CSCE 421: Machine Learning (Spring 2025)

Homework #2

Due 2/18/2025, 11:59 pm

1. You need to submit (1) a report in PDF and (2) your code files, both to Canvas.
2. Your PDF report should include (1) answers to the non-programming part, and (2) results and analysis of the programming part. For the programming part, your PDF report should at least include the results you obtained, for example the accuracy, training curves, parameters, etc. You should also analyze your results as needed.
3. Please name your PDF report "HW#_FirstName_LastName.pdf". Please put all code files into a compressed file named "HW#_FirstName_LastName.zip". Please submit two files (.pdf and .zip) to Canvas (i.e., do not include the PDF file into the ZIP file).
4. Only write your code between the following lines. Do not modify other parts.

   ### YOUR CODE HERE

   ### END YOUR CODE
5. LFD refers to the textbook "Learning from Data".
6. All students are highly encouraged to typeset their reports using Word or LaTeX. In case you decide to hand-write, please make sure your answers are clearly readable in scanned PDF.
7. Unlimited number of submissions are allowed and the latest one will be timed and graded. If you make a resubmission after the due date, it will be considered late.
8. Please read and follow submission instructions. No exception will be made to accommodate incorrectly submitted files/reports.
9. Please start your submission to Canvas at least 15-30 minutes before the deadline, as there might be latency. We do NOT accept E-mail submissions.

Linear Models for Handwritten Digits Classification: In this assignment, you will implement the binary logistic regression model and multi-class logistic regression model on a partial dataset from MNIST. In this classification task, the model will take a 16×16 image of handwritten digits as inputs and classify the image into different classes. For the binary case, the classes are 1 and 2 while for the multi-class case, the classes are 0, 1, and 2. The "data" fold contains the dataset which has already been split into a training set and a testing set. All data examples are saved in dictionary-like objects using "npz" file. For each data sample, the dictionary key 'x' indicates its raw features, which are represented by a 256-dimensional vector where the values between $[-1, 1]$ indicate grayscale pixel values for a 16×16 image. In addition, the key 'y' is the label for a data example, which can be 0, 1, or 2. The "code" fold provides the starting code. You must implement the models using the starting code.

1. (15 points) Data Preprocessing: In this problem, you need to finish "code/DataReader.py".

   (a) Explain what the function *train_valid_split* does and why we need this step.

(b) Before testing, is it correct to re-train the model on the whole training set? Explain your answer.

(c) In this assignment, we use two hand-crafted features:
The first feature is a measure of symmetry. For a 16×16 image x, it is defined as

$$F_{symmetry} = -\frac{\sum_{pixel} |x - flip(x)|}{256},$$

where 256 is the number of pixels and $flip(\cdot)$ means left and right flipping.
The second feature is a measure of intensity. For a 16×16 image x, it is defined as

$$F_{intensity} = \frac{\sum_{pixel} x}{256},$$

which is simply the average of pixel values.
Implement them in the function *prepare_X*.

(d) In the function *prepare_X*, there is a third feature which is always 1. Explain why we need it.

(e) The function *prepare_y* is already finished. Note that the returned indices stores the indices for data from class 1 and 2. Only use these two classes for binary classification and convert the labels to +1 and -1 if necessary.

(f) Test your code in "code/main.py" and visualize the training data from class 1 and 2 by implementing the function *visualize_features*. The visualization should not include the third feature. Therefore it is a 2-D scatter plot. Include the figure in your submission.

2. (25 points) Cross-entropy loss: In logistic regression, we use the cross-entropy loss.

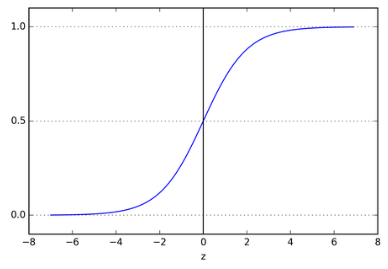
(a) Write the loss function $E(w)$ for one training data sample (x, y). Note that the binary labels are 1 and -1.

(b) Compute the gradient $\nabla E(w)$. Please provide intermediate steps of derivation.

(c) Once the optimal w is obtained, it can be used to make predictions as follows:

$$\text{Predicted class of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.5 \\ -1 & \text{if } \theta(w^T x) < 0.5 \end{cases}$$

where the function $\theta(z) = \frac{1}{1+e^{-z}}$ looks like



However, this is not the most efficient way since the decision boundary is linear. Why? Explain it. When will we need to use the sigmoid function in prediction?

(d) Is the decision boundary still linear if the prediction rule is changed to the following? Justify briefly.

$$\text{Predicted label of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases}$$

(e) In light of your answers to the above two questions, what is the essential property of logistic regression that results in the linear decision boundary?

3. (25 points) Sigmoid logistic regression: In this problem, you need to finish "code/LogisticRegression.py". **Please follow the instructions in the starting code. Please use data from class 1 and 2 for the binary classification.**

   (a) Based on (b) in the last problem, implement the function *_gradient*.

   (b) There are different ways to train a logistic regression model. In this assignment, you need to implement batch gradient descent, stochastic gradient descent and mini-batch gradient descent in the functions *fit_BGD*, *fit_SGD* and *fit_miniBGD*, respectively. Note that: In batch gradient descent, each model update is based on all training samples. In stochastic gradient descent, each model update is based on one training sample. In mini-batch gradient descent, you split your training data into (roughly equal-sized) mini-batches, and each model update is based on training samples in each mini-batch. Thus, BGD and SDG are actually special cases of mini-BGD. An epoch is completed when each training sample is used to update the model exactly once. In SDG and mini-BGD, you might want to reshuffle your samples before each epoch.

   (c) Implement the functions *predict* and *score* for prediction and evaluation, respectively. Additionally, please implement the function *predict_proba* which outputs the probabilities of both classes.

   (d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results*. The visualization should include the 2-D scatter plot of training data from class 1 and 2 and your decision boundary. Include the figure in your submission.

   (e) Implement the testing process and report the test accuracy of your best logistic regression model.

4. (25 points) Softmax logistic regression: In this problem, you need to finish "code/LRM.py". **Please follow the instructions in the starting code.**

   (a) Based on the course notes, implement the function *_gradient*.

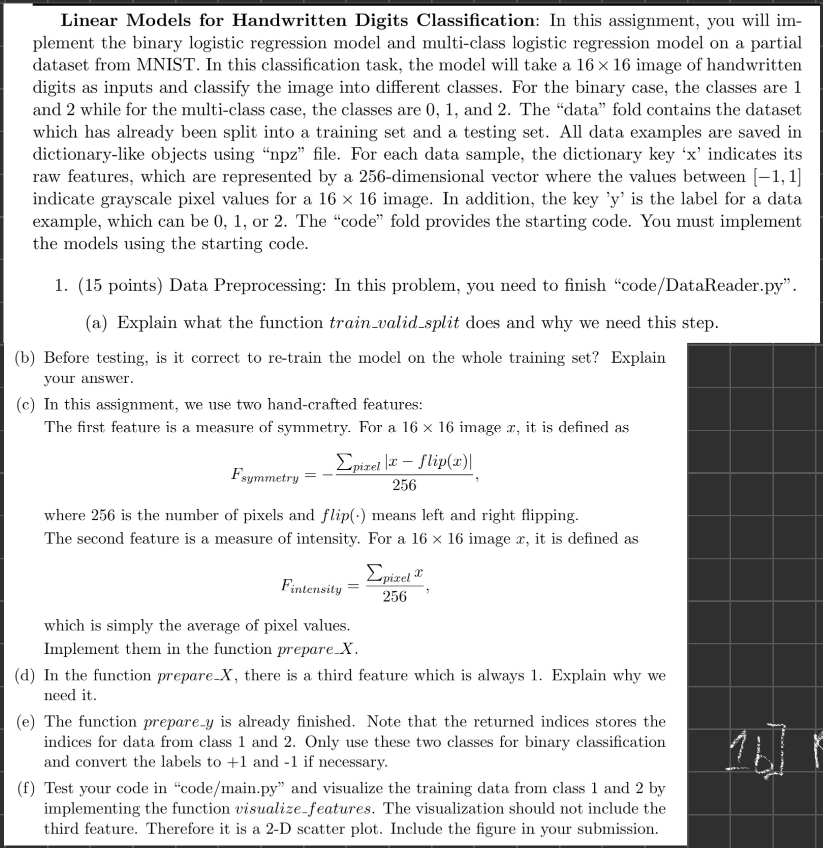
   (b) In this assignment, you only need to implement mini-batch gradient descent in the function *fit_miniBGD*.

   (c) Implement the functions *predict* and *score* for prediction and evaluation, respectively.

   (d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results_multi*. The visualization should include the 2-D scatter plot of training data from class 0, 1, and 2, and your decision boundaries. Include the figure in your submission.

   (e) Implement the testing process and report the test accuracy of your best logistic regression model.

5. (10 points) Softmax logistic vs Sigmoid logistic: In this problem, you need to experimentally compare these two methods. **Please follow the instructions in the starting code.** Use data examples from class 1 and 2 for classification. Train the softmax logistic classifier and the sigmoid logistic classifier using the same data until convergence. Compare these two classifiers and report your observations and insights.



Linear Models for Handwritten Digits Classification: In this assignment, you will implement the binary logistic regression model and multi-class logistic regression model on a partial dataset from MNIST. In this classification task, the model will take a 16×16 image of handwritten digits as inputs and classify the image into different classes. For the binary case, the classes are 1 and 2 while for the multi-class case, the classes are 0, 1, and 2. The "data" fold contains the dataset which has already been split into a training set and a testing set. All data examples are saved in dictionary-like objects using "npz" file. For each data sample, the dictionary key 'x' indicates its raw features, which are represented by a 256-dimensional vector where the values between $[-1, 1]$ indicate grayscale pixel values for a 16×16 image. In addition, the key 'y' is the label for a data example, which can be 0, 1, or 2. The "code" fold provides the starting code. You must implement the models using the starting code.

1. (15 points) Data Preprocessing: In this problem, you need to finish "code/DataReader.py".

   (a) Explain what the function *train_valid_split* does and why we need this step.

   (b) Before testing, is it correct to re-train the model on the whole training set? Explain your answer.

   (c) In this assignment, we use two hand-crafted features:
   The first feature is a measure of symmetry. For a 16×16 image x, it is defined as

   $$F_{symmetry} = -\frac{\sum_{pixel} |x - flip(x)|}{256},$$

   where 256 is the number of pixels and $flip(\cdot)$ means left and right flipping.
   The second feature is a measure of intensity. For a 16×16 image x, it is defined as

   $$F_{intensity} = \frac{\sum_{pixel} x}{256},$$

   which is simply the average of pixel values.
   Implement them in the function *prepare_X*.

   (d) In the function *prepare_X*, there is a third feature which is always 1. Explain why we need it.

   (e) The function *prepare_y* is already finished. Note that the returned indices stores the indices for data from class 1 and 2. Only use these two classes for binary classification and convert the labels to +1 and -1 if necessary.

   (f) Test your code in "code/main.py" and visualize the training data from class 1 and 2 by implementing the function *visualize_features*. The visualization should not include the third feature. Therefore it is a 2-D scatter plot. Include the figure in your submission.

Written Analysis of Answers/Programming

1a] The function train_valid_split separates the original training data $\Leftrightarrow$ [raw_data] into training sets and validation sets at a specified [split_index]

what it does

Why
- Model Evaluation
- Hyperparameter Tuning
  → allows assessing model performance on unseen data before testing
- Validation set helps us detect
  → overfitting
  → yuileds hyperparameter selection

1b] Retraining on entire training set before training is ✓
After validating hyperparameters (eg. learning rate) on the validation set
⇒ combines all available data [training + validation)
maximizes information used for the final model
This improving generalization to the test set!

1c] Feature 1 := Symmetry
↳ 2 := Intensity
↳ 3 := Bias [will be 1s]

3]
np.ones(raw_X.shape[0])

see Code Submission forrest

1d] The bias term is always 1
to represent the intercept w_0
in the linear model
$w^T x = b$
W/o it, the decision boundary
is forced through the origin
⇒ Limits Flexibility of model!

1e] See LaTeX @ end

1f] The Scatterplot below visualizes the crafted features
→ Symmetry [x-axis]
→ Intensity [y-axis]
for the binary classification task

Class 1: Red Points (+1)
Class 2: Blue Points (-1)

- Trend: Potential Linear Seperability

2. (25 points) Cross-entropy loss: In logistic regression, we use the cross-entropy loss.

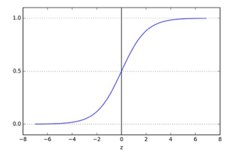
(a) Write the loss function $E(w)$ for one training data sample (x, y). Note that the binary labels are 1 and -1.

(b) Compute the gradient $\nabla E(w)$. Please provide intermediate steps of derivation.

(c) Once the optimal w is obtained, it can be used to make predictions as follows:

$$\text{Predicted class of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.5 \\ -1 & \text{if } \theta(w^T x) < 0.5 \end{cases}$$

where the function $\theta(z) = \frac{1}{1+e^{-z}}$ looks like



(d) Is the decision boundary still linear if the prediction rule is changed to the following? Justify briefly.

$$\text{Predicted label of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases}$$

(e) In light of your answers to the above two questions, what is the essential property of logistic regression that results in the linear decision boundary?

2a] Loss Function: Cross-entropy loss function $E(w)$ for one training data sample (x, y)

$:= E(w) = \ln(1 + e^{-y w^T x})$

w/ y is either $\begin{bmatrix} +1 \\ \text{or } -1 \end{bmatrix}$

2b] Gradient Computation

we must use chain rule

→ Let $z = w^T x$

→ we know $E = \ln(1 + e^{-yz})$

→ $\frac{dE}{dz} = \frac{-y e^{-yz}}{1 + e^{-yz}}$ →$\frac{dz}{dw} = x$

⇒ $\frac{dE}{dw} = \frac{dE}{\cancel{dz}} \cdot \frac{\cancel{dz}}{dw} = \frac{-y x}{1 + e^{y w^T x}}$

∴ the gradient is

$\nabla E(w) = \frac{-y x}{1 + e^{y w^T x}}$

2c] The given prediction rule is not the most efficient b/c:

→ The decision boundary is '<u>linear</u>' $:= w^T x = 0$

→ computing $\theta(w^T x)$ is unnecessary for classification

→ we can use the <u>sign</u> of $w^T x$ for prediction

Sigmoid Function $\theta(z)$ is needed for <u>Probability estimates</u>

<u>not just class predictions</u>

2d] Given Alternative Prediction Rule:

1 if $\theta(w^T x) \geq .9$

-1 if $\theta(w^T x) < .9$

⇒ The decision boundary is still <u>Linear</u>

Its defined by $w^T x = \ln 9$

[i.e a linear equation just shifted from origin]

2e] Essential Property of Logistic Regression

→ Linearity of $w^T x$

The Sigmoid function applies a monotonic transformation to this linear combination which <u>preserves</u> the linearity of the decision boundary.

Thus Logistic Regression can only seperate classes w/ a linear boundary

⇒ simple [easily interpretable]

⇒ inability to model non-linear relationships

3. (25 points) Sigmoid logistic regression: In this problem, you need to finish "code/LogisticRegression.py". **Please follow the instructions in the starting code. Please use data from class 1 and 2 for the binary classification.**

(a) Based on (b) in the last problem, implement the function *_gradient*.

(b) There are different ways to train a logistic regression model. In this assignment, you need to implement batch gradient descent, stochastic gradient descent and mini-batch gradient descent in the functions *fit_BGD*, *fit_SGD* and *fit_miniBGD*, respectively. Note that: In batch gradient descent, each model update is based on all training samples. In stochastic gradient descent, each model update is based on one training sample. In mini-batch gradient descent, you split your training data into (roughly equal-sized) mini-batches, and each model update is based on training samples in each mini-batch. Thus, BGD and SDG are actually special cases of mini-BGD. An epoch is completed when each training sample is used to update the model exactly once. In SDG and mini-BGD, you might want to reshuffle your samples before each epoch.

(c) Implement the functions *predict* and *score* for prediction and evaluation, respectively. Additionally, please implement the function *predict_proba* which outputs the probabilities of both classes.

(d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results*. The visualization should include the 2-D scatter plot of training data from class 1 and 2 and your decision boundary. Include the figure in your submission.

(e) Implement the testing process and report the test accuracy of your best logistic regression model.

3a] must implement gradient function from 2b]

$\Rightarrow \frac{dE}{dw} = \frac{dE}{dz} \cdot \frac{dz}{dw} = \frac{-y\,x}{1+e^{yw^Tx}}$

$\because$ the gradient is

$\nabla E(w) = \frac{-y\,x}{1+e^{yw^Tx}}$

$\rightarrow y \xrightarrow{\text{or}} -1$, $\;y \rightarrow 1$

x is feature vector

w is weight vector

3b] Required to implement 3 methods

1. Batch Gradient Descent [fit_BGD]
   - → update model using all training samples in each iteration
   - ↳ for each iteration we must:
     - compute gradient for all samples
     - update weights as
       $w = w - \text{learnin\_rate} \cdot \text{avg\_gradient}$

2. Stochastic Gradient Descent (SGD) [fit_SGD]
   - for each iteration
   - → Randomly select a sample
   - → Compute gradient for this sample
   - → update weights:
     $w = w - \text{learning\_rate} \cdot \text{gradient}$

3. Mini-Batch Gradient Descent [fit_miniBGD]
   for each iter:
   - → randomly select a 'mini-batch' of samples
   - → compute the average gradient for this 'mini-batch'
   - → update weights:
     $w = w - \text{Learning\_rate} \cdot \text{gradient}$

3c] Prediction / Evaluation functions

1) 'predict' function
   - → input: feature matrix X
   - → output: predicted labels -1 or 1
   - → implementation: return sign$(X \cdot w)$

2) 'score' function
   - → input: feature matrix X AND true labels y
   - → output: accuracy [% of correct predictions]
   - → returns % of correct predictions

3) 'predict_proba'
   - → feature matrix X [input]
   - → probabilities for both classes [out]
   - → sketch implementation
     - a] compute $z = X \cdot w$
     - b] return $\left[\frac{1}{1+e^{-z}}, \frac{1}{1+e^{z}}\right]$

3d] See LateX solution and Code Submission

Process
- Testing occurs in 'main.py'

4. (25 points) Softmax logistic regression: In this problem, you need to finish "code/LRM.py". **Please follow the instructions in the starting code.**

(a) Based on the course notes, implement the function *_gradient*.

(b) In this assignment, you only need to implement mini-batch gradient descent in the function *fit_miniBGD*.

(c) Implement the functions *predict* and *score* for prediction and evaluation, respectively.

(d) Test your code in "code/main.py" and visualize the results after training by using the function *visualize_results_multi*. The visualization should include the 2-D scatter plot of training data from class 0, 1, and 2, and your decision boundaries. Include the figure in your submission.

(e) Implement the testing process and report the test accuracy of your best logistic regression model.

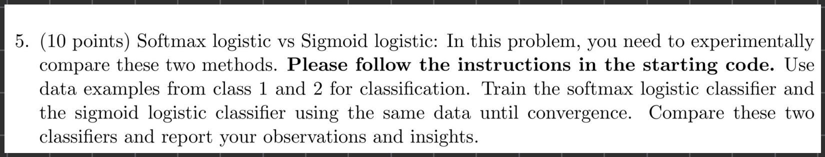
4a] See LateX / Code Submission

4b] See LateX / Code Submission

4c] See LateX / Code Submission

4d] See LateX and Code Submission

4e] See LateX and Code Submission



5. (10 points) Softmax logistic vs Sigmoid logistic: In this problem, you need to experimentally compare these two methods. **Please follow the instructions in the starting code.** Use data examples from class 1 and 2 for classification. Train the softmax logistic classifier and the sigmoid logistic classifier using the same data until convergence. Compare these two classifiers and report your observations and insights.

[See LateX for full breakdown]

Gen. Summary
→

[Pulled from Pycharm]

Output

```
[ 0.18761413  3.55944904 -2.06327285]
0.9444444444444444
[ 0.18761413  3.55944904 -2.06327285]
0.9444444444444444
[10.82741562 31.69274308  2.09324186]
0.9711111111111111
[11.05112829 31.87766955  2.24934715]
0.9703703703703703
[ 5.09262961 23.55262757 -3.14116836]
0.9733333333333334
Best parameters: learning_rate=0.5, batch_size=1
Best validation accuracy: 0.9788359788359788
Test accuracy: 0.9307359307359307
[[  6.75513933  -0.98294001  -5.7532125 ]
 [  0.27435837  15.72321774 -15.98338787]
 [ 10.43217196  -8.600453    -1.82368009]]
0.8960869565217391

Training multi-class classifier...
Best multi-class parameters: learning_rate=0.1, batch_size=100
Best multi-class validation accuracy: 0.8746031746031746

Training Softmax classifier...

Training Sigmoid classifier...

Classifier Comparison Results:
Softmax Classifier:
Training accuracy: 0.9719
Validation accuracy: 0.9788

Sigmoid Classifier:
Training accuracy: 0.9711
Validation accuracy: 0.9788
```

1. Experimental Setup
- Used digits 1 and 2 for binary classification
- Identical hyperparameters:
  * Learning rate = 0.1
  * Max iterations = 10000
  * Batch size = 50
- Features: Symmetry and Intensity

2. Performance Results
Softmax:
- Training accuracy: 97.19%
- Validation accuracy: 97.88%

Sigmoid:
- Training accuracy: 97.11%
- Validation accuracy: 97.88%

3. Key Observations
a) Accuracy:
- Nearly identical performance
- Difference in training accuracy is minimal (0.08%)
- Identical validation accuracy suggests equivalent generalization

b) Decision Boundaries:
- Both methods produced similar linear boundaries
- Boundaries align closely in feature space
- Effective separation between classes

4. Analysis
a) Theoretical Confirmation:
- Results validate theoretical equivalence for binary case
- Both methods converged to similar solutions
- Decision boundaries are mathematically equivalent

b) Implementation Differences:
- Softmax requires more computation (exponential terms)
- Both achieve same end result despite different approaches
- Choice between methods for binary classification should be based on implementation preference

5. Conclusion
The experimental results confirm that sigmoid and softmax logistic regression are functionally equivalent for binary classification, producing nearly identical performance metrics and decision boundaries.