Machine Learning Assignment

이해민

2021312394
인공지능융합전공

1. Problem Definition

:데이터셋은 네트워크 트래픽에 대한 정보를 담고 있는 csv파일로, interactive, bulk, web, video라는 4개의 class로 구분된다. 주어진 test.csv를 직접 구현한 모델과 데이터 전처리를 통해 4개의 class로 분류해내는 것이 이 프로젝트의 목표이다.

2. Data

2.1. Data Description

(1) 형태:

주어진 데이터는 각 class의 이름으로 시작되는 84개의 csv파일로, time, proto.data_len, ip_src, ip_dst, src_port라는 7개의 column으로 이루어져 있다. Time은 strtime형태로 저장되어 있으며. 각 ip주소 및 data_len이라는 int 데이터를 값으로 가지고 있다.

(2) 피처:

column의 개수는 7개로 동일하며, 각 csv파일 별로 약 3만개 정도의 열이 존재한다.

(3) 클래스의 개수:

interactive(0), bulk(1), web(2), video(3)라는 4개의 class가 존재한다.

2.2. Data Pre-processing

(1) 데이터 읽어오기

: 각 csv 파일을 데이터 프레임 형식으로 불러온 후 리스트에 할당한다.

(2) time 형식 변경

: strtime 형식으로 이루어진 time 컬럼을 %Y/%m/%d %H:%M:%S 형태로 변경한다.

이때, pd.to_datetime으로 datetime object로 바꿔준 후 strftime을 통해 원하는. 형태로. 변경할 수 있다.

(3) data_len 컬럼 개수 구하기 + ip_dst가 40개 이하인 ip 데이터 제거

: time과 ip_dst 컬럼으로 groupby하여 해당 일시에 대한 data_len 컬럼의 개수를 구한다. 이때, ip_dst의 데이터 개수가 40개를 초과한 경우만 남겨두기 위해 다시 ip_dst로 groupby를 진행한 후, filter함수를 통해 40개 이하의 행은 제거하였다. 이렇게 3번까지의 전처리를 진행한 데이터프레임을 df_gb라는 새로운 리스트에 할당하였다.

(4) windowing

: 먼저, t_columns라는 column명을 리스트로 정의해주었다. 이후, 이 콜럼명을 콜럼으로 가지는 class 별 데이터프레임을 생성하였다. 먼저 (3)에서 생성된 df_gb에서 각 데이터프레임을 하나씩 불러와 ip_dst별로 각 data_len을 groupby하였다. 이후 그 데이터프레임의 행의 개수만큼 for문을 반복하며 각 행을 리스트로 만들었다. 그 후, len(해당 리스트) - widow_size + 1 만큼 반복하여 총 11개의 int 데이터를 중첩하여 추출하였다. 이렇게 추출된 데이터를 동일한 column

을 가진 데이터프레임으로 변경해준 후, pd.concat을 통해 해당 class의 dataframe에 하나의 행으로 추가해주었다. 이때, index를 0부터 초기화하기 위해 ignore_index=True로 설정하였다.

(5) 라벨링

: 각 class별로 데이터프레임이 생성되어 있으므로 label이라는 column을 새로 정의하여. label을 넣어줬다. -> interactive(0), bulk(1), web(2), video(3). 이때, video 데이터프레임의 shape은(908,12(라벨 추가))로 example_video.csv와 같은 shape을 가진다. 마지막으로 각 label별 데이터프레임을 concat한 후 (ignor_index=True) 하나의 csv 파일(train_none_index.csv)로 저장했다.

3. Analysis

train_none_index.csv

```
traffic(t-10),traffic(t-9),traffic(t-8),traffic(t-7),traffic(t-6),traffic(t-5),traffic(t-4),traffic(t-3),traffic(t-2),traffic(t-
1,18,3,4,10,3,7,4,20,7,8,0
18,3,4,10,3,7,4,20,7,8,14,0
3,4,10,3,7,4,20,7,8,14,14,0
4,10,3,7,4,20,7,8,14,14,14,0
10,3,7,4,20,7,8,14,14,14,7,0
3,7,4,20,7,8,14,14,14,7,1,0
7,4,20,7,8,14,14,14,7,1,7,0
4,20,7,8,14,14,14,7,1,7,20,0
20,7,8,14,14,14,7,1,7,20,13,0
7,8,14,14,14,7,1,7,20,13,8,0
8,14,14,14,7,1,7,20,13,8,7,0
14,14,14,7,1,7,20,13,8,7,7,0
14,14,7,1,7,20,13,8,7,7,13,0
```

Figure 1. 최종 train csv 파일

해당 데이터는 네트워크 트래픽 데이터로 시계열 데이터이다. 즉, 각 class별로 특정한 패턴이 존재하므로 전체 데이터셋을 분석하는 것보다는 일정한 길이 별로 분리하여 분석하는 것이 효과적이므로 11개씩 windowing을 진행한 데이터이다.

4. Model

4.1. Classifier

: 문제를 해결하기 위해 기존에 과제로 구현했던 SVM 모델을 사용하였다. SVM classifier는 한 클래스의 데이터 점을 다른 클래스의 데이터 점과 가능한 한 가장 잘 구분해내는 초평면을 찾는 것이다. 즉 데이터를 잘 분리하며, margin을 최대화하는 구분선을 찾아내는 모델이다.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Loss function for SVM

Figure 2. SVM의 loss function[5]

SVM 모델을 구현할 때는 위의 loss function 수식을 통해, misclassification의 유무에 따라, gradient값을 업데이트하며 학습을 진행하였다.

구현에는 실패했으나 SVM에 가우시안, 선형, 다항식 커널을 적용하면 특징을 변환하고 비선형 문제를 처리할 수 있으므로, 모델의 정확도가 더욱 향상될 수 있다. 과제로 구현한 SVM은 이진 분류만 가능하므로, multiclass 분류를 위해서 4.2(2)에 언급할 기법들이 필요하다.

4.2. Design Consideration

(1) PCA

: PCA란 주성분 분석 기법으로 가장 널리 사용되는 차원 축소 기법이다. 원 데이

터의 분포를 보존하며 차원을 낮춰준다. PCA가 모델의 성능을 확실히 높여주는 것은 아니지만, 핵심 feature들(설명력이 높은 feature들)만 선택하여 모델을 만드는 것에는 유용하기에 이 기법을 적용해보았다. 최종으로 선택된 모델에서는 총 10개의 차원을 7개의 차원으로 감소시켜 학습했는데, 이 과정에서 미세한 성능의 향상 (약 0,005)이 있었다. 하지만, 이는 유의미하다고 보기에는 작은 수치였다.

(2) Multiclass 분류

:기존에 구현했던 SVM 모델은 이진분류만 가능하기에 이를 multiclass에 적용하기 위한 기법이 필요했다. 그 기법에는 One vs Rest와 One vs One 기법이 있다. 그 중 나는 one vs rest 기법을 선택하였다. 이 기법은 먼저 pd.get_dummies를 통해 해당 인덱스의 데이터에는 1, 아닌 데이터에는 0을 배정하여 one-hot encoding된 y값을 생성한다. 이후, 전체 class 만큼의 분류기를 생성한다. 그리고 각 분류기는 해당 class에 속한다고 판단되는 정도인 decision function값을 반환한다. 다음으로, 이를 리스트에 넣고 np.argmax를 통해 그 값이 가장 큰 인덱스를 반환하는 방법으로 이 기법을 구현하였다.

5. Experiments

5.1. Settings

(1) 파라미터 선정 방식

```
acc=[]

# SVM 모델 객체 생성
svm = SVMClassifier()

# 탐색할 하이퍼파라미터 정의
param = {
    'n_iters': [50, 100, 200], # 몇 회 반복하여 적절한 값을 찾을지 정하는 파라미터
    'lr': [0.0001, 0.001, 0.01, 0.05], # 학습률과 관련된 파라미터
    'lambda_param': [0.01, 0.1, 1] # 람다 파라미터
}


for i in param['n_iters']:
    for j in param['lr']:
        for k in param['lambda_param']:
            temp = [i,j,k]
            onevsrest = OneVsRest(n_iters=i,lr=j,lambda_param=k)
            onevsrest.fit(X_train, y_train)
            y_pred = onevsrest.predict(X_test)
            acc.append(f'{temp}={onevsrest.get_accuracy(y_test,y_pred)}')
```

Figure 3. 파라미터 테스트 코드

:test size = 0.33으로 고정한 후, 위와 같은 코드를 통해 여러 개의 파라미터들을 적용해가며 총 3*4*3 = 36개의 경우의 수를 실험하였다. 그 결과, [n_iters=200, lr= 0.0001, lambda_param=1]일때, Accuracy: 0.85790로 가장 높은 정확도를 보였다. 이 파라미터들을 최종으로 선택하였다.

(2) Train Test 비율

: [0.1, 0.33, 0.4]라는 3가지의 값으로 테스트를 진행하였다. 이때, 파라미터는 , [n_iters=200, lr= 0.0001, lambda_param=1]로 고정하였다. Test set의 비율을 낮게 하였을 때, 성능이 더 높을 것이라고 예측했지만, 각각의 accuracy는 [0.77154, 0.85790, 0.63280]으로 test_size= 0.33일때 가장 좋은 정확도를 보였다. Test set의 비율을 낮게 했을 때, 정확도가 낮아진 이유는 train dataset에 지나치게 fit된 것이라고 판단했다.

(3) 학습 프로세스

1. Train dataset을 train과 test 데이터셋으로 분리
2. PCA기법을 통해 데이터를 7차원으로 축소
3. One vs Rest 기법으로 각각의 class에 대한 classifier 학습
4. Decision function값의 비교를 통해 최종 해당 class 선정
5. 정확도 계산

5.2. Performance Metrics

Train accuracy를 통해 해당 모델의 성능을 측정하였다. Train accuracy를 계산하기 위해서는 과제2에서 작성한 get_accuracy 코드를 활용하였다. 아래 그림은 파라미터에 따른 성능 비교를 진행한 결과 중 일부이다. 최종 선택한 파라미터가 정확도 85.79% 가장 높은 정확도를 보이는 것을 알 수 있다.

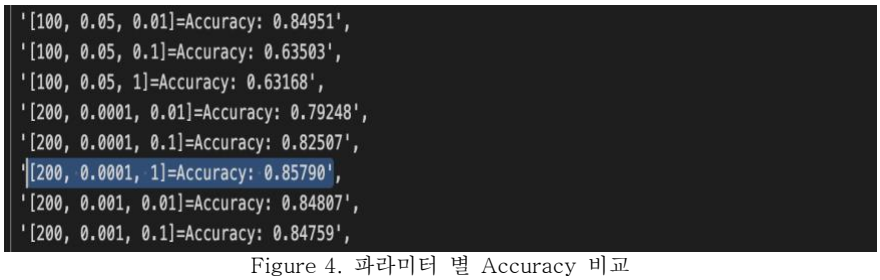

Figure 4. 파라미터 별 Accuracy 비교

5.3. Results

위와 같은 과정을 통해 SVM을 통한 Multi-classification을 진행하여 Kaggle public leader board 기준 0.74124의 정확도가 도출되었다. 위와 같이 Network traffic 상태를 분류해낼 수 있다면 병목 현상 등의 문제를 파악하여 네트워크를 안정적으로 운영하는 것에 도움을 주는 등, 다양한 네트워크 트래픽 관련 문제 해결에 도움을 줄 수 있을 것이다.

6. Discussion and Limitation

: 위와 같은 과정을 통해 구현한 SVM은 비선형적으로 데이터를 구분할 수 없기에 성능 향상에 한계를 가진다. 이러한 한계를 극복하기 위해 가우시안 커널을 적용하는 코드를 구현해보려고 하였지만, 수식에 대한 이해 부족으로 실패하였다. 또한, One vs rest기법에서 단순의 decision function의 값으로만 class를 분류하는 것도 모델의 설명력을 감소시킨다. 후에 추가적인 연구를 통해 위의 한계점을 극복하여 성능을 향상시킬 것을 기대한다.

7. References

(1) OnevsRest 클래스 참고-1
https://zephyrus1111.tistory.com/205
(2) OnevsRest 클래스 참고-2
https://tobigs.gitbook.io/tobigs/data-analysis/svm/python-svm-2
(3) PCA 구현 참고
https://chancoding.tistory.com/56
(4) PCA 수식 참고
https://datascienceschool.net/02%20mathematics/03.05%20PCA.html
(5) SVM 계산 부분 및 figure2 참고
https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47