Due date: Tuesday, 10/23 11:59pm

1 Introduction

In this project you will implement the logistic regression model and test your model on two classification tasks. Through this project, you will learn

- the code structure of a classifier,
- the internal training procedure of a logistic regression model,
- the usage of regularization,
- overfitting and underfitting a classifier, and
- model selection.

2 Logistic regression: training and predicting

The logistic regression classifier is trained by solving the following minimization problem,

$$
\min_{w,w_0} L(w, w_0) = \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^{N} \log p(y_i|x_i; w, w_0)
$$

(1)

Here we assume that $y_i \in \{0, 1\}$. Each log-likelihood term $a_i(w, w_0) = \log p(y_i|x_i; w, w_0)$ is

$$
a_i(w, w_0) = y_i \log \left( \frac{1}{1 + \exp(-\eta_i)} \right) + (1 - y_i) \log \left( \frac{1}{1 + \exp(\eta_i)} \right)
$$

$$
= y_i \log \left( \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \right) + (1 - y_i) \log \left( \frac{1}{1 + \exp(\eta_i)} \right)
$$

(2)

Here $\eta_i = w^T x_i + w_0$. As we have talked during the class, (2) is the log-likelihood of the label $y_i$.

We train a logistic regression model by minimizing the objective in (1). In this project, we use the gradient descent and Stochastic Gradient Descent (SGD) methods to solve the optimization problem.
You will need to calculate the gradient \( \nabla L(w, w_0) \) of \( L(w, w_0) \) with respect to \((w, w_0)\). Though there are tools (see below) automatically calculating the gradient of the objective, understanding the gradient is important for debugging. The gradient of \( a_i(w, w_0) \) with respect to \( w \) and \( w_0 \) is

\[
\frac{\partial a_i}{\partial w} = \frac{\partial a_i}{\partial \eta_i} \cdot \frac{\partial \eta_i}{\partial w} = \left( y_i - \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \right) \cdot x_i, \quad (3)
\]

\[
\frac{\partial a_i}{\partial w} = \frac{\partial a_i}{\partial \eta_i} \cdot \frac{\partial \eta_i}{\partial w_0} = \left( y_i - \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} \right) \cdot 1. \quad (4)
\]

Note that \( \frac{\exp(\eta_i)}{1 + \exp(\eta_i)} = \frac{1}{1 + \exp(-\eta_i)} = \mu_i \), which is the probability of \( y_i = 1 \) in the logistic regression model. This gradient has the following indications: 1) to maximize \( a_i \) for an instance \( i \), \( w \) should move in the direction of \( x_i \) if the true label \( y_i \) is 1, or the opposite direction of \( x_i \) if the true label \( y_i \) is 0; and 2) the move is weighted by \( |y_i - \mu_i| \), which means an instance near the boundary (\( \mu_i \approx 0.5 \)) desires a bigger move comparing to instances with good predictive probabilities.

The gradient descent method iteratively updates the solution of \((w, w_0)\) by taking small steps in the opposite direction of its gradient. Suppose the solution at the \( t \)-th step is \((w^t, w_0^t)\) and the gradient of \( L(w^t, w_0^t) \) with respect to \((w^t, w_0^t)\) is \( \nabla L(w^t, w_0^t) \). Let \((w^0, w_0^0)\) be the initial guess of the solution. Then the gradient descent method update the solution as follows.

\[
(w^{t+1}, w_0^{t+1}) = (w^t, w_0^t) - \alpha_t \nabla L(w^t, w_0^t), t = 0, 1, \ldots, T - 1 \quad (5)
\]

Here \( \alpha_t \) is the step length.

SGD is very similar to the gradient descent method. It is much faster than gradient descent method for large datasets and thus has gained its popularity in deep learning in recent years.

SGD differs with gradient descent method only by its gradient calculation. As suggested by its name, it uses stochastic gradients, instead of exact gradients, to update \((w^t, w_0^t)\). A stochastic gradient is an estimation of the exact gradient. It is calculated from a small subset of instances, instead of the entire training set, so it is much cheaper than an exact gradient.

Decompose the logistic regression as follows.

\[
L(w, w_0) = \sum_{i=1}^{N} L_i(w, w_0) \quad (6)
\]

with

\[
L_i = -\log p(y_i|x_i; w, w_0) + \frac{\lambda}{2N} \|w\|^2 \quad (7)
\]

Instead of calculating \( L(w, w_0) \) on the entire training set, we *randomly* choose a small *batch* \( S \subset \{1, \ldots, N\} \) of instances and estimate \( L(w, w_0) \) by

\[
L(w, w_0) \approx \tilde{L}(w, w_0) = \frac{N}{|S|} \sum_{i \in S} L_i(w, w_0). \quad (8)
\]

Then the corresponding estimation of the exact gradient \( \nabla L(w, w_0) \) is

\[
\nabla L(w, w_0) \approx \nabla \tilde{L}(w, w_0) = \frac{N}{|S|} \sum_{i \in S} \nabla L_i(w, w_0). \quad (9)
\]
The update rule of SGD is the same as the gradient descent method except using stochastic gradients,

\[ (w^{t+1}, w_0^{t+1}) = (w^t, w_0^t) - \alpha_t \nabla \hat{L}(w^t, w_0^t), \quad t = 0, 1, \ldots, T - 1. \] (10)

While stochastic gradients \( \nabla \hat{L}(w, w_0) \) are less accurate than exact gradients and possibly lead to more optimization iterations, stochastic gradients are much cheaper than exact gradients. Overall SGD is usually faster than the gradient descent method in large problems.

After the either gradient descent method or SGD converges, \((w^T, w_0^T)\) becomes your classifier \((w, w_0)\). The classifier uses the following rule to make prediction.

\[ y = \lfloor w^\top x + w_0 \rfloor > 0 \] (11)

2.1 Implementation

You need to implement the optimization algorithm to train your logistic regression classifiers. Specifically, you need to implement the objective function \( L(w, w_0) \). To implement the SGD algorithm, you need to implement the objective \( \hat{L}(w, w_0) \) instead. You are required to implement \( L(w, w_0) \) and \( \hat{L}(w, w_0) \) with vector calculation – no for loops are allowed within the two functions (otherwise, the auto-differentiation in the next step will be painful).

If you program with Python, you can use autograd to get a function that calculates the gradient of your objective function. If you program with Matlab, then you can define the objective as a symbolic function, use gradient to take its gradient, which is another symbolic function, and then convert both functions to numeric functions with matlabFunction. [I’m not an expert in Matlab anymore – there might be better ways.] Since the gradient function is derived from your objective calculations above, it naturally calculate the corresponding, exact or stochastic, gradients for you.

If you want to implement your own function for gradient calculation, you should be aware that there is a strict match between the objective function and its gradient. To make sure the match, you are strongly suggested to run a gradient checking. The math behind it is

\[ L(w + \delta, w_0 + \delta_0) - L(w, w_0) \approx (\nabla L(w, w_0))^\top (\delta, \delta_0) \] (12)

with a small vector \((\delta, \delta_0)\) (e.g. every entry has an absolute value smaller than \(10^{-4}\)). You should calculate the left hand side with your implementation of the objective function and the right hand side with your implementation of the gradient function. The difference between the two values should be tiny (less than \(10^{-6}\)).

Then you need to implement the gradient descent algorithm. Here you need to use loops to update the solution. There are three issues you need to take care of. First, you need to specify the initial value \((w^0, w_0^0)\) – a vector of small random numbers usually suffices. Second, you need specify step length \(\alpha_t\) for every step \(t\). You can try either an aggressive setting of \(\alpha_t\) being a small constant or a conservative setting \(\alpha_t = \frac{1}{1 + t}\). Third, you need to specify a termination condition. There are several ways to do so. In this project, the suggested condition is

\[ \text{diff} = \frac{1}{d + 1} \left( |w^t_0 - w_0^{(t-100)}| + \sum_{k=1}^{d} \left| w^t_k - w_k^{(t-100)} \right| \right) > \epsilon = 10^{-4} \] (13)

Here \(\text{diff}\) means the average difference at each entry. The thought behind it is: if \(\text{diff}\) is less than \(10^{-4}\), then probably \((w, w_0)\) will not have much changes in next, say, 1000 iterations. The
decisions would not change for most instances after another 1000 training iterations. The criterion $\epsilon$ can be a little bit larger or smaller depending on how accurate you want your final solution to be (just like wringing water from clothes - you can always get more water out, but you need to stop somewhere).

You should print out the objective $L(w, w_0)$ or $\tilde{L}(w, w_0)$ for every (say) 100 iterations. $L(w, w_0)$ should always decrease with iterations. $\tilde{L}(w, w_0)$ should have a clear decreasing trend possibly with some oscillation.

3 Training and testing

As we will talk in the class, we are interested in not only training a classifier but also estimating its accuracy.

In order to achieve the two goals, you first need to split your data into two sets: a train set and validation set. The train set is for training the classifier (i.e. provide data in the optimization of $w$ and $w_0$). The validation set is for testing the accuracy of your trained classifier. The validation accuracy is an estimation of the classifier trained on the training set.

In the situation you want to use all labeled data as the training set but at the same time want an estimation of the accuracy of the trained classifier, you should use cross-validation. To run cross-validation, you need to divide the dataset into $k$ folds and keep class ratio in each fold. Then in each round, you use one fold as the validation set and all other folds combined as the training set. After $k$ rounds, you have predictions for each instance (though predictions are from classifiers trained on different training sets), then the accuracy calculated from these predictions is an estimation of the classifier trained on the entire training set.

4 Experiments

Once all the above is implemented, you need run the following experiments and report the results.

4.1 Full gradient versus stochastic gradient

In this experiment, you will observe different performances of the two optimization methods. You need to train your logistic regression classifier on the entire training set with the two optimization methods and plot convergence curves. A convergence curve is a plot of objective values versus iterations or running time. Use $L(w, w_0)$ as the objective value of the gradient descent method and $\tilde{L}(w, w_0)$ for SGD.

You need to generate two plots: both use $y$-axis as the training objective, but one uses the $x$-axis as the training time in seconds, and the other one uses the $x$-axis as the number of iterations. Each plot has two curves, one from SGD with batch size of 10 and the other from the gradient descent method.

You only need to generate the two plots on the MNIST dataset.

4.2 Effects of regularizer

Separate the dataset into a training set and a test set by 7 : 3, and then run your logistic regression model with different $\lambda$ values: $\lambda \in \{0, 0.01, 0.1, 1, 10, 100, 1000\}$, and plot training and test accuracy
values versus different $\lambda$ values. You are expected to see how the regularizer weight influences training accuracy and test performance.

In each dataset, the last feature is a random feature. In theory, the model should not use the feature and the corresponding weight to the feature is zero. In the experiment, we expect the model will have smaller weight for the feature when the regularization is stronger. See if you can observe this effect in the experiment. Please plot the absolute weight value corresponding the random feature versus the regularizer value.

You need to run this experiment for each dataset.

4.3 Model Selection

With your previous experiment, you will find the best $\lambda$ value for each dataset. Use the best $\lambda$ value to train a classifier on the entire training set. Your classifier will be tested on our held-out dataset.

4.4 Cross-validation

Implement cross-validation and use it to estimate the accuracy of your classifier for submission. You need to report your estimation in the form ($accuracy \pm std$).

5 Interface of the implementation

You are supposed to implement the classifier with functions using the following interfaces in Python. If you are using the Matlab, please use the same interfaces.

```python
import numpy

class MyLogisticReg:
    """A class for Linear Regression Models""

    def __init__(self, options):
        """
        Put classifier options here as data attributes. Options include the type of gradients, regularizer weight, etc.

        Also put trainable classifier parameters here. The function fit below will learn these parameters.
        """
        pass

    def fit(self, X, y):
        """Fit model. This function trains model parameters with input train data X and y

        Parameters
        ----------
        X: numpy.array, shape (n_samples, n_features)
        """
```


Feature matrix of training instances
\( y: \text{numpy.array.shape(n_samples,)} \)
Label vector of training instances
"""
pass

def predict(self, X):
    """Predict using the logistic regression model
Parameters
    X : np.array, shape (n_samples, n_features)
        Test instances.
Returns
    y_pred : np.array, shape (n_samples,)
        Returns predicted values.
    """
    y_pre = np.zeros(X.shape[0])
    return y_pred

def evaluate(y_test, y_pred):
    """Evaluate the accuracy of predictions against true labels.
Parameters
    y_test: np.array, shape (n_samples,)
        True labels.
y_pred: np.array, shape (n_samples,)
        Predicted labels.
Returns
    error_rate: a float, in \([0, 1]\),
        The error rate of the prediction.
    """
    error_rate = (np.sum(np.equal(y_test, y_pred).astype(np.float)) / y_test.size)
    return error_rate

def main():
    """
    You should implement your own main function. Here is just an reference
use whatever preprocessing package you like to get data
matrix from data file. One example:

\[
X, Y = \text{read\_in\_data\_function}(\text{datafile\_name}, \text{vocabulary})
\]

split data into two folds
\[
(X_{\text{train}}, Y_{\text{train}},
X_{\text{test}}, Y_{\text{test}}) = \text{split\_data}(X, Y, \text{train\_test\_ratio})
\]

if you do the experiment of training, validating, and testing
further split the training set into training and validation
subsets
\[
X_{\text{validation}}, Y_{\text{validation}}, X_{\text{test}}, Y_{\text{test}} =
\text{split\_data}(X_{\text{train}}, Y_{\text{train}}, \text{train\_valid\_ratio})
\]

```python
# data preparation
(X_train, Y_train,
X_test, Y_test) = split_data(X, Y, train_test_ratio)

# model fitting
options = None
model = MyLogisticReg(options)
model.fit(X_train, Y_train)

# prediction
y_pred = model.predict(X_test)

# evaluation
error_rate = evaluate(y_test, y_pred)
print('The error rate of my classifier is ' + str(error_rate))
```

6 Programming Language and use of Libraries/Modules

You may write your program in any language as long as your code runs on our server homework.eecs.tufts.edu. You are of course allowed to use basic I/O, math library, randomization and other basic language facilities. But you should write your own code for all the portions described above. For example, you are free to use numpy and autograd libraries in Python, but you cannot use scikit-learn.

7 Submission requirements

You need to follow submission requirements in this section to guarantee that you submission can be evaluated smoothly.
You need to submit your report to gradescope and the following items to the homework server. In the submission to the server, you need to include

1) a zip file containing your code. Your submitted code should clearly show your implementation of the logistic regression classifier and required experiments in this project. Please make sure that your code runs on homework.eecs.tufts.edu. Please include a README file with instructions how to compile and run your code to reproduce the results of experiments. If this is nontrivial, please include a script to compile and run your code.

2) your classifier files. If you use Python, you should serialize your classifiers into files with Python pickle. The size of a classifier file cannot exceed 1M. Please save your two classifiers to two files: titanic_classifier.pkl and mnist_classifier.pkl. NOTE: please use exactly the two file names, otherwise, we won’t know which file have your classifier for a classification task. If you use Matlab, you should use the function saveCompactModel to save your classifiers. Please use the file names instructed above except the Matlab suffix .mat.

3) your nickname in a file. As we will generate a leaderboard for the competition, we need your nicknames to show the performance of your classifiers. You may want a cool nickname but others cannot identify you from it.

Put all the files mentioned in the previous subsection into a .zip file. For example call it myfile.zip. Then submit using provide comp135 proj1 myfile.zip.

7.1 Grading

Your assignment will be graded based on the code, its clarity, documentation and correctness, the presentation of the results/plots, and their discussion.

8 Two classification tasks

Below are Duc’s vivid introductions of two datasets.

8.1 Titanic dataset: Dead or Alive?

It was said that after surviving the sinking of the RMS Titanic, Rose returned to England, her heart broken into pieces. Every night she was haunted by the same thought: "If only he had decided to share the plank with me...", and yet her love for Jack was so strong that she believed that somehow Jack had survived the accident. So Rose set out to search for Jack. Unfortunately, she could not find his name under the list of survivors so she decided to ask a machine learning expert to help her build a model that can predict if Jack survived the sinking based on data about the survivors and deceased. Your task is build a model that can predict as accurately as possible whether a person survive the Titanic tragedy.

The dataset given to you will consist of 710 rows, and each row represents a passenger that boarded the RMS Titanic. Each passenger's information is made up of 1 label (survived or not) and 7 features, which are (starting from left to right):

- **Survived**: Whether the passenger survived or not, has 2 types 0 = deceased, 1 = survived
  
  (This is the label, which is what we want to predict)
• **Pclass**: Ticket class of the passenger in 3 types: 1 = 1st class, 2 = 2nd class, and 3 = 3rd class.

• **Sex**: Sex of the passenger in 2 types: 0=Female, and 1=Male.

• **Age**: Age in years of the dataset

• **Siblings/Spouses Aboard**: The number of siblings/spouses aboard the Titanic

• **Parents/Children Aboard**: The number of parents/children aboard the Titanic

• **Fare**: Passenger fare

• **rand-feature**: a random feature.

*Note: you should consider the ranges of different features. For example, the feature **Age** have a much larger range than **Sex**. Is it an issue for your classifier? If so, how to handle this issue?

9 MNIST dataset - patients’ story

It is a proven fact that doctors have bad handwritings, and people have come to terms with it. But Dr. Drake Nguyen’s handwriting is something that even the most patient patient in the world will give up trying to comprehend out of frustration. Patients at Tufts Medical Center, tired of deciphering his handwriting, decide to hire you to build a machine learning model to help recognize Dr. Drake Nguyen’s handwritten digits based on a dataset provided by the doctor himself. If you succeed, the number of people who overdose on medications because they misinterpret his handwriting will greatly reduce.

The make the problem a binary classification problem, we only classify whether the number being “8” or “9”. The dataset given to you is a csv file. Each row is a hand-written number. The first entry is the label, “8” or “9”. The rest 785 entries are pixel values of the $28 \times 28$ image of the written number and a random feature.