Assignment 1 - Linear Regression and Logistic Regression

Due Date: Tuesday February 26th @ 11:59 PM on blackboard

The goal of this assignment is to implement the models and concepts introduced so far during the course as a code library, similar to modern python machine learning libraries such as scikit-learn.

- Linear Regression
- Logistic Regression
- Stochastic Gradient Descent
- Regularization

In the next assignment you will be using this code to conduct various experiments to test the other concepts we learned in the course.

To begin this assignment download the file assignment_01.zip from blackboard and unzip it to your machine.

Implementing Models and Training

In this assignment you will implement the Linear Regression and Logistic Regression models we learned in the course, L2 weight Regularization, as well as the stochastic gradient descent training code.

Important notes that apply to both Linear and Logistic Regression:

We are using the alternate convention for computing the model output:

$\hat{y} = Xw$ rather than $\hat{y} = w^T X$

This means each row of X corresponds to a *sample*, and each column to a separate *feature*. Your weights w are still a column vector.

Your model weights **should include the bias**. We are assuming a column of 1's are already given in your data X.

Your code should be vectorized using numpy, for example your **predict** function, should take in an entire matrix of data and return a vector of predictions. If you attempt to avoid numpy and use a bunch of loops, you'll end up with slow code that is more likely to be incorrect.

In the assignment file you will find several files with existing code, a description of each is given below.

linear_regression.py

This file is where you are to implement a linear regression model and training loop and all necessary helper functions.

The class is 'stubbed out' and your task is to implement the unimplemented functionality **within the given structure**. Please do not alter the structure that is given. You may introduce additional helper functions or methods, but do not alter function names or argument structures.

This API structure is inspired by (but not an exact copy of) modules such as **scikit-learn** and **keras** that are very commonly used in practice.

The comments and docstrings provide additional information that should help with your implementation.

Functions you must implement:

- squared_l2_norm - This is for implementing weight regularization
- mean_squared_error - This is for calculating your objective function and plotting learning curves
- _initialize_weights - This is where you create the weights for your model
- fit - This is what implements a full training loop using mini-batch Stochastic Gradient Descent, to minimize a **regularized** linear regression model
- _train_on_batch - This function performs a single stochastic gradient descent weight update, and should be called in fit. This is where you should factor in your α and λ hyperparameters.
- predict - This function calculates your model output ($\hat{y}$) for a batch of data _mse_gradient - This function calculates the MSE gradient w.r.t the model weights (this calculation should not include hyperparameters such as α) _l2_regularization_gradient - This function calculates the L2 regularization gradient wrt the model weights (this calculation should not include hyperparameters such as α or λ)

logistic_regression.py

This file is where you are to implement a logistic regression model. As in the previous section, use the given structure. You may re-use code from your linear own linear regression implementation. There is actually a huge amount of overlap between the two models (as we showed during class, even the gradient turns out to be the same, the real difference is the objective and the model).

The comments and docstrings provide additional information that should help with your implementation.

Functions you must implement:

- **squared_l2_norm** - This is for implementing weight regularization, you can use the same code as in linear regression
- **accuracy** - Calculate the accuracy of binary predictions compared to ground truth values (as a number between 0 and 1)
- **binary_cross_entropy** - This is for calculating your objective function and plotting learning curves
- **_initialize_weights** - This is where you create the weights for your model
- **fit** - This is what implements a full training loop using mini-batch Stochastic Gradient Descent, to minimize a **regularized** linear regression model
- **_train_on_batch** - This function performs a single stochastic gradient descent weight update, and should be called in fit. This is where you should factor in your α and λ hyperparameters.
- **predict_proba** - This function calculates your model output ($\hat{y}$) for a batch of data using the **Logistic Regression** model
- **predict** - This function does the same as predict_proba, but returns either 0 or 1 by choosing the closest class
- **_binary_cross_entropy_gradient** - This function calculates the binary cross-entropy gradient w.r.t the model weights (this calculation should not include hyperparameters such as α)
- **_l2_regularization_gradient** - This function calculates the L2 regularization gradient wrt the model weights (this calculation should not include hyperparameters such as α or λ)

test_linear_regression.py

This file includes a very minimal set of unit tests for the linear_regression.py part of the assignment. Part of the assignment grade will be based on your code passing these tests (and possibly unspecified tests, so that you can't cheat and hard-code your programs to pass only the given tests).

I also included a set of tests to make sure you changed this file, you can run py.test --verbose test_student.py

Any changes you make to this file will not be graded. You may add additional tests to help you during developing your code, but they will be discarded when grading your assignment

You may run these tests using the command: `py.test --verbose test_linear_regression.py`

The following is roughly what your output should look like if all tests pass (I may have added additional tests since this)

```
collected 7 items

test_linear_regression.py::test_squared_l2_norm PASSED                  [ 14%]
test_linear_regression.py::test_mean_squared_error PASSED               [ 28%]
test_linear_regression.py::test_weight_dimension PASSED                 [ 42%]
test_linear_regression.py::test_predict PASSED                          [ 57%]
test_linear_regression.py::test_mse_gradient PASSED                     [ 71%]
test_linear_regression.py::test_l2_regularization_gradient PASSED       [ 85%]
test_linear_regression.py::test_train_on_batch PASSED                   [100%]

========================== 7 passed in 0.04 seconds ==========================
```

test_logistic_regression.py

This file includes a very minimal set of unit tests for the logistic_regression.py part of the assignment. Part of the assignment grade will be based on your code passing these tests (and possibly unspecified tests, so that you can't cheat and hard-code your programs to pass only the given tests).

Any changes you make to this file will not be graded. You may add additional tests to help you during developing your code, but they will be discarded when grading your assignment

You may run these tests using the command: `py.test --verbose test_logistic_regression.py`

The following is the output if your code passes all tests:

```
test_logistic_regression.py::test_squared_l2_norm PASSED                [ 11%]
test_logistic_regression.py::test_sigmoid PASSED                        [ 22%]
test_logistic_regression.py::test_binary_cross_entropy PASSED           [ 33%]
test_logistic_regression.py::test_weight_dimension PASSED               [ 44%]
test_logistic_regression.py::test_predict_proba PASSED                  [ 55%]
test_logistic_regression.py::test_predict PASSED                        [ 66%]
test_logistic_regression.py::test_cross_entropy_gradient PASSED         [ 77%]
test_logistic_regression.py::test_l2_regularization_gradient PASSED     [ 88%]
test_logistic_regression.py::test_train_on_batch PASSED                 [100%]
```

```
========================== 9 passed in 0.10 seconds ==========================
```

student_info.py and test_student.py

Change the file student_info.py to your information. It currently contains my information/false information. Additionally, change the line i_cheated_on_this_assignment to False

```
first_name = "Jason"
last_name = "Jennings"
student_id = "1000123456"
i_cheated_on_this_assignment = True
```

you can run `py.test --verbose test_student.py` to check that this has been completed (or at least altered)

Grading Criteria

- Passing Unit Tests - 80 points (~5 points each for each unique test, code that is completely duplicated does not count twice. Note: Not all tests are given, so you can't 'fake it'.)
- Qualitative Evaluation - 20 points (Grader may examine your code and subjectively award as many as 20 points.)

No partial credit or credit for 'effort' will be given. Because you are given many small chunks you may implement and code to test your implementation, you can achieve partial credit this way.

This may appear to be a strange assignment, since there is no "main" program to run. You may (maybe even **should**) develop your own small examples to test your code as you are developing, but only the unit tests will be run. You can look at **test_fit_functional** in each model's corresponding set of unit tests to see what a minimal working example may look like.

Submission Guidelines

Upload your completed code as a .zip file to blackboard. Before submitting, you may wish to run all tests.

You can do this by running `py.test --verbose` in the code directory with no other arguments.