EN3150 Assignment 02: Learning from data and related challenges and classification

Sampath K. Perera

September 15, 2023

1 Logistic regression weight update process

1. Use the code given in listing 1 to generate data.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
# Generate synthetic data
np.random.seed(0)
centers = [[-5, 0], [0, 1.5]]
X, y = make_blobs(n_samples=1000, centers=centers,
    random_state=40)
transformation = [[0.4, 0.2], [-0.4, 1.2]]
X = np.dot(X, transformation)
# Add a bias term to the feature matrix
X = np.c_[np.ones((X.shape[0], 1)), X]
# Initialize coefficients
W = np.zeros(X.shape[1])
# Define the logistic sigmoid function
def sigmoid(z):
return 1 / (1 + np.exp(-z))
# Define the logistic loss (binary cross-entropy)
    function
def log_loss(y_true, y_pred):
epsilon = 1e-15
y_pred = np.clip(y_pred, epsilon, 1 - epsilon)  # Clip
     to avoid log(0)
return - (y_true * np.log(y_pred) + (1 - y_true) * np.
    log(1 - y_pred))
# Gradient descent and Newton method parameters
learning_rate = 0.1
iterations = 10
loss_history = []
```

Listing 1: Data generation.

2. Initializing weights as zeros, perform gradient descent based weight update for the given data. Here, use binary cross entropy as a loss function. Further, use learning rate as $\alpha = 0.1$ and number of iterations as $t = 10$. Batch Gradient descent weight update is given below,

$$\boldsymbol{w}_{(t+1)} \leftarrow \boldsymbol{w}_{(t)} - \alpha \frac{1}{N} (\text{sigm}(\boldsymbol{w}_{(t)}^T \boldsymbol{X}) - \boldsymbol{y})\boldsymbol{X}.$$

Here, $\boldsymbol{X}$ is data matrix of dimension of $N \times (D+1)$. Here, N is total number of data samples and D is number of features. Now, $\boldsymbol{X}$ is given by

$$\boldsymbol{X} = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{D,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{D,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,i} & x_{2,i} & \cdots & x_{D,i} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,N} & x_{2,N} & \cdots & x_{D,N} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_i \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}.$$

3. Plot the loss with respect to number of iterations.

4. Initializing weights as zeros, perform Newton's method weight update for the given data. Here, use binary cross entropy as a loss function. Further, set number of iterations as $t = 10$. Batch Newton's method weight update is given below

$$\boldsymbol{w}_{(t+1)} \leftarrow \boldsymbol{w}_{(t)} - \left(\frac{1}{N} \boldsymbol{X}^T \boldsymbol{S} \boldsymbol{X}\right)^{-1} \left(\frac{1}{N} (\text{sigm}(\boldsymbol{w}_{(t)}^T \boldsymbol{X}) - \boldsymbol{y})\boldsymbol{X}\right).$$

and is $\boldsymbol{S}$ given by

$$\boldsymbol{S} = \text{diag}(s_1,\ s_2, \ldots, s_N),$$
$$\boldsymbol{s_i} = \left(\text{sigm}(\boldsymbol{w}_{(t)}^T \boldsymbol{x}_i) - \boldsymbol{y}_i)\right)\left(1 - \text{sigm}(\boldsymbol{w}_{(t)}^T \boldsymbol{x}_i) - \boldsymbol{y}_i)\right).$$

5. Plot the loss with respect to number of iterations.

6. Plot the loss with respect to number of iterations for both Gradient descent and Newton method's in a single plot. Comment on your results.

2 Perform grid search for hyper-parameter tuning

1. Use the code given in listing 2 to load data.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import GridSearchCV,
    train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.utils import check_random_state
# data loading

train_samples = 500
X, y = fetch_openml("mnist_784", version=1, return_X_y=True,
    as_frame=False)

random_state = check_random_state(0)
permutation = random_state.permutation(X.shape[0])
X = X[permutation]
y = y[permutation]
X = X.reshape((X.shape[0], -1))
X_train, X_test, y_train, y_test = train_test_split(X, y,
    train_size=train_samples, test_size=100)
```

Listing 2: Data loading.

2. Explain the purpose of "X = X[permutation]" and " y = y[permutation]".

3. Use lasso logistic regression for image classification as "LogisticRegression(penalty='l1', solver='liblinear', multi_class='auto')". Next, create a pipeline that includes the scaling, the Lasso logistic regression estimator, and a parameter grid for hyperparameter tuning (C value).[Hint refer url]

4. Use GridSearchCV to perform a grid search over the range (e.g., np.logspace(-2, 2, 9)) of to find optimal value of hyperparameter C [Hint refer url]

5. Plot the classification accuracy with respect to hyperparameter C. Comment on your results.

6. Calculate confusion matrix, precision, recall and F1-score. Comment on your results.

3 Logistic regression

Based on "James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer."

1. Consider a dataset collected from a statistics class that includes information on students. The dataset includes variables x_1 representing the number of hours studied, x_2 representing the undergraduate GPA, and y indicating whether the student received an A^+ in the class. After conducting a logistic regression analysis, we obtained the following estimated coefficients: $w_0 = -6$, $w_1 = 0.05$, and $w_2 = 1$.

(a) What is the estimated probability that a student, who has studied for 40 hours and has an undergraduate GPA of 3.5, will receive an A^+ in the class?

(b) To achieve a 50% chance of receiving an A^+ in the class, how many hours of study does a student like the one in part (1a) need to complete?

Submission

- Upload a report and your codes as a zip file named as "EN3150_your_indexno_A02.zip". Include the index number and the name within the report as well.
- The interpretation of results and the discussion are important in the report.
- An extra penalty of 10% is applied for late submission.
- Plagiarism will be checked and in cases of plagiarism, an extra penalty of 10% will be applied.