Term-Project RSLDA Report

프로젝트 목표

Linear Discriminant Analysis(LDA)에 대한 이해를 바탕으로 장점과 단점을 분석하여,
그 단점을 극복할 수 있는 새로운 LDA를 연구하고 Python으로 구현한다.

프로젝트 동기 : 새로운 LDA를 연구하기 전 우선적으로 기존 LDA의 단점을 알아보기로 했다.
기존 LDA는 노이즈와 데이터 분포에 대해 민감하며, 고차원 데이터에서의 분류를 효율적으로 이뤄내지 못한다 등의 단점들을 찾았고, 우리 팀은 앞서 나열한 단점 개선에 주목하기로 했다.

프로젝트 진행과정

1. 데이터 선정

고차원 데이터들은 여러가지 있지만, 비교적 구하기 쉽고 범용성이 높은 이미지 데이터로 선정하였다.
Kaggle을 통해 여러가지 데이터를 살펴보던 중 용량이 방대하지 않고 다른 이미지 분류에서 흔히 쓰이는 데이터셋이 아닌 Pizza_Not_Pizza 이미지 데이터셋을 발견해 프로젝트를 위한 데이터셋으로 선정하였다.

Not_pizza 데이터 중 3042363번의 사진.

Pizza 데이터 중 54540번의 사진.

---

2. 데이터 전처리

이미지 파일을 LDA의 입력 데이터로 사용시 적절한 전처리 과정이 필요했다.
전처리를 위해
`OpenCV` 라이브러리를 사용하였고, 해당 라이브러리는 이미지 크기 조정, 차원 축소,
이미지 평탄화 등의 작업을 지원했다. 따라서 Pizza_Not_Pizza 데이터들의 이미지 크기를 추후 언급할 RSLDA 논문의 예시와 같이 32X32의 크리고 변환한 후, numpy 배열로 변환하였다.
마지막으로
`sklearn.preprocessing`의 `StandardScalar()` 함수로 정규화를 진행했다.
그 후 피자와 피자가 아닌 것을 구분하기 위해 리스트를 통한 라벨링을 진행하였다.
(피자인 것은 1, 피자가 아닌 것은 0)

---

3. 기존 LDA 검증

우리는 기존 LDA의 단점을 파악하고 해당 단점을 타겟으로 데이터셋도 선정하고 전처리까지 진행하였지만,
과연 기존 LDA가 어느정도로 효율이 안좋게 나올지,
과연 새로운 LDA를 통해 큰 폭으로 정확성을 올릴 수 있을지 직접 확인하는 작업이 필요하다 생각했다.
따라서 기존 LDA를 사용해 전처리한 데이터셋을 입력 데이터로 분류를 시켜보았다.
test_size를 조금씩 바꾸면서 측정해보니 평균적으로 60%초반의 정확성을 보여주었다.
높은 수치가 아니라고 보여져 새로운 LDA를 통해 유의미한 개선이 가능할 것이라 판단하였다.

---

4. 논문 분석

우리는 무작정 LDA를 직접 건드려 보는 것 보다 일단 주어진 논문을 읽어보는 것이 더 효율적이라 생각했다.
여러 논문을 보던 중
Robust Sparse Linear Discriminant Analysis, IEEE Transactions on Circuits and Systems for Video Technology, 2018. 논문이 우리 팀이 집중하는 고차원 데이터 분류에 적합한 내용인 것 같아 해당 논문을 본격적으로 분석하기로 하였다.

RSLDA는 일반 LDA에 비해 노이즈에 대해 강인한 특징을 가지고, 특징선택을 통합하여 중요한 특징만을 사용해 고차원 데이터의 계산을 용이하게 하여 유연한 차원 선택이 가능하다는 장점이 있었다.
팀의 배경지식 수준이 얕기 때문에 논문을 완벽하게 이해하지는 못했지만, 지금까지의 기계학습 수업을 리뷰하고, 여러 자료들을 찾아보며 나름의 방식으로 방향을 잡아갔다.

우선적으로 RSLDA 논문에 등장하는 핵심 용어들 중 팀에게 낯선 용어들 위주로 학습를 진행했다.
PCA, SVD, 산포행렬, 희소성 제약, 소프트 임계값(자료부족으로 정의정도만), 투영, 잡음 행렬,
l_2 노름, 라그랑주 승수법, ADMM 알고리즘 등을 학습하였다.

노션을 활용해 학습한 것을 공유하며 진행하였다.

그 후 RSLDA가 일반 LDA에 비해 가지는 장점이 구체적으로 어떤 식으로 구현되었는지 파악하였다.
먼저 특징 해석에 있어서 RSLDA는 투영 행렬에 노름 제약 조건을 추가하여 투영 행렬의 행들이 “희소성”을 가지게 되어 결과적으로 중요한 특징만이 남게끔 하는 방식을 사용한다는 것을 알게되었다.
그 다음 노이즈에 대한 강점에서는 학습 과정에서 “희소 오차 항”이라는 것을 포함하여 더 robust한 특징들을 추출하는 방법을 사용하였고, 마지막 차원 선택에 유연성에 있어서는 PCA를 이용해 투영된 저차원 공간에서 원본 데이터의 주요 정보를 유지할 수 있도록 하는 방법을 사용한다는 것을 파악하였다.

---

5. RSLDA 구현

논문에 있는 것처럼 완벽한 RSLDA를 구현하는 것을 목표로, 분석을 통해 catch한 부분들에 집중하여 구현하는 방향으로 진행해나갔다.

LDA의 구현

앞서 RSLDA를 구현하기 전에, 일반 LDA를 먼저 구현해야 했다.
RSLDA의 산출물은 정확도 그 자체가 아니라 학습된 변환 행렬, 투영 행렬, 잡음 행렬, 목적 함수 값 등의
LDA의 입력을 전처리해서 전해주는 식이므로 LDA 자체가 필요했다.
구현 자체는 특이한 것이 없이 일반적인 LDA를 구현했다.

SVD, PCA, ScatterMat의 구현

RSLDA의 실행에 필요한 SVD, PCA, ScatterMat들을 구현했다.

- SVD : 논문의 SVD는 일반 SVD와 근본적으로 다르지는 않지만,
  계산 효율과 안정성을 위한 최적화가 포함되어 있었다.
  - 행렬 크기에 따른 XX^T 또는 X^TX의 교유값 분해의 선택
  - 희소 행렬에 따른 추가 처리
  - 작은 고유값 필터링
  - 원하는 축소 차원의 설정

  등의 차이점이 있다.
- PCA : 논문의 PCA 또한 일반 PCA와 비슷하지만, 아래와 같은 차이점이 있다.
  - 데이터 행렬에 대해 SVD를 수행해 고유벡터와 고유값을 구함
  - 희소 행렬의 처리
  - 차원 축소 옵션 사용
- ScatterMat : 산포 행렬을 계산하는 함수이다. 특이한 점은 없다.

RSLDA의 구현

논문의 RSLDA를 구현했다. 흐름은 아래의 단계들로 나뉜다.

1. 초기 설정 및 초기화
   a. 입력 데이터와 파라미터 받아옴
   b. 데이터 행렬 크기 계산 후 `max_mu`설정, 정규화 상수 `regu` 설정
   c. 산포 행렬 `Sw`, `Sb` 계산
   d. PCA를 활용해 초기 변환 행렬 `P1` 계산
   e. 투영, 잡음, 라그랑지 승수 행렬 각각 초기화
   f. 투영 행렬의 행 벡터들 $l2-$노름을 계산해 대각행렬 `D`를 생성
2. 반복 작업
   a. 변환 행렬 `P` 업데이트
   초기 PCA 변환 행렬 이후 반복은 SVD 사용
   b. 투영 행렬 `Q` 업데이트
   이 때,
   `Q1`은 클래스 내 산포와 클래스 간 산포를 포함한 행렬이며,
   `Q2`는 현재 잡음 행렬 `E`와 라그랑지 승수 `Y`를 포함한 행렬이다.
   c. 잡음 행렬 `E` 업데이트
   현재 변환 행렬
   `P`와 투영 행렬 `Q`를 사용하여 계산된 값이다.
   d. 라그랑지 승수 `Y` 와 파라미터 `mu` 업데이트
   제약 조건 위반 정도를 계산하고,
   `Y`와 `mu`를 업데이트한다.
   e. 목적 함수 값 계산 및 저장
   f. 조기 중단 조건 확인
3. 최종 반환
   학습된 변환 행렬
   `P`, 투영 행렬 `Q`, 잡음 행렬 `E`, 반복 과정에서 계산된 목적 함수 값 목록 `obj`를 반환

---

6. RSLDA의 성능체크 및 분석

구현된 RSLDA를 바탕으로 Pizza_not_Pizza 데이터셋을 분류하였다.
하이퍼파라미터들은 논문에서 사용했던 값들을 기본적으로 사용하였다.

```
Testing with params: lambda1=1e-05, lambda2=0.0001, dim=50, mu=0.01, rho=1.01
RUNNING RSLDA (Serial)!
Accuracy: 0.565989847715736
```

결과가 0.58이 나왔다.
이 수치는 일반 LDA의 성능에도 미치지 못하는 성능이었다.
논문에서는 91퍼센트의 정확도까지 나온 케이스도 존재했기에 뭔가 문제가 있는 것이 분명했다.
Pizza 데이터와 논문의 YaleB 안면 인식 데이터의 차이가 있기에 하이퍼파라미터 튜닝이 필요하다고 판단했다.

파라미터 튜닝

처음에는 현재 데이터 차원이 32,32인데, 이것을 64,64로 늘려서 실험해보았더니
4분 → 1시간 가량으로 학습 코스트가 대폭 늘게 되었고, 성능도 큰 차이가 없음을 확인했다.

논문의 YaleB 얼굴 데이터셋과 피자 구분 음식 사진 데이터셋의 특징이 전혀 다른 것이 저조한 성능의 이유라고 생각하고, 하이퍼 파라미터 튜닝을 시작했다.
현실적으로 고려해볼만한 하이퍼 파라미터 리스트는 아래와 같았다.

```
# 하이퍼파라미터 설정
lambda1_values = [0.00001, 0.0001, 0.001]
lambda2_values = [0.0001, 0.001, 0.01]
dim_values = [50, 100, 150]
mu_values = [0.01, 0.1, 1]
rho_values = [1.01, 1.05, 1.1]
```

현재 하드웨어 환경 지원 상 모든 조합에 대한 성능을 측정하려면,
243개의 조합 * 4분 정도의 시간이 걸리므로 측정에만 16시간이 넘게 걸리게 될 것으로 예상했다.

이를 더 효율적으로 측정하기 위해 아래 방법들을 고려했다.

1. GPU의 사용으로 계산 가속화 - 실패
   현재 하드웨어의 GPU가 산포 행렬 P 자체의 메모리 할당에 실패할 정도로 열악하여 포기했다.
2. 조기 중단
   성능이 최대 5번 이상 개선되지 않을 경우 가망이 없는 것으로 판단해 학습을 중단시켰다.
3. 랜덤 선택
   243개의 조합 중 임의로 20개의 조합을 뽑아 평가했다.
   모든 조합을 해보지 못하는 것은 아쉬운 일이지만 현재의 리소스 한계상 어쩔 수 없다고 판단했다.

```
INFO - Testing with params: lambda1=1e-05, lambda2=0.0001, dim=50, mu=0.01, rho=1.01
INFO - Mean Accuracy: 0.5589994962607046
INFO - Testing with params: lambda1=0.001, lambda2=0.0001, dim=50, mu=1, rho=1.05
INFO - Mean Accuracy: 0.672442877255525
INFO - Testing with params: lambda1=0.0001, lambda2=0.001, dim=50, mu=0.1, rho=1.01
INFO - Mean Accuracy: 0.6129241420286485
INFO - Testing with params: lambda1=0.0001, lambda2=0.01, dim=50, mu=0.1, rho=1.1
INFO - Mean Accuracy: 0.6459978558788959
INFO - Testing with params: lambda1=1e-05, lambda2=0.01, dim=150, mu=1, rho=1.1
INFO - Mean Accuracy: 0.6576833158962039
INFO - Testing with params: lambda1=1e-05, lambda2=0.0001, dim=100, mu=1, rho=1.05
INFO - Mean Accuracy: 0.6683625889616513
INFO - Testing with params: lambda1=0.001, lambda2=0.0001, dim=50, mu=0.01, rho=1.05
INFO - Mean Accuracy: 0.5651102414073701
INFO - Early stopping triggered after 5 rounds with no improvement.
INFO - Best accuracy: 0.672442877255525
INFO - Best parameters: {'lambda1': 0.001, 'lambda2': 0.0001, 'dim': 50, 'mu': 1, 'rho': 1.05}
```

5-fold CV를 거친 하이퍼파라미터 교차검증 결과

다양한 파라미터를 거친 결과 0.67의 성능을 보이는 최적의 파라미터를 선정했다.

회고

아쉬웠던 점 : 기간이 많이 주어진 프로젝트였지만, 프로젝트의 난이도는 쉽지 않았다.
조금 더 깊은 지식과 많은 시간이 있었다면, 현 프로젝트에서 진행한 RSLDA의 아이디어 뿐만 아니라
다른 개선된 LDA의 장점들도 분석하고 팀원 각자의 생각을 더해 더 나은 성능의 모델을 만들 수도 있었을 것 같은데 그 부분이 조금 아쉬웠다.
또한 리소스의 부족으로 모든 하이퍼파라미터 조합에 대한 성능을 검증하지 못한 것이 아쉽다.

좋았던 점 : 실제 논문을 분석해 본 것과, 인공지능 모델링을 처음부터 끝까지 진행해본 첫번째 경험이었다.
예상보다 더 험난한 과정이었지만, 중간중간 학교에서 지금까지 배운 것들이 조금씩 연결되는 경험도 해보며
얻어가는 것도 많았던 프로젝트였다.

기여도

코드 작성 - 1: 이대진 2: 오수만

레포트 작성 - 1: 오수만 2: 이대진