HW4 Titanic

2조
민현기, 이수호, 임지은, 전민재, 양성욱

1. Logit 방식, SVM 방식 등 2가지의 방식을 이용하여 “survival“을 예측하는 classifier를 만들려고 한다. Scikit-learn, Tensorflow, Keras 등 머신러닝 라이브러리를 이용하지 않고 classifier를 만들고 의미를 해석하라.

1) 데이터 전처리

(1) 변수확인 및 데이터 전처리

변수	정의	전처리
PassengerId, ticket, Name	승객번호, 티켓번호, 이름	- 생존여부와 큰 상관관계가 없을 것으로 예상되어 변수에서 제거
Cabin	객실번호	- 결측치(77%)가 많아 변수에서 제거
Age	나이	- 결측치(20%)를 평균값으로 대체
Embarked	탑승한 곳*	- 결측치(0.2%)를 최빈값으로 대체 - 범주형 데이터로 수치화 진행 (Q항 기준 가변수 생성)
Sex	성별	- 범주형 데이터로 수치화 진행 (남성0, 여성1)
Sibsp, Parch	형제/자매 수, 부모/자녀 수	- 가족의 크기라는 파생변수(Family) 생성

* C=Cherbourg, Q=Queenstown, S=Southampto

2) classifier 구현

(1) logit

☐ activation function: Sigmoid

시그모이드 함수는 임의의 실숫값[$-\infty,\infty$]을 입력으로 받아 확률값[0, 1]의 범위에 해당하는 값을 출력한다. 이런 성질을 이용해 Threshold를 0.5로 삼아, 시그모이드 값이 0.5를 초과하면 1을 반환하고, 0.5 이하인 경우에는 0을 반환하게 하여 0 또는 1의 이진값을 출력하도록 구현하였다.

```
# 활성화함수
def sigmoid(z):
    return 1/(1+np.exp(-z))
```

☐ loss function: the log likelihood function

위 분류기의 종속변수는 0 또는 1의 이진값을 가지는 확률변수이다, 이에 베르누이의 확률분포를 따를 것이라 추정하면 로그우드함수를 아래와 같이 표현할 수 있다.

```
diff = ( y *np.log(p ) +( 1 -y ) *np.log( 1 -p))  # 우도
cost = np.sum(diff ) * 1 /( N *M)  # 평균
```

입력값과 파라미터가 주어졌을 때 출력값의 확률이 최대화된 상태를 최적의 추정 즉, 손실이 최소화되는 것이라 할 수 있기 때문에 위 로그우도함수를 경사하강법을 이용하여 최대화시켰다.

□ loss function 최적화: gradients descent

로그우도함수를 파라미터에 대하여 편미분하여 기울기값을 구하고 이를 다시 파라미터에 더하여 파라미터를 갱신하여 주었다.

```
# 기울기 구하기
grad_w = np.dot((- p +y_train) ,x_train ) * 1 /( N *M)
grad_b = np.sum(- p +y_train ) * 1 /( N *M)

# 파라미터 갱신
w = w+ rate * grad_w
b = b + rate * grad_b
```

(2) SVM (함께 첨부한 Yet more simple SMO algorithm 논문과 코드를 참조하였으며, 코드에서 prediction 및 accuracy 부분을 추가하였습니다)

□ 기존 SVM의 dual problem은 maximize $\mathcal{L}^* = \sum_i \lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j K(x_i;x_j)$

with constraints: $0 \le \lambda_i \le C$, 로 표현되는데,

$$\sum_i \lambda_i y_i = 0$$

이를 SMO 알고리즘을 통해 구현할 수 있으며, 논문에 의하면 SMO 알고리즘의 heuristics이 개입되는 부분을 버리면 SMO 알고리즘의 구현이 간단해지고, 다시 선을 벡터로 표현하여 이를 numpy로 간단하게 구현 가능하다고 한다.

□ 구현 목적 함수

$v_0 = (\lambda_M\ ,\ \lambda_L)^T,\ k_0 = (1-\lambda^T K_M\ ,\ 1-\lambda^T K_L),\ Q = \begin{pmatrix} K_{M,M} & K_{M,L} \\ K_{L,M} & K_{L,L} \end{pmatrix}$ 를 활용하여

주어진 목적 함수를 재표현하면 $\overline{\mathcal{L}}^* = k_0^T v_0 + \frac{1}{2} v_0^T Q v_0$가 되며, 이를 SMO 알고리즘의

아이디어에 따라 λ_M과 λ_L을 제외한 모든 람다를 고정한 다음 제약조건을 만족하며 람다를 최소화한다. 이후 v를 새로운 스칼라 변수 t에 대해 $v(t) = v_0 + tu$, $u = (-y_L \ , \ y_M)^T$ 라고 가정한 뒤, t에 대한 최소화를 수행한다.

```
def fit(self, X, y):
    self.X = X.copy()
    self.y = y * 2 - 1 # 클래스 범위를 0/1에서 -1/1로 조정
    self.lambdas = np.zeros_like(self.y, dtype=float) # 제약 조건 중 0 이상 범위를 만족하는 lambda 생성
    self.K = self.kernel(self.X, self.X) * self.y[:, np.newaxis] * self.y # kernel matrix 생성

    for _ in tqdm.tqdm(range(self.max_iter)):
        for idxM in range(len(self.lambdas)): # lambda_M 반복
            idxL = np.random.randint(0, len(self.lambdas)) # lambda_L 랜덤 선택
            Q = self.K[[[idxM, idxM], [idxL, idxL]], [[idxM, idxL], [idxM, idxL]]] # lambda_M 및 lambda_L에 대한 Kernel matrix(Q) 설정
            v0 = self.lambdas[[idxM, idxL]] # v_0 설정
            k0 = 1 - np.sum(self.lambdas * self.K[[idxM, idxL]], axis=1) # k_0 설정
            u = np.array([-self.y[idxL], self.y[idxM]]) # u 벡터 설정
            t_max = np.dot(k0, u) / (np.dot(np.dot(Q, u), u) + 1E-15) # argmax lambda*(t)를 계산하고, lambda_M 및 lambda_L이 같은 경우에 업데이트를 위하여 작은 상수를 더함
            self.lambdas[[idxM, idxL]] = v0 + u * self.restrict_to_square(t_max, v0, u) # Cost를 활용하여 범위를 제한하여 제약 조건을 충족시킴

    idx, = np.nonzero(self.lambdas > 1E-15) # 서포트 벡터 선택
    self.b = np.sum((1.0 - np.sum(self.K[idx] * self.lambdas, axis=1)) * self.y[idx]) / len(idx) # bias 계산
```

2. 구축된 classifier를 이용하여 predicted survival을 train_tatinic 내에서 만들고 실제 survival과 비교하여 예측 정확도를 각각 계산하라.

1) train/test split

□ 데이터 셋에서 7:3 비율로 train / test 셋을 분리하였으며 데이터 셋의 index를 구한다음 np.random.shuffle 함수를 통하여 idx를 재배열하여 train과 test 데이터를 분리함

```
# 종속/독립변수 나누기
y_data=y_data
x_data=df1

y_data=np.array(y_data)
x_data=np.array(x_data)



# 학습/검증용 나누기
size = y_data.shape[0] # 891

idx=np.arange(size)     # 일련번호 생성 후 재배열
np.random.shuffle(idx)

num=int(size*0.7)       # 분리 비율(7:3)

x_train = x_data[idx[:num]]
x_test = x_data[idx[num:]]
y_train = y_data[idx[:num]]
y_test = y_data[idx[num:]]
```

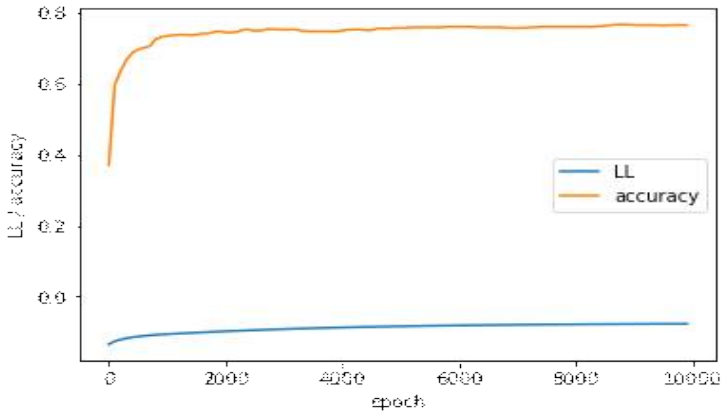
2) 예측 정확도 확인

(1) Logit

☐ parameter 설정 : 학습률 0.05 , 학습횟수 10,000회

☐ accuracy: 75%

* 학습횟수에 따른 우도값과 정확도의 변화



☐ Logit 방식을 통한 각 변수의 유의성 추정

	bias	Sex	age	family	fare	pclass	PortS	PortC
coef	1.176	2.746	-0.025	-0.322	0.037	-0.749	-0.307	0.263
odds	3.241	15.580	0.975	0.725	1.038	0.473	0.736	1.301
per	2.241	14.580	-0.025	-0.275	0.038	-0.527	-0.264	0.301

- 생존에 큰 영향을 미친 것은 성별, 오차항, 티켓등급, 탑승항, 가족크기, 나이 순으로 나타났다.
- 오차항의 크기가 큰 것으로 보아 추가적으로 탐색할만한 요소가 있을 것으로 판단된다.
- 티켓등급이 높을수록 생존 가능성이 높은 것으로 보아, 높은 등급의 객실이 탈출하기에 용이한 위치에 있었던 것으로 보인다. 이에 분석에서 배제되었던 'Cabin'(객실번호)과 'ticket'(티켓번호) 자료의 추가분석이 필요해 보인다.
- Q항 탑승객 대비 C항 탑승객에서 생존 가능성이 높았던 반면, S항 탑승객에서는 생존 가능성이 낮게 나타났다. 상대적으로 C항에서 경제력을 갖춘 승객이 많이 탑승한 것으로 보인다.
- 나이가 큰 영향을 미치지 않는 것으로 보이나, 실제 나이에 따른 생존 확률을 추가적으로 탐색하고 놓친 부분이 없는지 확인해야 할 것으로 판단되다.

(2) SVM

☐ parameter 설정 : Cost: 100, iteration: 1000, (polynomial kernel의 경우 3차 함수, radial basis function의 경우 gamma: 1로 설정)

☐ accuracy

- linear kernel: 0.65
- radial basis function kernel: 0.61
- polynomial kernel: 0.62

```
model = Custom_SVM()
model.fit(x_train, y_train)
predict = model.predict(x_test)
model.accuracy(predict, y_test)
```

```
100%|                                                                                | 1000/1000 [00:54<00:0
0, 18.26it/s]
0.6567164179104478
```

```
model = Custom_SVM(kernel='rbf')
model.fit(x_train, y_train)
predict = model.predict(x_test)
model.accuracy(predict, y_test)
```

```
100%|                                                                                | 1000/1000 [00:55<00:0
0, 18.06it/s]
0.6156716417910447
```

```
model = Custom_SVM(kernel='poly')
model.fit(x_train, y_train)
predict = model.predict(x_test)
model.accuracy(predict, y_test)
```

```
100%|                                                                                | 1000/1000 [00:54<00:0
0, 18.25it/s]
0.6268656716417911
```

3. 2의 결과 중 2가지 방식 중 가장 좋은 예측력을 보이는 방식을 무엇인가? 그리고 그 이유는 무엇인가? (이론적으로 설명하라.)

☐ logit 모델의 경우 75%로, SVM 모델의 경우 평균 약 64% 정도로 logit 모델의 예측력이 높은 것으로 나타났다.

☐ 서포트 벡터 머신의 경우 hyperplane과 feature 사이의 거리를 계산하며 margin의 크기를 최대화하게 되는데, 이때 변수들의 크기를 조정하지 않으면 한 feature에 매우 큰 값이 있는 경우 거리를 계산할 때 큰 값이 존재하는 feature를 우선하여 margin을 최대화하는 시도를 하게 되므로 prediction에 있어 일반화 성능을 저해시키는 요소로 작용할 수 있다.

☐ 따라서, 이를 변수들의 값을 보정하기 위한 여러 scaling 기법을 활용하여 예측 성능을 높일 수 있는데, 테스트를 위해 변수를 min-max scaling 기법을 활용하여 값을 조정한 뒤 결과를 비교한 결과, SVM이 큰 폭으로 상승하였음을 확인할 수 있었다.

□ 예측 정확도 비교

모델		min-max scaling 전 정확도	min-max scaling 후 정확도
Logit		**75.3%**	82.0%
SVM	linear kernel	65.6%	81.3%
	radial basis function	61.5%	**83.9%**
	polynomial kernel	62.6%	82.8%