بسمه تعالی

موضوع: گزارش مینی پروژه اول (ماشین لرنینگ)

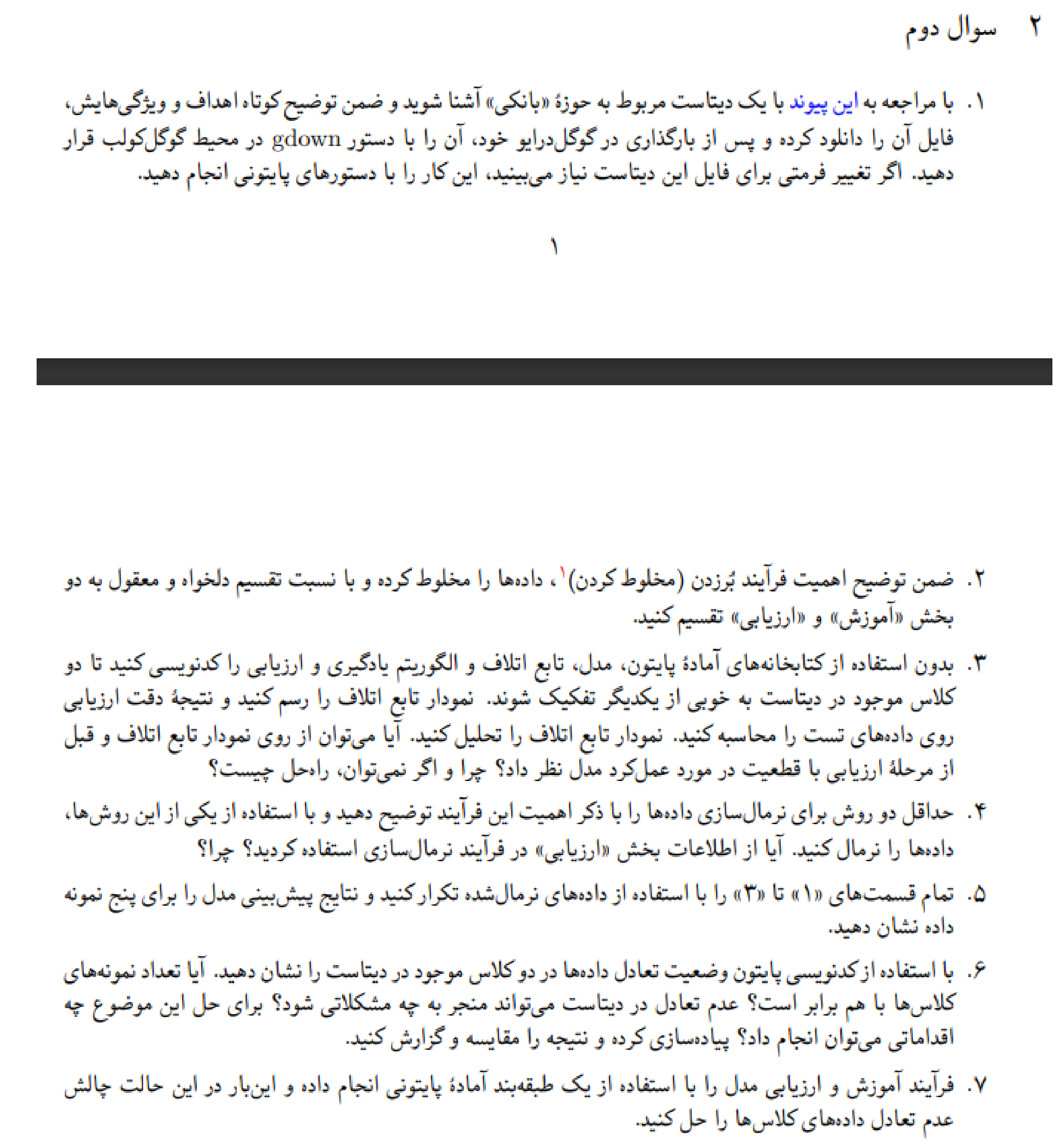
نام و نام خانوادگی: محمدعرفان مومنی نسب

شماره دانشجویی: ٤٠٢١٦٣١٢١١

نام دانشگاه: دانشگاه جامع انقلاب اسلامی

نام استاد درس: جناب آقای دکتر علیاری

نام استاد یار درس: جناب آقای مهندس محمدجواد احمدی



۲ سوال دوم

۱. با مراجعه به این پیوند با یک دیتاست مربوط به حوزهٔ «بانکی» آشنا شوید و ضمن توضیح کوتاه اهداف و ویژگی‌هایش، فایل آن را دانلود کرده و پس از بارگذاری در گوگل‌درایو خود، آن را با دستور gdown در محیط گوگل‌کولب قرار دهید. اگر تغییر فرمتی برای فایل این دیتاست نیاز می‌بینید، این کار را با دستورهای پایتونی انجام دهید.

۱

۲. ضمن توضیح اهمیت فرآیند بُرزدن (مخلوط کردن)[۱]، داده‌ها را مخلوط کرده و با نسبت تقسیم دلخواه و معقول به دو بخش «آموزش» و «ارزیابی» تقسیم کنید.

۳. بدون استفاده از کتابخانه‌های آمادهٔ پایتون، مدل، تابع اتلاف و الگوریتم یادگیری و ارزیابی را کدنویسی کنید تا دو کلاس موجود در دیتاست به خوبی از یکدیگر تفکیک شوند. نمودار تابع اتلاف را رسم کنید و نتیجهٔ دقت ارزیابی روی داده‌های تست را محاسبه کنید. نمودار تابع اتلاف را تحلیل کنید. آیا می‌توان از روی نمودار تابع اتلاف و قبل از مرحلهٔ ارزیابی با قطعیت در مورد عملکرد مدل نظر داد؟ چرا و اگر نمی‌توان، راه‌حل چیست؟

۴. حداقل دو روش برای نرمال‌سازی داده‌ها را با ذکر اهمیت این فرآیند توضیح دهید و با استفاده از یکی از این روش‌ها، داده‌ها را نرمال کنید. آیا از اطلاعات بخش «ارزیابی» در فرآیند نرمال‌سازی استفاده کردید؟ چرا؟

۵. تمام قسمت‌های «۱» تا «۳» را با استفاده از داده‌های نرمال‌شده تکرار کنید و نتایج پیش‌بینی مدل را برای پنج نمونه داده نشان دهید.

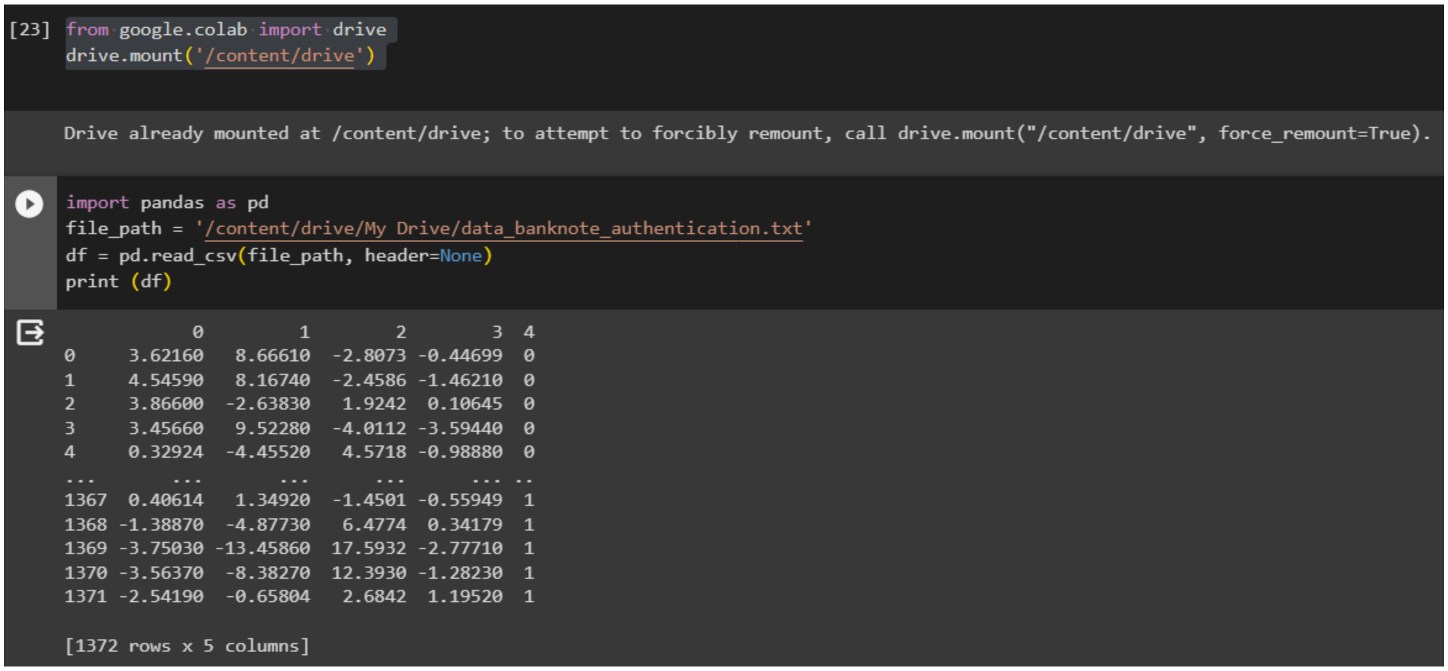
۶. با استفاده از کدنویسی پایتون وضعیت تعادل داده‌ها در دو کلاس موجود در دیتاست را نشان دهید. آیا تعداد نمونه‌های کلاس‌ها با هم برابر است؟ عدم تعادل در دیتاست می‌تواند منجر به چه مشکلاتی شود؟ برای حل این موضوع چه اقداماتی می‌توان انجام داد؟ پیاده‌سازی کرده و نتیجه را مقایسه و گزارش کنید.

۷. فرآیند آموزش و ارزیابی مدل را با استفاده از یک طبقه‌بند آمادهٔ پایتونی انجام داده و این‌بار در این حالت چالش عدم تعادل داده‌های کلاس‌ها را حل کنید.

Part 1: Dataset Acquisition

- **Task**: Acquire the Banknote Authentication dataset[1] from the UCI Machine Learning Repository and load it into a Python environment.

- **Implementation**: The dataset was downloaded, uploaded to Google Drive, and loaded into a Google Colab environment using pandas.

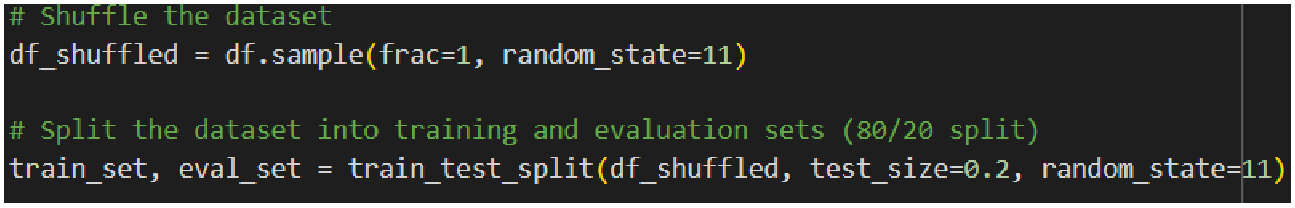


```
[23] from google.colab import drive
     drive.mount('/content/drive')

     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

     import pandas as pd
     file_path = '/content/drive/My Drive/data_banknote_authentication.txt'
     df = pd.read_csv(file_path, header=None)
     print (df)

                0         1        2        3  4
     0     3.62160   8.66610  -2.8073 -0.44699  0
     1     4.54590   8.16740  -2.4586 -1.46210  0
     2     3.86600  -2.63830   1.9242  0.10645  0
     3     3.45660   9.52280  -4.0112 -3.59440  0
     4     0.32924  -4.45520   4.5718 -0.98880  0
     ...       ...       ...      ...      ... ..
     1367  0.40614   1.34920  -1.4501 -0.55949  1
     1368 -1.38870  -4.87730   6.4774  0.34179  1
     1369 -3.75030 -13.45860  17.5932 -2.77710  1
     1370 -3.56370  -8.38270  12.3930 -1.28230  1
     1371 -2.54190  -0.65804   2.6842  1.19520  1

     [1372 rows x 5 columns]
```

Part 2: Data Shuffling[2] and Splitting

- **Importance of Shuffling**: Shuffling prevents order bias and ensures a diverse and representative distribution of data across the training and evaluation sets.

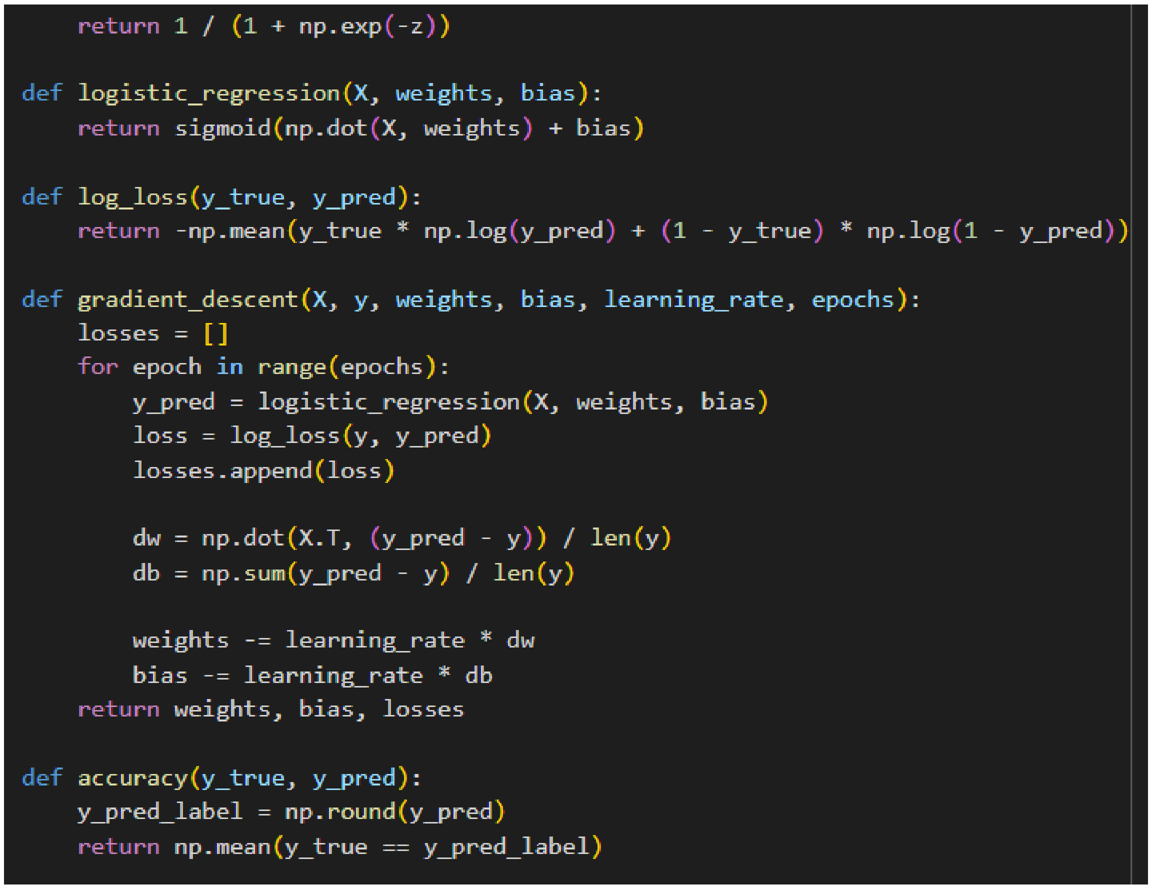
- **Data Splitting**: The dataset was shuffled and split into training (80%) and evaluation (20%) sets using `train_test_split` from scikit-learn.



```
# Shuffle the dataset
df_shuffled = df.sample(frac=1, random_state=11)

# Split the dataset into training and evaluation sets (80/20 split)
train_set, eval_set = train_test_split(df_shuffled, test_size=0.2, random_state=11)
```

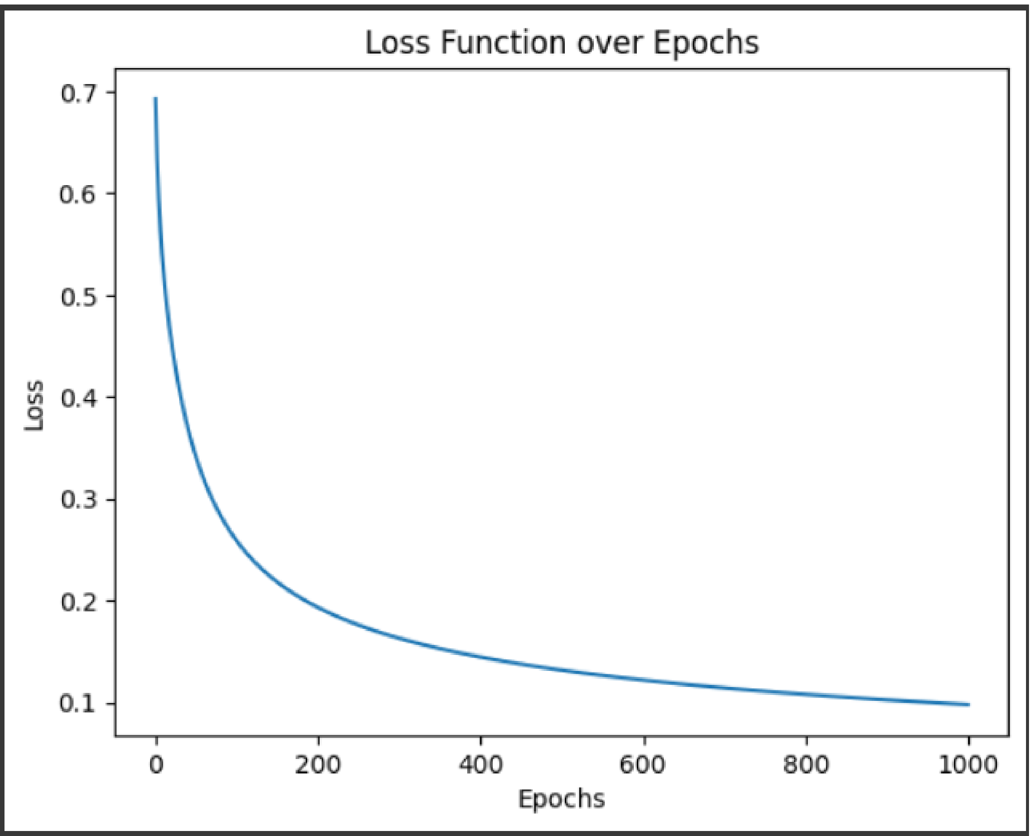
Part 3: Model, Loss Function, and Learning Algorithm

- **Custom Implementation**: A logistic regression model, binary cross-entropy loss function, and gradient descent algorithm were coded from scratch.



```
    return 1 / (1 + np.exp(-z))

def logistic_regression(X, weights, bias):
    return sigmoid(np.dot(X, weights) + bias)

def log_loss(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

def gradient_descent(X, y, weights, bias, learning_rate, epochs):
    losses = []
    for epoch in range(epochs):
        y_pred = logistic_regression(X, weights, bias)
        loss = log_loss(y, y_pred)
        losses.append(loss)

        dw = np.dot(X.T, (y_pred - y)) / len(y)
        db = np.sum(y_pred - y) / len(y)

        weights -= learning_rate * dw
        bias -= learning_rate * db
    return weights, bias, losses

def accuracy(y_true, y_pred):
    y_pred_label = np.round(y_pred)
    return np.mean(y_true == y_pred_label)
```

- **Graph and Analysis**[3]: The loss function graph was plotted to analyze the learning process. It was noted that while the loss graph indicates learning effectiveness, it doesn't directly translate to performance on unseen data.



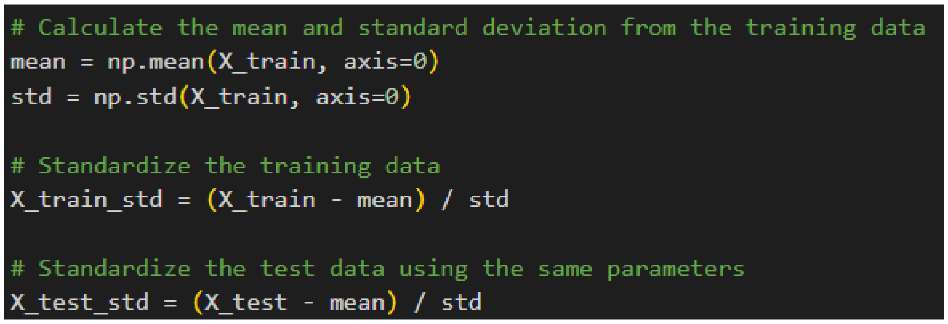
```
Accuracy on Test Data: 0.96
```

Part 4: Data Normalization[4]

- **Importance**: Normalization ensures uniform scale, faster convergence, and numerical stability.

- **Methods**: Discussed min-max scaling and standardization (Z-score normalization). Standardization was chosen for its robustness.

- **Implementation**: Data was standardized using the mean and standard deviation of the training set. The test set was standardized using the same parameters to avoid data leakage.



```
# Calculate the mean and standard deviation from the training data
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)

# Standardize the training data
X_train_std = (X_train - mean) / std

# Standardize the test data using the same parameters
X_test_std = (X_test - mean) / std
```

Part 5: Repeating with Normalized Data

- **Normalization Impact**: The logistic regression model was retrained using normalized data. It demonstrated the impact of normalization on model training and performance.

- **Model Predictions**: Predictions were made for five samples from the normalized test set, showcasing the model's outputs.

Part 6: Data Balance Check and Resolution

- **Balance Check**: Analyzed the dataset for class balance and found disparities in class distribution.

- **Problems and Solutions**: Discussed problems like model bias due to imbalance and solutions such as resampling and altered class weights.

- **Implementation**: Applied solutions like class weight adjustment and resampling (e.g., SMOTE) to address imbalance.

Part 7: Using Pre-built Classifiers with Imbalance Handling

- **Classifier Utilization**: Employed `scikit-learn`'s Logistic Regression with strategies to handle class imbalance.

- **Strategies**: Implemented class weight adjustment and SMOTE resampling.

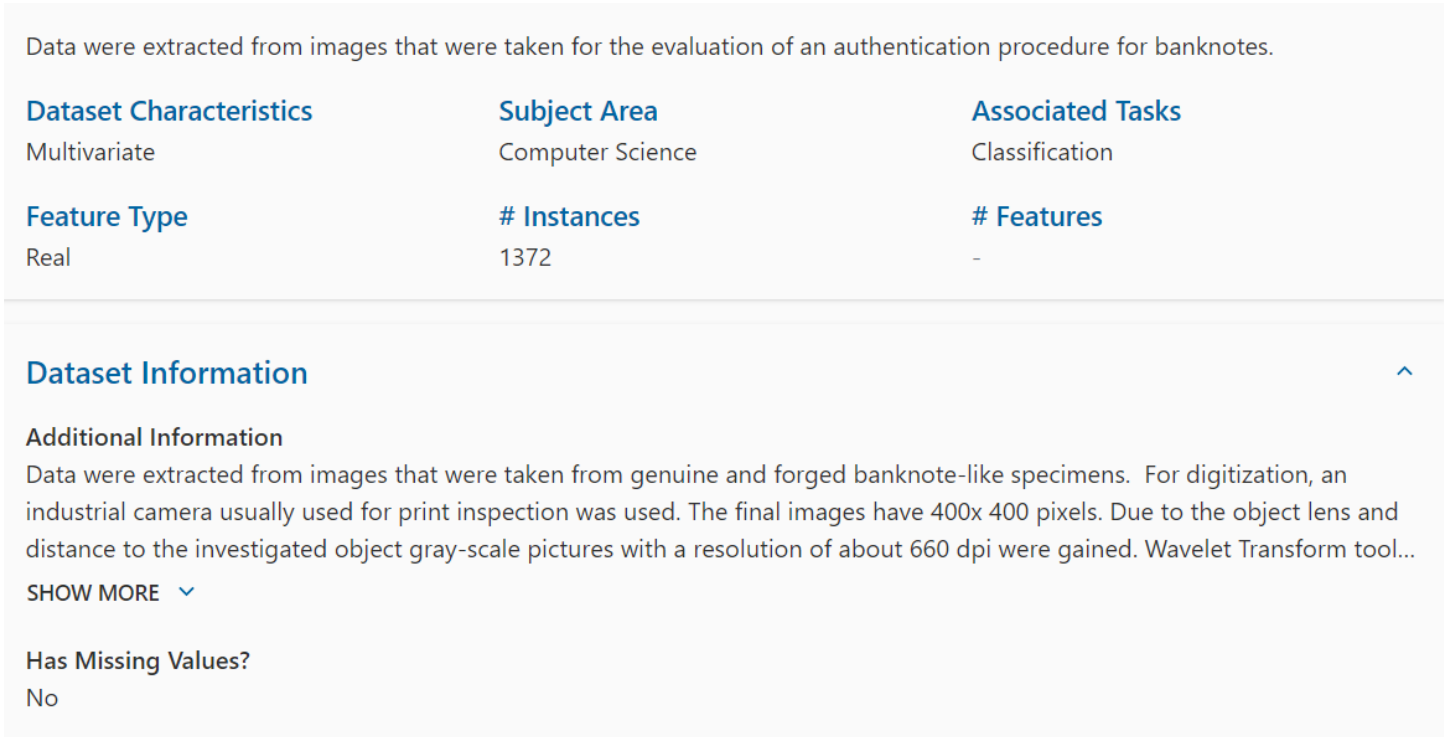
- **Evaluation**: The model was trained and evaluated with each strategy, demonstrating their effectiveness in handling class imbalance.

Conclusion

This project comprehensively covered dataset loading, preprocessing (including shuffling and normalization), custom model implementation, and handling class imbalance. It highlighted the importance of data preprocessing, custom model coding, and strategies for dealing with class imbalance, providing valuable insights into the practical aspects of machine learning.

---

(1): [banknote authentication - UCI Machine Learning Repository]



Data were extracted from images that were taken for the evaluation of an authentication procedure for banknotes.

Dataset Characteristics	Subject Area	Associated Tasks
Multivariate	Computer Science	Classification
Feature Type	**# Instances**	**# Features**
Real	1372	-

Dataset Information

Additional Information

Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400x 400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool...

SHOW MORE

Has Missing Values?

No

(2): Importance of Data Shuffling

1. Prevents Order Bias: Data in real-world datasets might come ordered in some way (e.g., chronologically, by class label). If not shuffled, the model might learn patterns specific to the order rather than the actual data features.

2. Ensures Diversity in Training and Testing: Shuffling helps in distributing all types of data across the training and evaluation sets. This diversity ensures that both sets are representative of the overall distribution.

3. Improves Generalization: By training on a well-mixed dataset, the model is less likely to overfit to specific sequences or patterns present in the unshuffled data, leading to better generalization to new data.

(3): Analyzing the graph of the loss function during the training of a logistic regression model can provide valuable insights into the learning process, but it does not necessarily allow for definite conclusions about the model's performance on unseen data. Here's a detailed analysis:

Analyzing the Loss Function Graph

1. Decreasing Loss: A continuously decreasing loss curve indicates that the model is learning effectively. The weights are being adjusted in the right direction to minimize the loss.

2. Flattening of the Curve: As the curve begins to flatten, it suggests that the model is approaching the minimum loss, and further training might not yield significant improvements.

3. Fluctuations in Loss: If the loss fluctuates or increases, it could indicate issues such as a high learning rate or a poorly specified model.

Predicting Model Performance from the Loss Graph

- General Trends: While the loss graph shows how well the model is fitting the training data, it does not directly translate to how well the model will perform on unseen data (test data).

- Overfitting Concerns: Even with a continuously decreasing loss, the model might be overfitting to the training data, capturing noise rather than the underlying pattern. This overfitting would result in poor performance on the test data.

Why the Loss Graph Might Not Be Enough

- Training Data Limitation: The loss function is calculated on the training data, so it only reflects the model's performance on this data.

- Unseen Data Performance: The true test of a model's performance is how well it generalizes to new, unseen data.

Solution for a Comprehensive Evaluation

- Validation Set: Use a separate validation set during training to monitor the model's performance on unseen data. This approach helps in detecting overfitting.

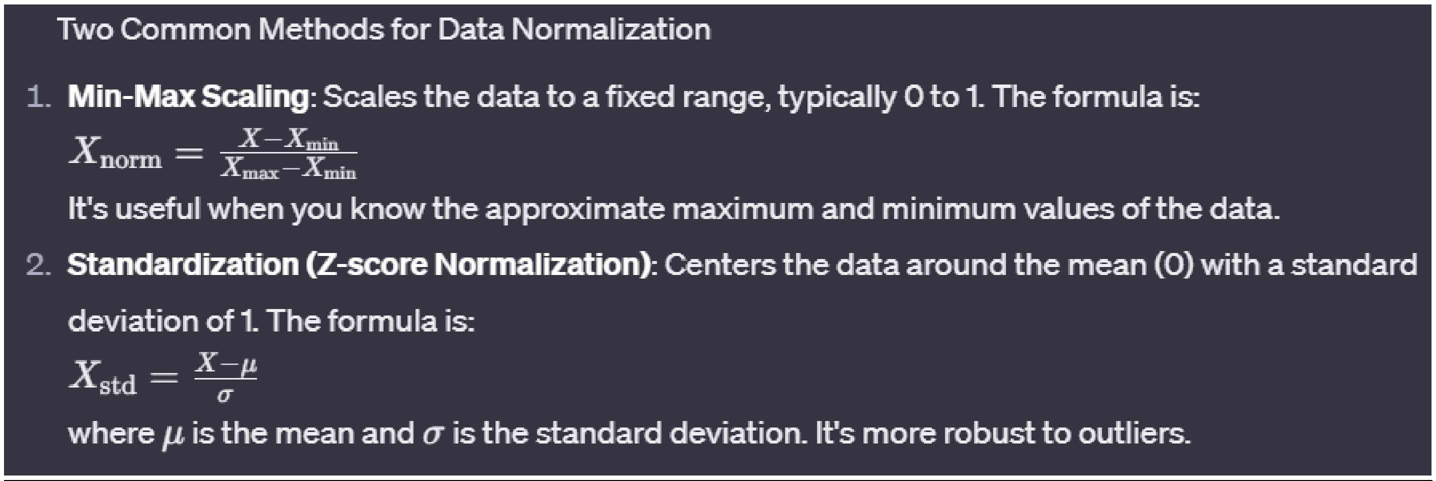
- Early Stopping: Implement early stopping based on the validation loss. If the validation loss stops decreasing or starts to increase, it's a sign to stop training to avoid overfitting.

- Cross-Validation: Use cross-validation to assess the model's generalization capability. It provides a more robust estimate of the model's performance on new data.

<u>In summary, while the loss function graph is a useful tool for understanding the learning process and diagnosing issues with model training, it does not provide a complete picture of the model's performance. Evaluating the model on a -</u>

separate validation set and using techniques like cross-validation are essential for a more accurate assessment of the model's ability to generalize to new data.

(4):



Two Common Methods for Data Normalization

1. **Min-Max Scaling**: Scales the data to a fixed range, typically 0 to 1. The formula is:
   $X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$
   It's useful when you know the approximate maximum and minimum values of the data.
2. **Standardization (Z-score Normalization)**: Centers the data around the mean (0) with a standard deviation of 1. The formula is:
   $X_{\text{std}} = \frac{X - \mu}{\sigma}$
   where μ is the mean and σ is the standard deviation. It's more robust to outliers.