate_accuracy(predicted_labels, true_labels)
  return train_matrices, test_matrices, Q, R
```

2-3-1- نتایج بخش اول

```
Shape of class 1 (A) train matrix: (784, 200)
Type of class 1 train matrix: <class 'numpy.ndarray'>
Shape of concatenated test matrices: (784, 520)
100%|██████████| 26/26 [04:55<00:00, 11.36s/it]
Predicting Labels: 100%|██████████| 520/520 [05:55<00:00,  1.46it/s]Accuracy per class:
Class 1: 70.00%
Class 2: 60.00%
Class 3: 40.00%
Class 4: 60.00%
Class 5: 70.00%
Class 6: 60.00%
Class 7: 5.00%
Class 8: 45.00%
Class 9: 90.00%
Class 10: 60.00%
Class 11: 60.00%
Class 12: 40.00%
Class 13: 60.00%
Class 14: 65.00%
Class 15: 100.00%
Class 16: 75.00%
Class 17: 60.00%
Class 18: 75.00%
Class 19: 60.00%
Class 20: 70.00%
Class 21: 60.00%
Class 22: 70.00%
Class 23: 75.00%
Class 24: 70.00%
Class 25: 60.00%
Class 26: 75.00%

Overall Accuracy: 62.88%
```

2-2- بخش دوم پروژه

در این بخش تجزیه QR را به صورت افزایشی و با استفاده از ماتریس ها دوران گیونز انجام می دهیم. به طور خلاصه در تکه کد زیر پارامتر های سینوس و کسینوس ماتریس های دوران گیونز را با توجه به عنصری از بردار که از آنجا به پایین را می خواهیم صفر کنیم و عنصر بالایی آن در بردار بدست می آوریم. در صورت مشخص شدن این پارامتر ها، بقیه عناصر ماتریس های دوران مشخص هستند:

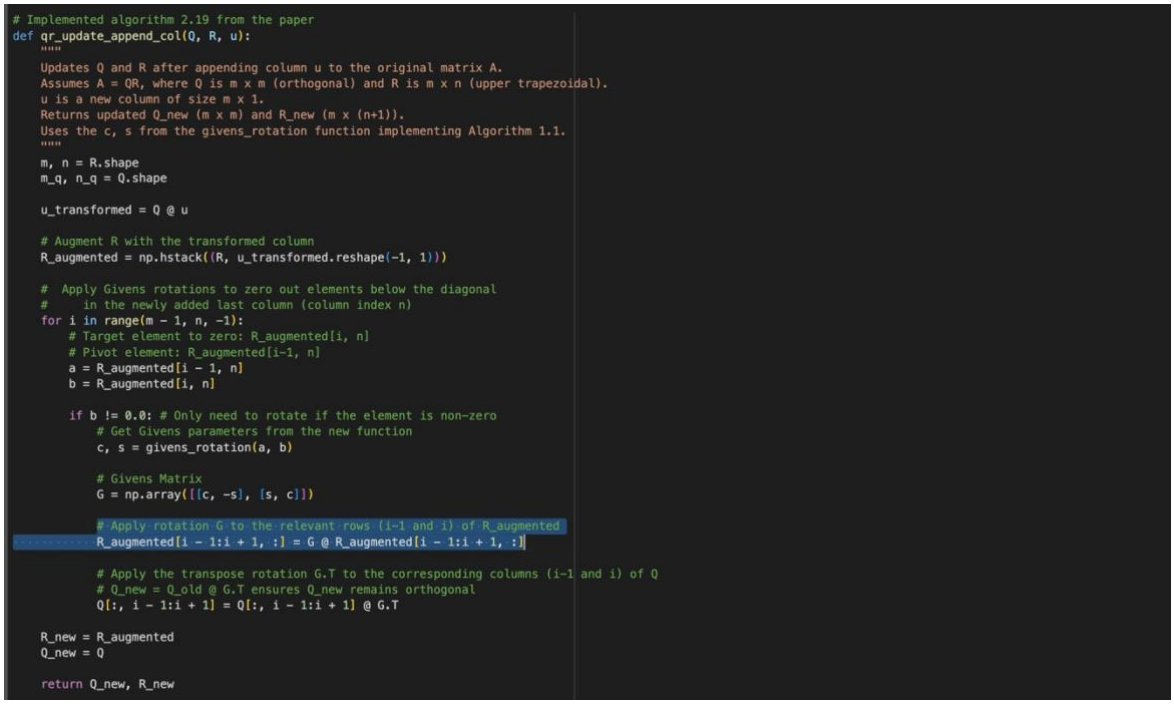
```python
def givens_rotation(a, b):
    #  Computes Givens rotation parameters c, s based on Algorithm 1.1.
    if b == 0.0:
        c = 1.0
        s = 0.0
    else:
        if abs(b) >= abs(a):
            t = -a / b
            s_denom = np.sqrt(1.0 + t**2)
            s = 1.0 / s_denom
            c = s * t
        else:
            t = -b / a
            c_denom = np.sqrt(1.0 + t**2)
            c = 1.0 / c_denom
            s = c * t

    return c, s
```

در تکه کد زیر، با توجه به الگوریتم 2.19 در فایل ضمیمه پروژه با نام Updating the QR factorization and the least squares problem متعلق به دانشگاه منچستر،داریم:

تابع qr_update_append_col برای به‌روزرسانی تجزیه QR یک ماتریس پس از افزودن یک ستون جدید u طراحی شده است. ورودی‌های این تابع شامل ماتریس متعامد Q، ماتریس بالا مثلثی R و بردار ستونی جدید u است. ابتدا، ستون جدید u با استفاده از ماتریس Q به فضای جدید تصویر می‌شود. سپس، این بردار ستونی به ماتریس R افزوده می‌شود و یک ماتریس جدید به نام R_augmented ایجاد می‌کند.

سپس با استفاده از دوران های گیونز، عناصر زیر قطر اصلی در ستون جدید صفر می‌شوند. این کار با محاسبه پارامترهای کسینویس و سینوس که در بالا توضیح دادیم انجام می‌شود. دوران معکوس نیز به ماتریس Q اعمال می‌شود تا متعامد بودن آن حفظ شود. در نهایت، تابع ماتریس‌های به‌روزرسانی‌شده Q و R را برمی‌گرداند.



```python
# Implemented algorithm 2.19 from the paper
def qr_update_append_col(Q, R, u):
    """
    Updates Q and R after appending column u to the original matrix A.
    Assumes A = QR, where Q is m x m (orthogonal) and R is m x n (upper trapezoidal).
    u is a new column of size m x 1.
    Returns updated Q_new (m x m) and R_new (m x (n+1)).
    Uses the c, s from the givens_rotation function implementing Algorithm 1.1.
    """
    m, n = R.shape
    m_q, n_q = Q.shape

    u_transformed = Q @ u

    # Augment R with the transformed column
    R_augmented = np.hstack((R, u_transformed.reshape(-1, 1)))

    #  Apply Givens rotations to zero out elements below the diagonal
    #     in the newly added last column (column index n)
    for i in range(m - 1, n, -1):
        # Target element to zero: R_augmented[i, n]
        # Pivot element: R_augmented[i-1, n]
        a = R_augmented[i - 1, n]
        b = R_augmented[i, n]

        if b != 0.0: # Only need to rotate if the element is non-zero
            # Get Givens parameters from the new function
            c, s = givens_rotation(a, b)

            # Givens Matrix
            G = np.array([[c, -s], [s, c]])

            # Apply rotation G to the relevant rows (i-1 and i) of R_augmented
            R_augmented[i - 1:i + 1, :] = G @ R_augmented[i - 1:i + 1, :]

            # Apply the transpose rotation G.T to the corresponding columns (i-1 and i) of Q
            # Q_new = Q_old @ G.T ensures Q_new remains orthogonal
            Q[:, i - 1:i + 1] = Q[:, i - 1:i + 1] @ G.T

    R_new = R_augmented
    Q_new = Q

    return Q_new, R_new
```

qr_decomposition_updating_of_all_matricesبرای به‌روزرسانی تجزیه QR برای هر کلاس از داده‌های آموزشی طراحی شده است. این تابع به‌طور خاص برای افزودن ۲۰ نمونه جدید به هر کلاس از داده‌ها استفاده می‌شود.

ورودی‌های تابع شامل ماتریس‌های Qو R، مجموعه داده‌های آموزشی ، اندازه اولیه آموزش initial_train_sizeو تعداد نمونه‌هایی که باید اضافه شوند هستند.

سپس، با استفاده از یک حلقه، به‌روزرسانی برای هر یک از کلاس های ۱ تا ۲۶ انجام می‌شود. در هر تکرار، اندیس کلاس به‌دست می‌آید و نمونه‌های مربوط به آن کلاس از مجموعه داده استخراج می‌شود. اندیس‌های شروع و پایان برای انتخاب نمونه‌های جدید تعیین می‌شوند.

سپس، مقادیر پیکسل های نمونه‌های جدید به‌دست می‌آید(ماتریس شامل داده های جدید) و برای هر نمونه جدید، ستونی به نام new_col_u ساخته می‌شود. این ستون به‌عنوان ورودی به تابع **qr_update_append_col** ارسال می‌شود تا Q و Rبه‌روزرسانی شوند. بدین ترتیب ماتریس های Q و R مربوط به هر کلاس با اضافه شدن داده های جدید به هر کلاس به روزرسانی می شوند.

پس از به‌روزرسانی، دو بررسی انجام می‌شود: اول اینکه آیا Q متعامد باقی مانده است یا خیر و دوم اینکه آیا Rهمچنان بالا مثلثی است یا خیر.

در نهایت، تابع ماتریس‌های به‌روزرسانی‌شده Q و Rرا برمی‌گرداند.

در تکه کد زیر پایپ لاین این روش و نتایج حاصله از اضافه شدن ۲۰ داده جدید به هر کلاس را می توانیم مشاهده کنیم:

```
[ ]  def pipline_without_updating():
       train_matrices, test_matrices = creating_train_and_test_matrices({}, [])
       Q, R = qr_decomposition_of_all_matrices(train_matrices, [], [])
       predicted_labels = predict_labels_using_least_square(test_matrices, Q, R)
       accuracy_per_class, overall_accuracy = calculate_accuracy(predicted_labels, true_labels)
       return train_matrices, test_matrices, Q, R
```

2-3-2- نتایج بخش دوم

```
100%|██████████| 26/26 [00:40<00:00,  1.58s/it]
Predicting Labels: 100%|██████████| 520/520 [06:34<00:00,  1.32it/s]Accuracy per class:
Class 1: 70.00%
Class 2: 60.00%
Class 3: 40.00%
Class 4: 60.00%
Class 5: 70.00%
Class 6: 55.00%
Class 7: 5.00%
Class 8: 45.00%
Class 9: 90.00%
Class 10: 60.00%
Class 11: 60.00%
Class 12: 45.00%
Class 13: 60.00%
Class 14: 60.00%
Class 15: 100.00%
Class 16: 75.00%
Class 17: 60.00%
Class 18: 75.00%
Class 19: 65.00%
Class 20: 70.00%
Class 21: 60.00%
Class 22: 75.00%
Class 23: 75.00%
Class 24: 70.00%
Class 25: 60.00%
Class 26: 80.00%

Overall Accuracy: 63.27%
```

همان طور که مشاهده می کنیم با اینکه داده بیش تری برای آموزش داشتیم، دقت کلی افت کرده است. در کلاس هایی که دقت تغییر کرده است(مثل کلاس ۲۶، کلاس ۱۹ و ...) کاهش دقت را داشته ایم. این مسئله می تواند دلایل متفاوتی داشته باشد. مثلا داده های جدید دارای نویز باشند

و فضـای سـتونی حاصـل از ماتریس آموزش و معادلا فضـای سـتونی ماتریس متعامد Q حاصـل از تجزیه آن، به خوبی فضای مربوطه به آن کلاس را نسازد و فضای ستونی خیلی تغییر کند با اضافه کردن داده های جدید. یا ممکن است داده های پرت باشند و خیلی فضا را تغییر دهند.

نتیجه گیری

در این پروژه ما از تجزیه QR برای حل مسئله کمترین مربعات استفاده کردیم و دیدیم که این روش نسبتا ساده از لحاظ پیاده سازی می تواند دقت نسبتا خوبی روی دیتاست Emnist به ما بدهد. در ادامه و با پیاده سـازی بخش دوم، متوجه شـدیم که داشـتن داده بیش تر و اضـافه کردن داده به دیتاسـت اولیه الزاما موجب بهبود عملکرد مدل پیش بینی کننده نمی شـود. عواملی مانند نویز در داده های جدید، پرت بودن نسـبت فضـای سـتونی ماتریس آموزش اولیه و ... می توانند موجب تغییر زیاد فضـای جدید و در نتیجه تغییر در دقت کلی مدل شـود(فضـای سـتونی ماتریس اولیه که تغییر کند، فضـای سـتونی ماتریس متعامدQ حاصـله از تجزیه QR نیز تغییر می کند چون فضـای سـتونی آنها طبق قضـیه ای در جزوه یکسـان اسـت). کما اینکه در این پروژه با کاهش دقت مواجه بودیم. این مورد در دیگر الگوریتم های یادگیری ماشـین نیز مشـاهده می شـود و بیش تر کردن داده آموزشی اگر از کیفیت و کمیت لادم برخوردار نباشـد، می تواند موجب اشـتباه در فرایند یادگیری شود و دقت مدل ما، کاهش پیدا کند.

لینک کد پروژه

https://colab.research.google.com/drive/1NbjfiCuuKaap_2SX8VaHCsbFFGHdKPwY - scrollTo=dX5XYj29hkas

منابع

https://eprints.maths.manchester.ac.uk/1192/1/qrupdating_12nov08.pdf

جزوه درس کاربرد های جبر خطی در علم داده دکتر فاطمه شاکری

https://iasbs.ac.ir/~narimani/rsrc/cdm/Matrix_methods_in_data_mining.pdf