표지판, 시계이(가) 표시된 사진

자동 생성된 설명

2020년 2학기

Classification 실습과제

|  |
| --- |
| 과목: 데이터사이언스개론 |
| 담당교수: 신효섭 |
| 제출일: 2020.10.15(목) |
| 학과: 기술경영학과 |
| 학번: 201512082 |
| 성명: 백 승 호 |

|  |
| --- |
| [ 목 차 ]  1. Classification 알고리즘  1.1 SVM  1.2 Logistic Regression  1.3 Random Forest  2. Grade B 에 대한 Binary classification  2.1 train\_test\_split 결과  2.2 K-fold cross validation 결과  2.3 Binary classification 분석 결과  3. Grade A B C 에 대한 Multi-class classification  3.1 train\_test\_split 결과  3.2 K-fold cross validation 결과  3.3 Multi-class classification 분석 결과  4. 코드 |

**1. Classification 알고리즘**

**1.1 SVM**

SVM은 support vector machine의 약자이다. 주어진 샘플 그룹에 대해 그룹분류규칙을 찾아내는 기법 중 하나다. 패턴인식분야에서 제안된 알고리즘으로 글자 인식 분야에서 그 성능을 인정받아 기계학습분야에서 분류를 위한 대표적 알고리즘이 되었다. 서포트 벡터 머신의 목적은 n차원의 데이터공간에서 샘플그룹들을 구분해내는 최적의 분할선을 찾아내는것이다. 커널 함수를 어떻게 정의하느냐에 따라 선형 또는 비선형 분할선을 가진다.

데이터 공간에서 샘플 그룹을 구분할 수 있는 초평면의 가지 수는 무수히 많다. 이때 초평면에 의해 분리된 점들 중 초평면과 가장 가까운 점과 초평면사이의 거리를 마진이라고 정의하며 초평면마다 고유의 마진을 가지게 된다. SVM이 지향하는 최적의 분할선의 성질은 초평면들 중 최대의 마진을 가지는 초평면이며 이러한 초평면을 최대 마진 초평면 이라한다. 텍스트이(가) 표시된 사진

자동 생성된 설명

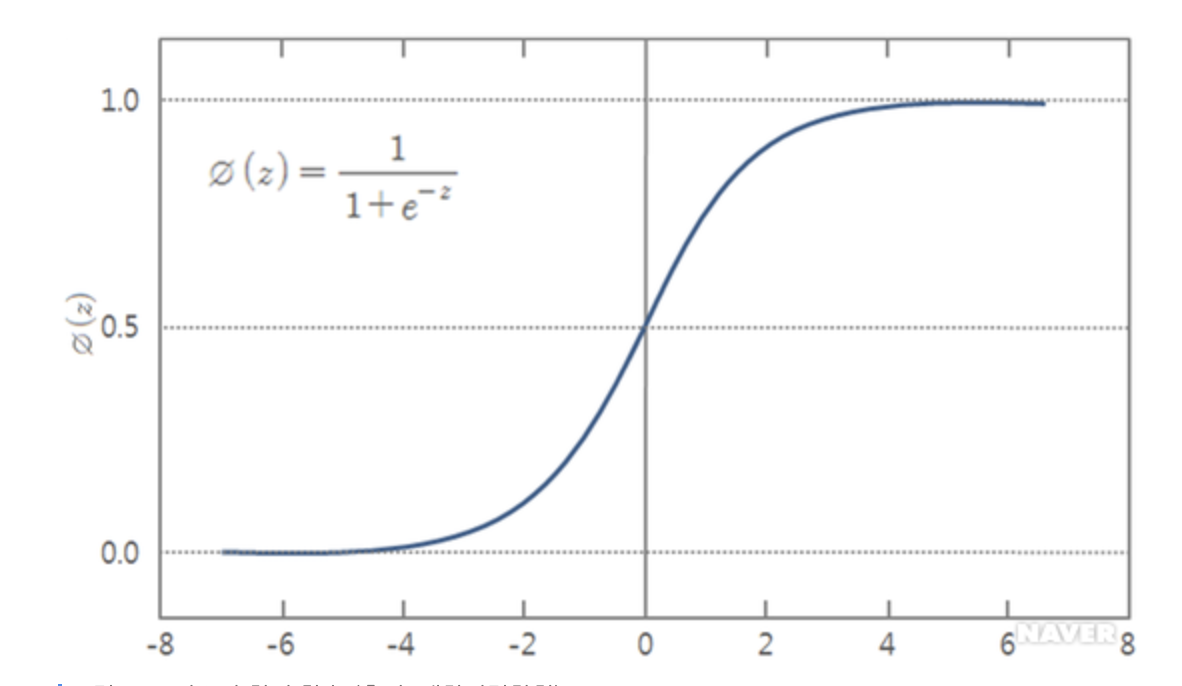
데이터 분포가 선형 분류를 하기에는 샘플 분할 경계가 복잡한 경우 선형 서포트 벡터 머신의 분류 성능이 저하된다. 이런 선형 서포트 벡터 머신의 한계는 커널이라는 매핑 함수를 도입해서 비선형 또는 비 모수 분할선을 정의함으로써 해결된다. 대표적인 비선형 함수는 n차 다항 함수 RBF함수가 있다. 라그랑즈 승수법을 적용하여 파라미터 값을 계산해낼 수 있다.

다음은 아이리스 데이터를 여러 커널 함수를 이용한 서포트 벡터 머신으로 분리한 예시이다. 아이리스 데이터는 Setosa, Vesicolor, Virginica의 3종류 아이리스가 있으며 꽃받침의 너비와 길이 정보를 가지고있다. 이를 서포트 벡터 머신으로 수행하면 초평면에 의해 3개의 데이터 공간으로 분리되며 다음과같이 공간내 속한 데이터는 같은 종끼리 묶이게 된다. 스크린샷이(가) 표시된 사진

자동 생성된 설명

**1.2 Logistic Regression**

로지스틱 회귀는 예측 분석을 위한 회귀분석중 특히 종속변수가 이분형일때 수행할 수 있는 회귀 분석 기법의 한 종류이다. 하나의 종속 이진 변수와 하나 이상의 숫자형, 명목형, 순서형의 독립변수 간의 관계를 함수를 이용하여 정량적으로 설명하는데 사용된다. 즉, 로지스틱 회귀 분석을 통해 하나의 종속 변수와 여러 독립 변수간의 다변수 회귀관계를 조사할 수 있다. 로지스틱 회귀는 3가지 장점이있다. 우선, 종속변수와 독립변수 사이의 관계가 비선형 관계로 식별되기 때문에 정규분포의 가정이 독립변수에 적용되지 않는다. 또, 명목형 연속형 순서형을 비롯한 다양한 독립 변수에 다양한 자료유형을 사용할 수 있기 때문에 복잡한 현상을 설명할 수 있다. 마지막으로 로지스틱 회귀 분석의 결과에는 각 요인에 대한 여러 분석값이 포함된다. 이런 분석값은 다른 지역을 대상으로 하는 유사한 연구에서 사용될 수 있다. 그러나 로지스틱 회귀는 종속변수가 발생할 가능성을 결정하기 위해 최적화된 회귀 방정식을 찾기 위한 계산을 반복하기 때문에 계산 프로세스를 추적할 수 없다는 점에서 한계가 있다.



로지스틱 회귀공식에서 종속 변수 및 하나 이상의 독립변수간의 관계는 다음과 같다.

계수 bi는 회귀 모델의 기울기 계수이며 b는 모형의 절편을, xi는 독립변수를 뜻한다. P는 사건이 발생할 확률이고 e는 자연로그다. 여러 독립변수를 로지스틱 회귀공식인 위의식에 적용하여 어떤 사건이 발생할 확률을 구하는 것이다. 로지스틱 회귀공식을 적용하면 이러한 확률 뿐 아니라 로지스틱 회귀공식을 사용해 로지스틱 다중 회귀 계수, 기울기 계수의 표준오차, 월 테스트, 유의수준 및 지수화 된 기울기 계수를 계산해 낼 수 있다.

**1.3 Random Forest**

의사결정나무는 학습 데이터에 대해 과대적합을 불러일으킬수있다는 단점이있다. 랜덤 포레스트는 의사결정 나무의 단점을 극복하기 위한 모델이다. 앙상블 모델의 일종으로 독립적으로 다수의 모델을 생성하고 결합해 예측값의 평균을 산출한 최종 모델을 생성한다. 이때 과대적합을 피하기위해 독립적으로 임의의 숲을 구성한다. 개별 모형들 사이의 상관계수를 감소시켜 모형 선능의 변동을 최소화할 수 있는 방법을 사용한다. 일반적으로 결정 트리를 이용한 방법의 경우 그 결과 또는 성능의 변동 폭이 크다. 특히 학습데이터에 따라 생성되는 결정 트리가 랜덤성에 따라 매우 다르기 때문에 일반화해서 사용하기엔 어려움이 따른다. 특히 결정 트리는 계층적 접근방식이기 때문에 만약 중간에 에러가 발생한다면 다음 단계로 에러가 계속 전파되는 특성을 가진다. 배깅 또는 랜덤노드 최적화와 같은 랜덤화 기술은 결정 트리가 가진 이런 단점을 극복하고 좋은 일반화 성능을 갖도록 한다. 배깅은 bootstrap aggregating의 약자로 부트스트랩을 통해 조금씩 다른 훈련 데이터에 대해 훈련된 기초 분류기들을 결합시키는 방법이다. 배깅을 통해 랜덤 포레스트를 훈련시키는 과정은 다음 과 같이 3단계로 진행된다. 첫째, 부트스트랩 방법을통해 T개의 훈련 데이터 셋을 생성한다. 둘째, T개의 기초 분류기들을 훈련시킨다. 셋째, 기초 분류기들을 하나의 분류기로 결합한다. 트리는 작은 편향과 큰 분산을 갖기 때문에 매우 깊이 성장한 트리는 훈련데이터에 대해 과적합하게 된다. 부트 스트랩과정은 트리들의 편향은 그대로 유지하면서 분산은 감소시키기 때문에 포레스트의 성능을 향상시킨다. 즉, 한 개의 결정 트리의 경우 훈련 데이터에 있는 노이즈에 대해서 매우 민감하지만 트리들이 서로 상관화 되어있지 않다면 여러 트리들의 평균은 노이즈에 대해 강인해진다.

랜덤포레스트의 장점은 큰 데이터에 적용 가능하며 일반화 성능이 좋고 스케일 변환이 불필요하다는 것이다. 그러나 내부를 알 수 없는 블랙박스 모델이며 텍스트 데이터에는 비적합하다는 특징을 가지고있다.

랜덤 포레스트는 여러개의 의사결정나무를 조합해 최종 모델을 생성하므로 가지치기 기준 역시 의사결정 나무와 같다.

랜덤포레스트역시 scikitlearn이 제공하고있으며 from sklearn.ensemble import RandomForestClassifier 라이브러리를 사용한다. 중요 매개 변수로 포레스트의 크기 T와 최대허용깊이 D, 임의성의 정도와 종류, 노드분할함수의 선택, 훈련목적함수의 정의, 실제 적용에서의 특징 벡터의 선택이다.

응용사례로 키넥트에서의 신체 트랙킹, 컴퓨터 단층 촬영 영상 내 해부학 구조 검출 및 위치파악, 다채널 자기공명영상 내 고악성도 신경교종 검출등의 연구가 있다.

**2. Grade B 에 대한 Binary classification**

**2.1 train\_test\_split 결과**

|  |  |  |  |
| --- | --- | --- | --- |
|  | SVM | Logistic Regression | Random Forest |
| Accuracy | 0.709677 | 0.677419 | 0.709677 |
| Precision | 0.000000 | 0.000000 | 0.500000 |
| Recall | 0.000000 | 0.000000 | 0.444444 |
| F1 score | 0.000000 | 0.000000 | 0.470588 |

**2.2 K-fold cross validation 결과**

|  |  |  |  |
| --- | --- | --- | --- |
|  | SVM | Logistic Regression | Random Forest |
| Accuracy | 0.663157 | 0.563742 | 0.706432 |
| Precision | 0 | 0.04 | 0.563333 |
| Recall | 0 | 0.666667 | 0.432381 |
| F1 score | 0 | 0.05 | 0.480757 |

**2.3 Binary classification 분석결과**

분석 결과 SVM과 Logistic Regression 알고리즘을 통해 Binary classification을 하면 Accuracy는 어느정도 나오지만 Precision, Recall, F1 score는 아예 0이거나 굉장히 낮게 나오는걸 확인할 수 있었다. Random Forest의 경우 앞선 두 알고리즘에 비해 상대적으로 높은 수치를 보이지만 f1 score가 0.5이하로, 굉장히 낮은 성능을 보인다는것을 확인할수있었다. 따라서 3가지 알고리즘 모두 주어진 데이터 셋에 대한 Binary classification성능이 매우 떨어진다는 결론을 내릴수있다. 왜 SVM과 Logistic Regression이 이렇게 낮은 정확도를 보이는지 생각해보면, 우선 우리는 Binary classification을 위해 ‘B’와 ‘B가 아닌 것’으로 재분류를 했다. 그로 인해 B는 A와 C 인 경우의 사이에 위치하게 된다. 그런데 이때 SVM과 Logistic Regressioin은 레이블과 각 레이블의 분포를 통해 그 둘간을 구분 지을수 있는 선을 그리려고 하는데 B가 A와 C 중간에 위치하기 때문에 선 하나로 구분하기 불가능해진다. 따라서 SVM과 Logistic Regression의 결과가 0으로 매우 낮게 나오게 된 이유는, 우리가 분류하려고 지정한 B의 독립 데이터 분포가, 우리가 구분 지어야 할 A와 C의 독립 데이터들 사이에 분포되어있기 때문이라고 추측 할 수 있다. 실제로 양 끝 단에 분포하는 “A와 A가 아닌 경우”, 혹은 “C와 C가 아닌경우”의 Binary classification을 진행하면 약 0.7 ~ 0.8 정도의 높은 정확도를 보이게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그림 1 “A”와 “A가 아닌 것”으로 Binary classification 진행한 결과

**3. Grade A B C 에 대한 Multi-class classification**

**3.1 train\_test\_split 결과 및 분석**

**[ SVM ]**

약 0.580645 Accuracy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1\_score** |
| **A** | 0.583333 | 0.636364 | 0.608696 |
| **B** | 0.458333 | 0.55 | 0.5 |
| **C** | 0.580645 | 0.580645 | 0.580645 |

**[ Logistic Regression]**

약 0.580645 Accuracy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1\_score** |
| **A** | ﻿0.583333 | 0﻿.636364 | ﻿0.608696 |
| **B** | ﻿0.476190 | ﻿0.500000 | ﻿0.487805 |
| **C** | ﻿0.580645 | 0﻿.580645 | 0﻿.580645 |

**[ Random Forest ]**

약 0.645161 Accuracy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1\_score** |
| **A** | ﻿﻿0.636364 | ﻿0.636364 | ﻿0.636364 |
| **B** | ﻿0.578947 | 0.55 | 0.564103 |
| **C** | 0.645161 | 0.645161 | 0.645161 |

**3.2 K-fold cross validation 결과 및 분석**

**[ SVM ]**

약 ﻿0.62046 Accuracy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1\_score** |
| **A** | ﻿0.645 | ﻿0.507857 | ﻿0.56166 |
| **B** | ﻿0.51747 | ﻿0.581794 | ﻿0.547176 |
| **C** | ﻿0.620467 | ﻿0.620467 | ﻿0.620467 |

**[ Logistic Regression]**

약 ﻿0.663742 Accuracy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1\_score** |
| **A** | ﻿0.62 | ﻿﻿0.499285 | ﻿﻿0.546010 |
| **B** | ﻿0.578327 | ﻿0.588461 | ﻿0.582043 |
| **C** | ﻿0.663742 | ﻿0.663742 | ﻿0.6637426 |

**[ Random Forest ]**

약 ﻿0.652046 Accuracy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1\_score** |
| **A** | ﻿0.59333 | ﻿0.507857 | ﻿0.545592 |
| **B** | ﻿0.583396 | ﻿0.563699 | ﻿0.573119 |
| **C** | ﻿0.652046 | ﻿0.6520467 | ﻿0.652046 |

**3.3 Multi-class classification**

분석 결과 3가지 알고리즘 모두 멀티 클래스 분류에 대해선 어느정도 정확도를 보이는것으로 나타난다. F1 score기준으로 평가하면 SVM, Logistic Regression, Random Forest 세가지 알고리즘 보두 약 60 ~ 70% 정도의 정확도를 보이며 train test split시 Random Forest가, K-fold cross validation시 Logistic Regression이 제일 성능이 좋은 것으로 보여진다. 결국 3가지 알고리즘 모두 Multi class classification에서 비슷한 성능을 보이며 binary classification을 진행한것 보다 더 높은 정확도를 보인다.

**4. 코드**

|  |
| --- |
| ﻿#!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  """  Created on Tue Oct 6 18:56:32 2020  @author: baegseungho  """  import pymysql  import pandas as pd  import numpy as np  from sklearn import svm  from sklearn.linear\_model import LogisticRegression  from sklearn.ensemble import RandomForestClassifier  def load\_data():  file = 'db\_score\_3\_labels.xlsx' #####경로확인  db\_score = pd.read\_excel(file)    conn = pymysql.connect(host='localhost', user='root', password='password', db='university')  curs = conn.cursor(pymysql.cursors.DictCursor)    drop\_sql = """drop table if exists db\_score\_3\_labels""" ####테이블확인  curs.execute(drop\_sql)  conn.commit()    import sqlalchemy  database\_username = 'root'  database\_password = 'password'  database\_ip = 'localhost'  database\_name = 'university'  database\_connection = sqlalchemy.create\_engine('mysql+pymysql://{0}:{1}@{2}/{3}'.format(database\_username, database\_password, database\_ip, database\_name))  db\_score.to\_sql(con=database\_connection, name='db\_score\_3\_labels', if\_exists='replace') ###name 확인  def performance\_eval(y, y\_predict, flag):  if flag == 0:  tp, tn, fp, fn = 0,0,0,0  for y, yp in zip(y, y\_predict):  if y == 1 and yp == 1:  tp += 1  elif y == 1 and yp == -1:  fn += 1  elif y == -1 and yp == 1:  fp += 1  else:  tn += 1  accuracy = (tp+tn)/(tp+tn+fp+fn)  if tp + fp != 0:  precision = tp/(tp+fp)  else:  precision = 0  recall = tp/(tp+fn)  if precision + recall != 0:  f1\_score = 2\*precision\*recall/(precision+recall)  else:  f1\_score = 0  return accuracy, precision, recall, f1\_score  else:  i = 0  tp = [0,0,0]  tn = [0,0,0]  fp = [0,0,0]  fn = [0,0,0]  y\_temp = y  yp\_temp = y\_predict  while i < 3:  y = y\_temp  y\_predict = yp\_temp  for y, yp in zip(y, y\_predict):  if y == i and yp == i:  tp[i] += 1  elif y == i and yp != i:  fn[i] += 1  elif y != i and yp == i:  fp[i] += 1  elif y != i and yp != i:  tn[i] += 1  i += 1  i = 0  num = 0  dnum = 0  while i < 3:  num += tp[i]  i += 1  accuracy = num/len(y\_predict)    i = 0  num = 0  dnum = 0  precision = [0,0,0]  recall = [0,0,0]  f1\_score = [0,0,0]  while i < 3:  num += tp[i]  dnum += tp[i]+fp[i]  precision[i] = num/dnum  i += 1    i = 0  num = 0  dnum = 0  while i < 3:  num += tp[i]  dnum += tp[i]+fn[i]  if dnum != 0:  recall[i] = num/dnum  else:  recall[i] = 0  i += 1    i = 0  while i < 3:  if precision[i] + recall[i] != 0:  f1\_score[i] = 2\*precision[i]\*recall[i]/(precision[i]+recall[i])  else:  f1\_score[i] = 0  i += 1    return accuracy, precision, recall, f1\_score  def train\_split(X, y, flag):  from sklearn.model\_selection import train\_test\_split    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size = 0.33, random\_state=42)  print("SVM")  clf = svm.SVC(kernel='linear')  clf.fit(X\_train, y\_train)  y\_predict = clf.predict(X\_test)  accuracy, precision, recall, f1\_score = performance\_eval(y\_test, y\_predict, flag)  print("accuracy=%f" %accuracy)  if flag == 0:  print("precision=%f" %precision)  print("recall=%f" %recall)  print("f1\_score=%f" %f1\_score)  print()  else:  print("Grade A")  print("precision=%f" %precision[0])  print("recall=%f" %recall[0])  print("f1\_score=%f" %f1\_score[0])  print()  print("Grade B")  print("precision=%f" %precision[1])  print("recall=%f" %recall[1])  print("f1\_score=%f" %f1\_score[1])  print()  print("Grade C")  print("precision=%f" %precision[2])  print("recall=%f" %recall[2])  print("f1\_score=%f" %f1\_score[2])  print()    print("Logistic Regression")  model = LogisticRegression(max\_iter=50000, C=10, random\_state=42)  model.fit(X\_train, y\_train)  y\_predict = model.predict(X\_test)  accuracy, precision, recall, f1\_score = performance\_eval(y\_test, y\_predict, flag)  if flag == 0:  print("precision=%f" %precision)  print("recall=%f" %recall)  print("f1\_score=%f" %f1\_score)  print()  else:  print("Grade A")  print("precision=%f" %precision[0])  print("recall=%f" %recall[0])  print("f1\_score=%f" %f1\_score[0])  print()  print("Grade B")  print("precision=%f" %precision[1])  print("recall=%f" %recall[1])  print("f1\_score=%f" %f1\_score[1])  print()  print("Grade C")  print("precision=%f" %precision[2])  print("recall=%f" %recall[2])  print("f1\_score=%f" %f1\_score[2])  print()  print("Random Forest")  forest = RandomForestClassifier(n\_estimators=100, oob\_score=True, random\_state=42)  forest.fit(X\_train, y\_train)  y\_predict = forest.predict(X\_test)  accuracy, precision, recall, f1\_score = performance\_eval(y\_test, y\_predict, flag)  if flag == 0:  print("precision=%f" %precision)  print("recall=%f" %recall)  print("f1\_score=%f" %f1\_score)  print()  else:  print("Grade A")  print("precision=%f" %precision[0])  print("recall=%f" %recall[0])  print("f1\_score=%f" %f1\_score[0])  print()  print("Grade B")  print("precision=%f" %precision[1])  print("recall=%f" %recall[1])  print("f1\_score=%f" %f1\_score[1])  print()  print("Grade C")  print("precision=%f" %precision[2])  print("recall=%f" %recall[2])  print("f1\_score=%f" %f1\_score[2])  print()  def k\_fold\_cross\_validation(X, y, flag):  from sklearn.model\_selection import KFold  kf = KFold (n\_splits=5, random\_state=42, shuffle=True)  s\_accuracy = []  l\_accuracy = []  f\_accuracy = []  if flag == 1:  s\_precision = [[],[],[]]  s\_recall = [[],[],[]]  s\_f1\_score = [[],[],[]]  l\_precision = [[],[],[]]  l\_recall = [[],[],[]]  l\_f1\_score = [[],[],[]]  f\_precision = [[],[],[]]  f\_recall = [[],[],[]]  f\_f1\_score = [[],[],[]]  else:  s\_precision = []  s\_recall = []  s\_f1\_score = []  l\_precision = []  l\_recall = []  l\_f1\_score = []  f\_precision = []  f\_recall = []  f\_f1\_score = []  for train\_index, test\_index in kf.split(X):  X\_train, X\_test = X[train\_index], X[test\_index]  y\_train, y\_test = y[train\_index], y[test\_index]  i = 0  while i < 3:  if i == 0:  clf = svm.SVC(kernel='linear')  clf.fit(X\_train, y\_train)  y\_predict = clf.predict(X\_test)  acc, prec, rec, f1 = performance\_eval(y\_test, y\_predict, flag)  s\_accuracy.append(acc)  if flag == 0:  s\_precision.append(prec)  s\_recall.append(rec)  s\_f1\_score.append(f1)  else:  j = 0  while j < 3:  s\_precision[j].append(prec[j])  s\_recall[j].append(rec[j])  s\_f1\_score[j].append(f1[j])  j += 1  elif i == 1:  model = LogisticRegression(max\_iter=50000, C=10, random\_state=42)  model.fit(X\_train, y\_train)  y\_predict = model.predict(X\_test)  acc, prec, rec, f1 = performance\_eval(y\_test, y\_predict, flag)  l\_accuracy.append(acc)  if flag == 0:  l\_precision.append(prec)  l\_recall.append(rec)  l\_f1\_score.append(f1)  else:  j = 0  while j < 3:  l\_precision[j].append(prec[j])  l\_recall[j].append(rec[j])  l\_f1\_score[j].append(f1[j])  j += 1  else:  forest = RandomForestClassifier(n\_estimators=100, oob\_score=True, random\_state=42)  forest.fit(X\_train, y\_train)  y\_predict = forest.predict(X\_test)  acc, prec, rec, f1 = performance\_eval(y\_test, y\_predict, flag)  f\_accuracy.append(acc)  if flag == 0:  f\_precision.append(prec)  f\_recall.append(rec)  f\_f1\_score.append(f1)  else:  j = 0  while j < 3:  f\_precision[j].append(prec[j])  f\_recall[j].append(rec[j])  f\_f1\_score[j].append(f1[j])  j += 1  i += 1    import statistics  print("SVM")  print("average\_accuracy =", statistics.mean(s\_accuracy))  if flag == 0:  print("average\_precision =", statistics.mean(s\_precision))  print("average\_recall =", statistics.mean(s\_recall))  print("average\_f1\_score =", statistics.mean(s\_f1\_score))  print()  else:  j = 0  while j < 3:  if j == 0:  print("Grade A")  print("average\_precision =", statistics.mean(s\_precision[j]))  print("average\_recall =", statistics.mean(s\_recall[j]))  print("average\_f1\_score =", statistics.mean(s\_f1\_score[j]))  print()  elif j == 1:  print("Grade B")  print("average\_precision =", statistics.mean(s\_precision[j]))  print("average\_recall =", statistics.mean(s\_recall[j]))  print("average\_f1\_score =", statistics.mean(s\_f1\_score[j]))  print()  else:  print("Grade C")  print("average\_precision =", statistics.mean(s\_precision[j]))  print("average\_recall =", statistics.mean(s\_recall[j]))  print("average\_f1\_score =", statistics.mean(s\_f1\_score[j]))  print()  j += 1    print("Logistic Regression")  print("average\_accuracy =", statistics.mean(l\_accuracy))  if flag == 0:  print("average\_precision =", statistics.mean(l\_precision))  print("average\_recall =", statistics.mean(l\_recall))  print("average\_f1\_score =", statistics.mean(l\_f1\_score))  print()  else:  j = 0  while j < 3:  if j == 0:  print("Grade A")  print("average\_precision =", statistics.mean(l\_precision[j]))  print("average\_recall =", statistics.mean(l\_recall[j]))  print("average\_f1\_score =", statistics.mean(l\_f1\_score[j]))  print()  elif j == 1:  print("Grade B")  print("average\_precision =", statistics.mean(l\_precision[j]))  print("average\_recall =", statistics.mean(l\_recall[j]))  print("average\_f1\_score =", statistics.mean(l\_f1\_score[j]))  print()  else:  print("Grade C")  print("average\_precision =", statistics.mean(l\_precision[j]))  print("average\_recall =", statistics.mean(l\_recall[j]))  print("average\_f1\_score =", statistics.mean(l\_f1\_score[j]))  print()  j += 1  print("Random Forest")  print("average\_accuracy =", statistics.mean(f\_accuracy))  if flag == 0:  print("average\_precision =", statistics.mean(f\_precision))  print("average\_recall =", statistics.mean(f\_recall))  print("average\_f1\_score =", statistics.mean(f\_f1\_score))  print()  else:  j = 0  while j < 3:  if j == 0:  print("Grade A")  print("average\_precision =", statistics.mean(f\_precision[j]))  print("average\_recall =", statistics.mean(f\_recall[j]))  print("average\_f1\_score =", statistics.mean(f\_f1\_score[j]))  print()  elif j == 1:  print("Grade B")  print("average\_precision =", statistics.mean(f\_precision[j]))  print("average\_recall =", statistics.mean(f\_recall[j]))  print("average\_f1\_score =", statistics.mean(f\_f1\_score[j]))  print()  else:  print("Grade C")  print("average\_precision =", statistics.mean(f\_precision[j]))  print("average\_recall =", statistics.mean(f\_recall[j]))  print("average\_f1\_score =", statistics.mean(f\_f1\_score[j]))  print()  j += 1  def binary\_classification(data):  X = [ (t['homework'], t['discussion'], t['midterm']) for t in data ]  X = np.array(X)  y = [ 1 if (t['grade'] == 'B') else -1 for t in data ]  y = np.array(y)  print("======Binary Classification=====")  print()  print("-----Train\_test\_split-----")  print()  train\_split(X, y, 0)  print("-----K-Fold-----")  print()  k\_fold\_cross\_validation(X, y, 0)  def multi\_classification(data):  X = [ (t['homework'], t['discussion'], t['midterm']) for t in data ]  X = np.array(X)  y = [ 0 if (t['grade'] == 'A') else 1 if (t['grade'] == 'B') else 2 for t in data ]  y = np.array(y)  print("======Multi Classification=====")  print()  print("-----Train\_test\_split-----")  print()  train\_split(X, y, 1)  print("-----K-Fold-----")  print()  k\_fold\_cross\_validation(X, y, 1)  load\_data()  conn = pymysql.connect(host='localhost', user='root', password='password', db='university')  curs = conn.cursor(pymysql.cursors.DictCursor)  sql = "select homework, discussion, midterm, grade from db\_score\_3\_labels"  curs.execute(sql)  data = curs.fetchall()  curs.close()  conn.close()  binary\_classification(data)  multi\_classification(data) |