Fundamentals of Machine Learning (Spring 2025)

Homework #2 (100 Pts, Due date: April 16)

Student ID 2021311716

Name 김서영

Files you need to submit

`ML_HW2_YourName_StudentID.zip`: All code in the directory and your report.

- `ML_HW2_YourName_StudentID.pdf`: Your report converted into a PDF file.
- `Code(folder)`

NOTE 1: Please write your code only in the 'EDIT HERE' section.

NOTE 2: You may need to install the NumPy and Matplotlib libraries.

NOTE 3: Please read the comments in the code carefully.

1. **[Implementation]** Implement the logistic regression and SoftMax classifier in your code. **Once you have completed your implementation, run the checker code ('1_LogisticRegression_Checker.py' or '1_SoftmaxClassifier_Checker.py') to validate if your code is executed correctly.**

(a) [10 pts] Implement the functions in 'models/LogisticRegression.py'. ('**forward**', '**compute_grad**', '**_sigmoid**', and '**eval**' respectively). Given a mini-batch data (X, Y), the error function for a mini-batch is defined as follows:

$$\boldsymbol{E(w) = -\frac{1}{|\mathcal{B}|}\sum_{(x_i,y_i)\in\mathcal{B}} y_i * \log(\hat{y}_i) + (1-y_i) * \log(1-\hat{y}_i),}$$

$$\textbf{where } \hat{y}_i = \text{sigmoid}(w^T x_i), |\mathcal{B}| \textbf{ is the number of the mini} - \textbf{batch samples.}$$

Fill in your code here. Please submit your code to i-campus.

Answer:

```
def forward:
# ========== EDIT HERE ==========
    z = np.dot(x, self.W)
    logits = self._sigmoid(z)
```

```
        loss = -np.mean(y * np.log(logits + eps) + (1 – y) * np.log(1 – logits + eps)
# ==================================


def compute_grad:
# ========== EDIT HERE ==========
        grad_weight = np.dot(x.T, (logit – y)) / num_data
# ==================================


def _sigmoid:
# ========== EDIT HERE ==========
        sigmoid = 1 / (1 + np.exp(-x))
# ==================================


def eval:
# ========== EDIT HERE ==========
        z = np.dot(x, self.W)
        prob = self._sigmoid(z)
        pred = (prob >= threshold).astype(int).flatten( )
# ==================================
```

(b) [10 pts] Implement functions in 'models/SoftmaxClassifier.py'. ('**forward**', '**compute_grad**', '**_softmax**', and '**eval**' respectively). Given a mini-batch data (X, Y), the error function for a mini-batch is defined as follows:

$$E(w) = -\frac{1}{|\mathcal{B}|}\sum_{(x_i,y_i)\in\mathcal{B}} y_i * \log(\hat{y}_i),$$

$$\text{where } \hat{y}_i = \text{softmax}\left(w^T x_i\right), |\mathcal{B}| \text{ is the number of the mini} - \text{batch samples}.$$

Fill in your code here. Please submit your code to i-campus.

Answer:

```
def forward:
# ========== EDIT HERE ==========
    logits = np.dot(x, self.W)
    prob = self._softmax(logits)
    y_onehot = np.zeros_like(prob)
    y_onehot = [np.arange(num_data), y] = 1
    eps = 1e – 10
    softmax_loss = -np.sum(y_onehot * np.log(prob + eps)) / num_data
# ================================


def compute_grad:
# ========== EDIT HERE ==========
    yy = np.zeros(num_classes)
    yy[y[j]] = 1
    pp = prob[j, :]
    grad_weight += np.outer(xx, (pp – yy))
# ================================


def _softmax:
# ========== EDIT HERE ==========
    x_shifted = x – np.max(x, axis – 1, keepdims = True)
    exp_x = np.exp(x_shifted)
    softmax = exp_x / np.sum(exp_x, axis = 1, keepdims = True)
# ================================


def eval:
# ========== EDIT HERE ==========
    logits = np.dot(x, self.W)
```

```
        prob = self._softmax(logits)

        pred = np.argmax(prob, axis = 1)

# ================================
```

Instructions: For problems 2, 3, and 4, we have prepared two classification datasets: Banknote Authentication and Iris. The **Banknote authentication dataset is used for binary classification**. It consists of **four features, such as variance, skewness, kurtosis, and entropy,** to predict authentication for Banknotes. On the other hand, the **Iris dataset is used for multi-class classification**, representing three Iris-types. It consists of **four features, such as sepal length and width, and petal length and width.** For detailed information on each dataset, please refer to the link below.

(Default hyperparameter settings for (b) and (c): Epoch = 100, Batch_size = 256, learning_rate = 0.1)

Dataset	# training data	# test data	# classes	Details
Banknote	1,029	343	2	link
Iris	125	25	3	link

2. **[Normalization] Z-score normalization** transforms data by subtracting the mean and dividing by the standard deviation:

$$Z = \frac{(X - \mu)}{\sigma}$$

Min-max normalization is a technique to rescale data by transforming it into a specified range, typically between 0 and 1, by subtracting the minimum value and dividing by the range of the data.

$$x_{norm} = \frac{x - X_{min}}{X_{max} - X_{min}}$$

(a) **[5 pts]** Implement the **Z-score Normalization and Min-Max Normalization** in **utils.py (z_score, min_max function)**

Fill in your code here. Please submit your code to i-campus.

Answer:

```
def z_score:
# ========== EDIT HERE ==========
    x_col = (x_col – x_col.mean( )) / x_col.std( )
# ================================


def min_max:
# ========== EDIT HERE ==========
    x_col = (x_col – x_col.min( )) / (x_col.max( ) – x_col.min( ))
# ================================
```

(b) **[15 pts]** For the Banknote and Iris datasets, compare the performance before and after normalization in default settings. Analyze why the performance differs as follows.

Run 2_LogisticRegression_main.py. and 2_SoftmaxClassifier_main.py

Banknote	Train	Test
Unnormalized	0.9825	0.9737
Z-Score Norm	0.9640	0.9708
Min-Max Norm	0.8327	0.8363

Iris	Train	Test
Unnormalized	0.7339	0.6667
Z-Score Norm	0.9032	0.8333
Min-Max Norm	0.7258	0.7917

Answer:

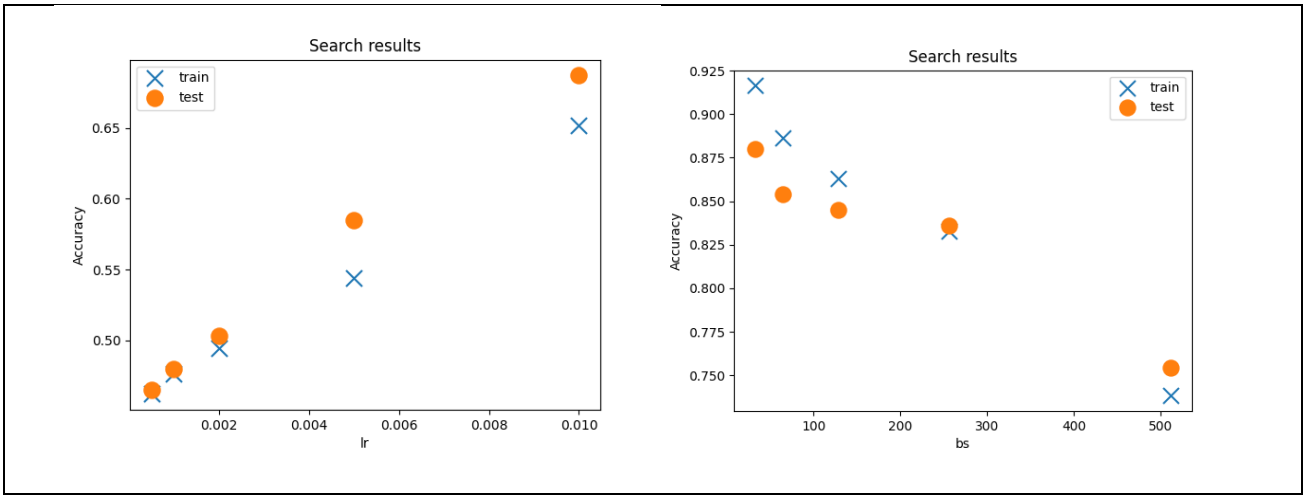
Banknote 데이터셋의 경우, train data와 test data모두 정규화하지 않은 상태에서 가장 높은 정확도를 보였다. 이는 데이터의 각 feature가 이미 적절한 스케일로 분포되어 있어 정규화 없이도 좋은 학습 결과를 보였음을 의미한다. Z-score 정규화를 적용한 경우에도 큰 성능 저하 없이, 높은 정확도를 보였다. 이는 Z-score가 스케일의 균형을 유지하면서도 데이터의 분포를 크게 왜곡하지 않았기 때문이다. 반면, Min-Max 정규화를 적용한 경우에는 성능이 크게 저하되었다. 이는 각 feature를 0~1 사이로 조정하는 과정에서 원래의 분포가 훼손되어 모델 학습에 방해되었다고 해석할 수 있다.

Iris 데이터셋의 경우, 정규화를 적용하지 않았을 때, 가장 낮은 정확도를 보였다. 이는 feature 간 스케일 차이로 모델이 class의 경계를 제대로 학습하지 못했기 때문이다. 반면, Z-score 정규화를 적용한 경우, train data와 test data 모두 가장 높은 정확도를 높였다. 이는 각 feature의 분포를 조정하면서 class 간 거리를 효과적으로 반영할 수 있었기 때문이다. Min-Max 정규화를 적용한 경우에는 test data의 정확도를 향상시켰지만, Z-score만큼 안정적인 성능을 보이지는 못했다. 이는 Min-Max 정규화가 이상치에 민감하며, class 간 상대적 거리를 왜곡할 가능성이 있기 때문이라고 해석할 수 있다.

3. **[Hyperparameter]** Draw the plot by adjusting the **learning rate and the batch size**. The x-axis represents the values of the hyperparameters being searched, and the y-axis represents the accuracy score. Try various values (i.e., 0.0005, 0.001, 0.002, 0.005, 0.01) for the learning rate, and (i.e., 32, 64, 128, 256, 512) for batch size. Analyze each of the two results and determine which parameter has the most significant impact on performance

(a) [10 pts] For the **Banknote dataset**, draw the plot by adjusting the parameters. (Fix the other hyperparameters as default settings.) **Run 2_LogisticRegression_main.py.**

Answer:



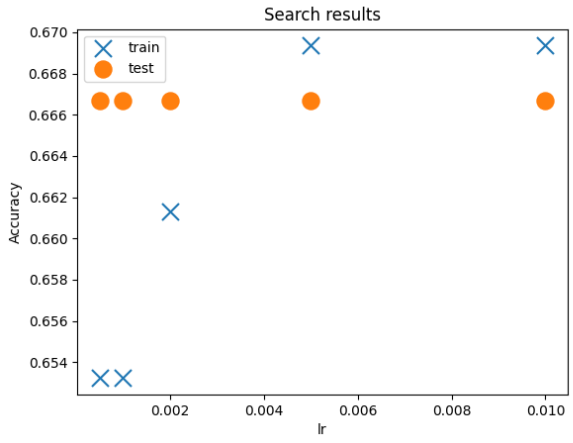
Learning rate를 0.0005, 0.001, 0.002, 0.005, 0.01로 실험해본 결과, train data와 test data 모두 learning rate가 커질수록 정확도가 증가하는 경향을 보였다. 특히, learning rate가 0.01일 때, 가장 높은 정확도를 보였다. 이는, learning rate가 너무 작으면 모델의 학습 속도가 느려서 충분히 수렴하지 못하기 때문이라고 해석할 수 있다.

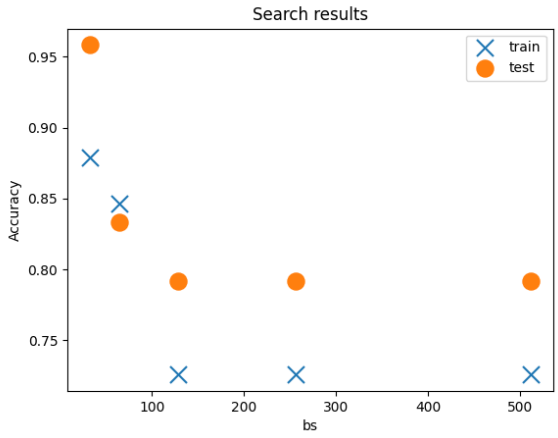
Batch size를 32, 64, 128, 256, 512로 실험해본 결과, train data와 test data 모두 batch size가 커질수록 정확도가 감소하는 경향을 보였다. 특히, batch size가 512일 때, 가장 낮은 정확도를 보였고, batch size가 32일 때, 가장 높은 정확도를 보였다. 이는 batch size가 작을수록 모델이 더 자주 업데이트되며 성능 향상으로 이어지기 때문이라고 해석할 수 있다.

Learning rate를 변화시키며 정확도를 확인했을 때, 가장 높은 정확도와 가장 낮은 정확도의 차이는 약 0.2정도이다. Batch size를 변화시키며 정확도를 확인했을 때, 가장 높은 정확도와 가장 낮은 정확도의 차이는 약 0.125정도이다. 따라서, learning rate가 batch size보다 모델의 성능에 더 큰 영향을 미친다고 볼 수 있다.

(b) [10 pts] For the **Iris dataset**, draw the plot by adjusting the parameters. (Fix the other hyperparameters as default settings.) **Run 2_SoftmaxClassifier_main.py**

Answer:





Learning rate를 0.0005, 0.001, 0.002, 0.005, 0.01로 실험해본 결과, train data는 learning rate가 커질수록 정확도가 증가하는 경향을 보였지만, test data는 learning rate와 관계없이 정확도가 일정하게 유지되는 경향을 보였다. 이는, learning rate가 커지더라도 overfitting이나 급격한 변동 없이 안정적으로 학습이 진행되기 때문이라고 해석할 수 있다.

Batch size를 32, 64, 128, 256, 512로 실험해본 결과, train data와 test data 모두 batch size가 커질수

록 정확도가 감소하는 경향을 보였다. 특히, batch size가 128 이상일 때, train data와 test data 모두에서 정확도가 낮아졌다. 이는, batch size가 작을수록 모델이 더 자주 업데이트되며 성능 향상으로 이어지기 때문이라고 해석할 수 있다.

따라서, batch size가 learning rate보다 모델의 성능에 더 큰 영향을 미친다고 볼 수 있다.

4. **[40 pts] (Kaggle challenge)** Perform a Binary Classification Task using the given **imbalanced "Back Order Prediction" Dataset**. You can improve performance by modifying the data features, and you can further enhance it through a more effective model. You need to sign up for Kaggle, the site where you can find various competitions. Observe the site below and follow the instructions. You should submit your answer on the site.

Kaggle Site: https://www.kaggle.com/t/23ea2f302e5245ae950cd183b9a6c8fa

Note that you are free to use additional libraries. Feel free to edit any part of the code while working on this task.

You can run "baseline.ipynb" or rewrite the code and report the results.

Please explain your method and provide the final accuracy.

Answer:

1. **Feature engineering**

   A. **'sku' 변수 제거**

   Sku는 제품의 고유 식별자로, 예측에 직접적인 영향을 주지 않을 것이라고 판단하여 제거하였다.

   B. **파생변수 'inv_to_sales_1m' 추가**

   재고 대비 수요를 파악하기 위해 현재 재고를 최근 1개월 판매량으로 나눈 값으로 파생변수 'inv_to_sales_1m'를 생성하였다. 이를 통해 재고가 충분한지 파악할 수 있다.

   C. **파생변수 'inv_to_forecast_3m' 추가**

   재고 대비 수요를 예측하기 위해 현재 재고를 향후 3개월 예측 수요로 나눈 값으로 파생변수 'inv_to_forecast_3m'를 생성하였다. 이를 통해 재고 부족 가능성을 파악할 수 있다.

   D. **파생변수 'sales_ratio_1_3', 'sales_ratio_3_6', 'sales_ratio_1_9' 추가**

   판매 추세의 증가 도는 감소를 파악하기 위해 과거 1, 3, 6, 9개월 간의 판매량을 비율로 나타내었다. 과거 1개월 간의 판매량을 과거 3개월 간의 판매량으로 나눈 값, 과거 3개월 간의 판매량을 과거 6개월 간의 판매량으로 나눈 값, 과거 1개월 간의 판매량을 과거 9개월 간의 나눈 값으로 파생변수 'sales_ratio_1_3', 'sales_ratio_3_6', 'sales_ratio_1_9'를 생성하였다.

   E. **파생변수 'adjusted_lead_time' 추가**

   단순히 제품의 공급 속도나 공급 업체의 평균 성능을 반영하는 것이 아니라, 이 두 지표를 종합적으로 판단하여 제품 공급의 지연 가능성을 파악하기 위해 제품 공급 속도를 공급 업체의 최근 6개월 간 평균 성능으로 나눈 값으로 파생변수 'adjusted_lead_time'을 생성하였다. 이를 통해 실제 공급 실패 위험성을 파악할 수 있다.

   F. **파생변수 'log_sales_1m', 'log_inv', 'log_past_due' 추가**

   이상치의 영향을 줄이고, 스케일을 안정화하기 위해 'sales_1_month,

‘national_inv’, ‘pieces_past_due’ 변수에 로그를 적용하여 파생변수 ‘log_sales_1m’, ‘log_inv’, ‘log_past_due’를 생성하였다.

G. 파생변수 ‘forecast_ratio’ 추가

최근 수요 예측의 상승 또는 감소 여부를 파악하기 위해 향후 3개월 수요 예측 값을 향후 9개월 수요 예측 값으로 나누어 파생변수 ‘forecast_ratio’를 생성하였다. 이를 통해 단순히 향후 3개월 또는 9개월 수요 예측 값을 반영하는 것보다 수요가 급격히 증가하거나 감소하는 등의 더 동적인 리스크 상황을 반영할 수 있다.

H. 파생변수 ‘avg_perf’ 추가

공급 업체의 평균적인 성능을 파악하기 위해 공급 업체의 최근 6개월 간 성능과 최근 12개월 간 성능의 평균값을 활용하여 파생변수 ‘avg_perf’를 생성하였다. 이를 통해 공급 업체의 평균 공급 신뢰도를 반영할 수 있다.

I. 파생변수 ‘delta_perf’ 추가

공급 업체의 성능의 변화 추세를 파악하기 위해 공급 업체의 최근 6개월 간 성능과 최근 12개월 간 성능의 차이를 활용하여 파생변수 ‘delta_perf’를 생성하였다. 이를 통해 공급 업체의 성능이 향상되고 있는지, 악화되고 있는지를 파악해 모델에 반영하여, 공급망 리스크를 효율적으로 포착할 수 있다.

J. 파생변수 ‘any_risk’ 추가

위험 요소가 하나라도 있는지의 여부가 모델에 더 큰 영향을 미칠 수 있다고 판단하여 6개의 위험 요소 (‘potential_issue’, ‘deck_risk’, ‘oe_constraint’, ‘ppap_risk’, ‘stop_auto_buy’, ‘rev_stop’) 중 하나라도 위험이 감지되면 1로 판단하는 파생변수 ‘any_risk’를 생성하였다. 이를 통해 모델에 영향을 미칠 수 있는 위험성을 더 민감하게 포착할 수 있다.

K. 파생변수 ‘total_risk_flags’ 추가

위험 요소의 누적 강도를 파악하기 위해 6개의 위험 요소 (‘potential_issue’, ‘deck_risk’, ‘oe_constraint’, ‘ppap_risk’, ‘stop_auto_buy’, ‘rev_stop’)의 이진 플래그 값들을 전부 더한 값으로 파생변수 ‘total_risk_flags’를 생성하였다. 이를 통

해 위험 정도의 심각성을 더 직관적으로 반영할 수 있다.

L. 파생변수 'zero_sales_1m' 추가

최근 판매량의 정보를 더 직관적으로 파악하기 위해 최근 1개월 간의 판매량이 0이면 1로 판단하는 파생변수 'zero_sales_1m'를 생성하였다. 이를 통해 공급 문제 혹은 수요 감소 등 위험 징후를 보다 직접적으로 반영할 수 있다.

M. 파생변수 'no_forecast' 추가

짧은 미래의 수요 예측값을 더 직관적으로 파악하기 위해 향후 3개월 간 수요 예측값이 0이면 1로 판단하는 파생변수 'no_forecast'를 생성하였다. 이를 통해 수요 예측이 불확실한 상황이거나, 공급의 리스크가 예상되는 등의 위험 징후를 보다 직접적으로 반영할 수 있다.

N. 파생변수 'forecast_to_sales' 추가

수요 예측값과 실제 판매량 간의 차이를 직관적으로 파악하기 위해 향후 3개월 간 수요 예측값을 최근 3개월 간의 판매량으로 나눈 파생변수 'forecast_to_sales' 변수를 생성하였다. 이를 통해 수요 예측의 신뢰도를 효과적으로 파악할 수 있다.

분모가 0이 되어 무한대나 NaN이 발생하는 것을 방지하기 위해, 나눗셈 연산의 분모에는 소량의 값을 더해 수치 안정성을 확보하고자 했다.

2. 이상치 및 결측값 제거

비율 기반의 파생 변수들을 생성하는 과정에서 분모가 0에 가까운 경우, 'inf' 또는 '-inf'등의 무한대 값이 발생할 가능성이 있다. 이러한 이상치는 모델 학습에 치명적인 영향을 미칠 수 있기 때문에 'NaN'으로 대체하였다. 이후, 모든 결측값들은 '0'으로 대체함으로써 결측치를 제거하였다. 이 과정을 통해 수치적 안정성을 확보하고자 했다.

3. 데이터 스케일 조정

모델 학습의 안정성과 성능 향상을 위해 파생변수를 포함한 모든 feature들에 대해서 스케일링을 진행하였다. 스케일링은 StandardScaler를 활용하여 각 feature를 평

균 0, 표준편차 1의 분포로 변환하였다.

4. Oversampling 진행

주어진 데이터가 negative:positive = 9:1 정도로 클래스 불균형이 매우 심해 주어진 데이터만으로 학습을 진행하면 negative에 치우친 예측을 하게 될 것이라고 판단했다. 이에, oversampling과 undersampling을 진행하여 클래스의 비율을 균등하게 조정하여 실험을 진행하였다. Oversampling에는 SMOTE를, undersampling에는 Random Undersampling을 적용하였다. Random Undersampling 기법을 적용하였을 때에는, 전체적으로 데이터의 크기가 작아져 모델의 성능이 낮아지는 결과를 보였다. 반면, SMOTE를 적용하였을 때에는, positive의 recall값과 f1-score가 크게 개선되며 전체적으로 모델 성능이 향상되었다. 이에, SMOTE를 적용하여 클래스 불균형 문제를 해결하고 분포가 균등한 데이터셋을 학습에 사용하였다.

5. 모델링

새롭게 생성한 파생 변수의 개수가 많아, 모델이 불필요한 feature에 의존하는 것을 방지하기 위해 L1 정규화를 적용하였다. 최적화 알고리즘으로는 L1 정규화와 함께 사용할 수 있는 'liblinear' solver와 'saga' solver를 사용하여 실험을 진행하였다. 그 결과, 'liblinear' solver의 성능이 더 좋았다. 이는, 'liblinear' solver는 소규모 데이터셋에 적합하고, 'saga' solver는 대규모 데이터셋에 적합한 기법이기 때문이라고 해석하였고, 최종 모델링에는 'liblinear' solver를 사용하였다. max_iter 값은 5000으로 설정하여 수렴에 실패하는 현상을 방지하고자 했다.

6. Best threshold 탐색

확률 기반 예측 결과에 다양한 threshold를 적용하여 실험을 진행함으로써 f1-score를 최대화하는 최적 threshold를 찾고자 하였다. 0.3부터 0.7까지 총 41개의 threshold로 실험을 진행하였고, threshold가 0.51일 때 가장 높은 성능을 보였다. 이는 oversampling과 class weighting 과정에서 균형이 잘 맞추어졌음을 시사한다고 볼 수 있다.

7. Final accuracy

최종적으로 학습을 진행하고, 성능을 평가했을 때, train data의 F1 score는 0.8718, ROC AUC 값은 0.9304, accuracy는 0.8701이였다. 상세한 classification report는 아래

표와 같다.

	precision	recall	F1-score	Support
0.0	0.88	0.86	0.87	90000
1.0	0.86	0.88	0.87	90000
Accuracy			0.87	180000
Macro avg	0.87	0.87	0.87	180000
Weighted avg	0.87	0.87	0.87	180000

Table 1 test data의 classification report

Test data의 accuracy는 (public score 기준) 0.88이였다.