데이터 마이닝 보고서

컴퓨터공학과

12161558 김혜윤

제출일 : 2020.05.17

⓵코드 설명

- `white_wine = pd.read_csv('/content/winequality-white.csv')`
  `red_wine = pd.read_csv('/content/winequality-red.csv')` 로 wine csv 파일을 읽어온다
  `merge_wine = pd.concat([red_wine,white_wine])` 는 red wine, white wine 각 파일을 통합해주는 코드이다.

- ```
  def generate_data(df,t_r):
    X = df.drop(['quality'],axis=1)
    Y = df['quality']
    X_train,X_test,Y_train,Y_test = train_test_split(X,Y,train_size = t_r)

    return X_train.values,Y_train,X_test.values,Y_test
  ```
  : X는 class를 제외한 attribute로 구성되고, Y는 model이 알아내야 하는 attribute를 가진다. Train_test_split()함수는 train을 t_r 비율 만큼 가지게 하도록 하여 x_train, x_test, y_train, y_test를 분리해준다.

- `x_train_r,y_train_r,x_test_r,y_test_r = generate_data(red_wine,0.7)` 으로 red_wine의 data를 train set과 test set으로 분리해준다.

- `model_r = Sequential()` 순차 모델 생성 : 순차적으로 레이어 층을 더해주는 모델, 만들기가 쉬움

- ```
  model_r.add(Dense(32,input_shape=(11,)))
  model_r.add(Activation('relu'))
  model_r.add(Dense(10))
  model_r.add(Activation('softmax'))
  ```
  : add 함수로 각 레이어 층을 더해준다. dense 레이어는 입력과 출력을 모두 연결해준다.

  ➔ Dense(32,input_shape=(11,)) : 32는 출력 뉴런의 수, input_shape 는 입력 뉴런을 설정

  ➔ Activation(“sigmoid”) : 활성화 함수를 설정하는데, relu는 rectifier 함수로 은닉층에 주로 사용되고, softmax는 다중 클래스 분류 문제에서 출력층에 주로 사용된다.

- ```
  opt = keras.optimizers.Adam(learning_rate=0.01)
  model_r.compile(loss='sparse_categorical_crossentropy',optimizer=opt,metrics=["accuracy"])
  ```
  ```
  %%time
  hist_r = model_r.fit(x_train_r,y_train_r,epochs=50,batch_size=512,validation_data=(x_test_r,y_test_r),verbose=2)
  ```
  : Adam optimizer. Learning_rate는 0 보다 크거나 같은 float 값으로 학습율이다. Complie()은 학습에 대한 설정을 해줘야 한다. Loss = “sparse_categorical_crossentropy” 다중 분류 손실 함수로 integer type 클래스이다. Metrics=[“accuracy”] 은 모델의 성능을 판정하는데 사용하는 지표 함수이며, 리스트 형태로 설정한다. Fit() 함수는 정해진 epoch 동안 model을 훈련한다. X_train, y_train은 입력 data이고, batch_size는 정수이며 기울기를 업데이트할 sample의 개수, verbose는 0: stdout에 log를 주지않고, 1:진행 바 형태의 로그 ,2: epoch당 1줄의 로그를 준다. Validation_datasms 검증 데이터로 구분되는 data이다. Epoch당 결과를 hist에 저장한다. Red, white 와 merge의 model은 기본적으로 동일하지만 epoch 의 값을 다르게 설정 해주었다. Red, white, merge 순으로 data양이 많으므로 그 순으로 epoch 양을 증가시켜주었다,

- ```
  test_loss, test_acc = model_m.evaluate(x_test_m,  y_test_m, verbose=2)
  print(test_loss)
  print(test_acc)
  ```
  : evaluate() 함수는 test set에 대해 batch 마다 손실과 정확도를 계산하고, 출력해 주었다.

```
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.plot(hist_r.history['loss'],'b-',label="training")
plt.plot(hist_r.history['val_loss'],'r',label="validation")
plt.title("loss")
plt.subplot(1, 2, 2)
plt.title("accuracy")
plt.plot(hist_r.history['accuracy'], 'b-', label="training")
plt.plot(hist_r.history['val_accuracy'], 'r:', label="validation")
plt.legend()
plt.tight_layout()
plt.show()
```

: epoch 별로 저장해 놓은 손실과 정확도를 graph로 출력하였다.

② 성능향상

- Learning_rate (학습률) 조정을 통해 성능향상을 기대해 봄

- ```
  opt = keras.optimizers.Adam(learning_rate=0.01)
  ```

  ```
  .106464942296346
  .518750011920929
  ```

- ```
  opt = keras.optimizers.Adam(learning_rate=0.1)
  ```

  ```
  1.223128358523051
  0.4333333373069763
  ```

- ```
  opt = keras.optimizers.Adam(learning_rate=0.05)
  ```

  ```
  1.0072004874547322
  0.550000011920929
  ```

  ➔ 0.05로 설정했을 때 성능이 제일 좋음.

- Dropout 사용 (Dropout : 네트웍의 일부를 생략하는 것)

- ```
  model_r.add(Dropout(0.5))
  ```

- ```
  1.2171227057774863
  0.3687500059604645
  ```

- 0.1, 0.5, 0.3 등 확률로 drop out을 해보니 오히려 성능이 저하됨

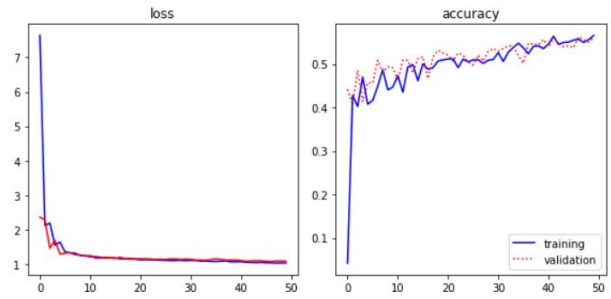
- Normalization

- ```
  ]  ###normalization
     from sklearn.preprocessing import MinMaxScaler
     scaler = MinMaxScaler()
     red_wine_n = red_wine.copy()
     red_wine_n [:11]=scaler.fit_transform(red_wine_n[:11])
  ```

- ```
  1.0937318046887716
  0.6187499761581421
  ```

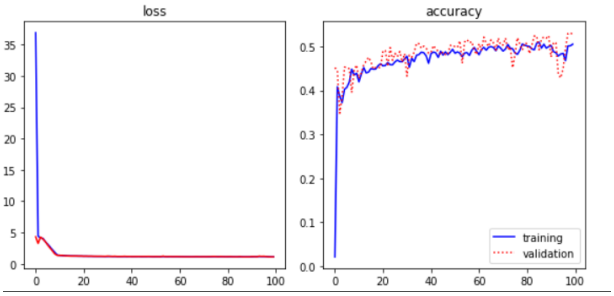
- Normalization 한 결과 조금이지만 상승했다. 그러나 loss 양은 크게 변화가 없다.
  ```

❸실험 결과 및 고찰

- 

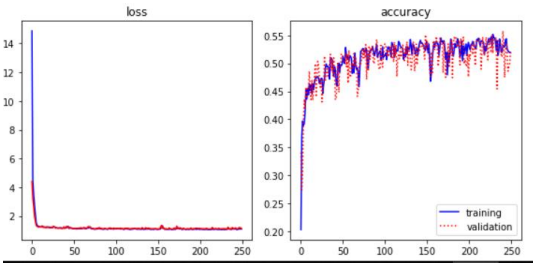
- 1.0934503118197123
0.5604166388511658

- Red_wine : 평가 결과 loss 1.093, accuracy 0.56 수준이 나왔다. 그래프는 loss는 이상적인 그래프인 log n 형태와 유사하게 출력됨을 확인 할 수 있었다. Accuracy 그래프는 epoch별로 들쭉날쭉한 부분들이 존재 하지만 대체적으로 상승됨을 알 수 있다.

- 

- 1.1191641783227726
0.5272108912467957

- White_wine : 평가 결과 loss 1.12, accuracy 0.53 수준이 나왔다. 그래프는 red_wine과 유사하다.

- 

- 1.1122087668149898
0.5210256576538086

- Merge_wine : loss 1.11 , accuracy 0.52 수준이 나왔다. 그래프 형태는 red, white wine과 유사하지만 둘보다 epoch 당 accuracy의 들쭉날쭉함이 좀 더 심하게 나타났다.

- 데이터가 추출되는 결과 마다 다르게 나타나지만 전반적으로 loss는 1.1, accuracy는 0.56 수준으로 나왔다. Loss graph는 log n 형태로 나타났고, accuracy도 증가하는 추세를 보였다. 하지만 다른 data set이 추출될 때마다 값이 변화하는 것으로 봐서 generate_data 함수 속 data set을 분류하는 공부가 좀

더 필요한 것으로 보이다. 그리고 loss의 값이 생각 이상으로 높게 나오는 것 또한 accuracy를 높이는 부분 보다 loss를 낮추기 위한 방법을 공부할 필요가 있다고 생각된다.