02_assignment_stochastic_gradient_descent_robertson

March 23, 2025

Assignment 2: Stochastic Gradient Descent and Momentum

CPSC 381/581: Machine Learning

Yale University

Instructor: Alex Wong

Student: Hailey Robertson

Prerequisites:

1. Enable Google Colaboratory as an app on your Google Drive account
2. Create a new Google Colab notebook, this will also create a "Colab Notebooks" directory under "MyDrive" i.e.

```
/content/drive/MyDrive/Colab Notebooks
```

3. Create the following directory structure in your Google Drive

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Assignments
```

4. Move the 02_assignment_stochastic_gradient_descent.ipynb into

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Assignments
```

so that its absolute path is

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Assignments/02_assignment
```

In this assignment, we will optimize a linear function for the logistic regression task using the stochastic gradient descent and its momentum variant. We will test them on several binary classification datasets (breast cancer, digits larger or less than 5, and fir and pine coverage). We will implement a training and validation loop for the binary classification task and test it on the testing split for each dataset.

Submission:

1. Implement all TODOs in the code blocks below.
2. Report your training, validation, and testing scores.

```
Report training, validation, and testing scores here.
```

3. List any collaborators.

```
Collaborators: N/A
```

IMPORTANT:

- For full credit, your mean classification accuracies for all trained models across all datasets should be no more than 8% worse the scores achieved by sci-kit learn's logistic regression model across training, validation and testing splits.
- You may not use batch sizes of more than 10% of the dataset size for stochastic gradient descent and momentum stochastic gradient descent.
- You will only need to experiment with gradient descent (GD) and momentum gradient descent (momentum GD) on breast cancer and digits (toy) datasets. It will take too long to run them on fir and pine coverage (realistic) dataset to get reasonable numbers. Of course, you may try them on fir and pine coverage :) but they will not count towards your grade.
- Note the run time speed up when comparing GD and momemtum GD with stochastic gradient descent (SGD) and momentum stochastic gradient descent (momentum SGD)! Even though they are faster and observing batches instead of the full dataset at each time step, they can still achieving similar accuracies!

[6]:
```
import numpy as np
import sklearn.datasets as skdata
import sklearn.metrics as skmetrics
from sklearn.linear_model import LogisticRegression as LogisticRegressionSciKit
import warnings
import time

warnings.filterwarnings(action='ignore')
np.random.seed = 1
```

Implementation of stochastic gradient descent optimizer for logistic loss

[7]:
```
class Optimizer(object):

    def __init__(self, alpha, eta_decay_factor, beta, optimizer_type):
        '''
        Arg(s):
            alpha : float
                initial learning rate
            eta_decay_factor : float
                learning rate decay rate
            beta : float
                momentum discount rate
            optimizer_type : str
                'gradient_descent',
                'momentum_gradient_descent',
                'stochastic_gradient_descent',
                'momentum_stochastic_gradient_descent'
        '''

        self.__alpha = alpha
```

```
        self.__eta_decay_factor = eta_decay_factor
        self.__beta = beta
        self.__optimizer_type = optimizer_type
        self.__momentum = None

    def __compute_gradients(self, w, x, y, loss_func='logistic'):
        '''
        Returns the gradient of a loss function

        Arg(s):
            w : numpy[float32]
                d x 1 weight vector
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            loss_func : str
                loss by default is 'logistic' only for the purpose of the␣
↪assignment
        Returns:
            numpy[float32] : d x 1 gradients
        '''

        # DONE: Implement compute_gradient function

        if loss_func == 'logistic':
            # print('w shape: {}'.format(w.shape)) # (1, 30)
            N = x.shape[1] # number of samples, 341
            z = np.dot(w.T, x)
            # print('y shape: {}'.format(y.shape)) # (1, 341)
            # print('z shape: {}'.format(z.shape)) # (1, 341)
            sigmoid = 1 / (1 + np.exp(-y * z))
            gradient = -np.dot(x, ((y * (1 - sigmoid))).T) / N

            # print('gradient shape: {}'.format(gradient.shape))

            return gradient

        else:
            raise ValueError('Unupported loss function: {}'.format(loss_func))

    def __polynomial_decay(self, time_step):
        '''
        Computes the polynomial decay factor t^{-a}

        Arg(s):
            time_step : int
```

```
                current step in optimization
        Returns:
            float : polynomial decay to adjust (reduce) initial learning rate
        '''

        # DONE: Implement polynomial decay to adjust the initial learning rate
        if self.__eta_decay_factor is None:
            return self.__alpha
        else:
            decay_factor = time_step ** (-self.__eta_decay_factor)
            decayed_alpha = self.__alpha * decay_factor

        return decayed_alpha

    def update(self,
               w,
               x,
               y,
               loss_func,
               batch_size,
               time_step):
        '''
        Updates the weight vector based on

        Arg(s):
            w : numpy[float32]
                d x 1 weight vector
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            loss_func : str
                loss function to use, should be 'logistic' for the purpose of␣
↪the assignment
            batch_size : int
                batch size for stochastic and momentum stochastic gradient␣
↪descent
            time_step : int
                current step in optimization
        Returns:
            numpy[float32]: d x 1 weights
        '''

        # DONE: Implement the optimizer update function
        # For each optimizer type, compute gradients and update weights
        eta = self.__polynomial_decay(time_step) # step size
        gradient = self.__compute_gradients(w, x, y, loss_func)
```

```
        if self.__optimizer_type == 'gradient_descent':
            # print('gradient shape: {}'.format(gradient.shape)) # (30, 1)
            # print('w shape: {}'.format(w.shape)) # (30, 1)
            # print("Before update: w[0]:", w[0])
            w = w - eta * gradient
            # print("After update: w[0]:", w[0])
            return w

        elif self.__optimizer_type == 'momentum_gradient_descent':
            if self.__momentum is None:
                self.__momentum = np.zeros(w.shape)
            self.__momentum = self.__beta * self.__momentum + (1-self.__beta) *␣
↪gradient
            w = w - eta * self.__momentum
            return w

        elif self.__optimizer_type == 'stochastic_gradient_descent':
            indices = np.random.choice(x.shape[1], batch_size, replace=False)
            x_batch = x[:, indices]
            y_batch = y[:, indices]
            gradient = self.__compute_gradients(w, x_batch, y_batch, loss_func)
            w = w - eta * gradient
            return w

        elif self.__optimizer_type == 'momentum_stochastic_gradient_descent':
            if self.__momentum is None:
                self.__momentum = np.zeros(w.shape)
            indices = np.random.choice(x.shape[1], batch_size, replace=False)
            x_batch = x[:, indices]
            y_batch = y[:, indices]
            gradient = self.__compute_gradients(w, x_batch, y_batch, loss_func)
            self.__momentum = self.__beta * self.__momentum + (1-self.__beta) *␣
↪gradient
            w = w - eta * self.__momentum
            return w

        else:
            raise ValueError('Unsupported optimizer type: {}'.format(self.
↪__optimizer_type))
```

Implementation of our logistic regression model for binary classification

[8]:
```
class LogisticRegression(object):

    def __init__(self):
        # Define private variables
```

```
        self.__weights = None
        self.__optimizer = None

    def fit(self,
            x,
            y,
            T,
            alpha,
            eta_decay_factor,
            beta,
            batch_size,
            optimizer_type,
            loss_func='logistic'):
        '''
        Fits the model to x and y by updating the weight vector
        using gradient descent

        Arg(s):
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            T : int
                number of iterations to train
            alpha : float
                learning rate
            eta_decay_factor : float
                learning rate decay rate
            beta : float
                momentum discount rate
            batch_size : int
                number of examples per batch
            optimizer_type : str
                'gradient_descent',
                'momentum_gradient_descent',
                'stochastic_gradient_descent',
                'momentum_stochastic_gradient_descent'
            loss_func : str
                loss function to use, by default is 'logistic' only for the␣
↪purpose of the assignment
        '''

        # DONE: Instantiate optimizer and weights
        self.__optimizer = Optimizer(alpha, eta_decay_factor, beta,␣
↪optimizer_type)
        self.__weights = np.random.randn(x.shape[0], 1) * 0.01
```

```
        for t in range(1, T + 1):

            # DONE: Compute loss function
            loss = self.__compute_loss(x, y, loss_func)


            if (t % 1000) == 0:
                print('Step={}  Loss={}'.format(t, loss))

            # DONE: Update weights
            self.__weights = self.__optimizer.update(self.__weights, x, y,␣
↪loss_func, batch_size, t)
            # print("loss", loss, "weights", self.__weights)


    def predict(self, x):
        '''
        Predicts the label for each feature vector x

        Arg(s):
            x : numpy[float32]
                d x N feature vector
        Returns:
            numpy[float32] : 1 x N vector
        '''

        # DONE: Implements the predict function
        # Hint: logistic regression predicts a value between 0 and 1
        z = np.dot(self.__weights.T, x)
        sigmoid = 1 / (1+ np.exp(-z))
        # print(sigmoid.shape)
        predictions = np.where(sigmoid >= 0.5, 1, -1)
        # print(predictions)

        return predictions

    def __compute_loss(self, x, y, loss_func):
        '''
        Computes the logistic loss

        Arg(s):
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            loss_func : str
```

```
                loss function to use, by default is 'logistic' only for the␣
↪purpose of the assignment
        Returns:
            float : loss
        '''

        # DONE: Implements the __compute_loss function
        if loss_func == 'logistic':

            N = x.shape[1] # number of samples, 341
            z = np.dot(self.__weights.T, x)
            loss = np.sum(np.log(1 + np.exp(-y * z))) / N

        else:
            raise ValueError('Unsupported loss function: {}'.format(loss_func))

        return loss
```

Training, validating and testing logistic regression for binary classification

[9]:
```
def compare_scores(score_train, score_val, score_test, sk_score_train,␣
↪sk_score_val, sk_score_test):
    """
    Compare the performance of our logistic regression model with scikit-learn␣
↪model.
    If the performance is more than 8% worse for any dataset (train/validation/
↪test),
    print 'Rerun' because I don't want to assess this by hand.
    """
    threshold = 0.08

    if score_train < sk_score_train * (1 - threshold):
        print(f"\nTraining score is more than 8% worse. Ours: {score_train:.
↪4f}, Scikit: {sk_score_train:.4f}. Rerun.")

    if score_val < sk_score_val * (1 - threshold):
        print(f"\nValidation score is more than 8% worse. Ours: {score_val:.
↪4f}, Scikit: {sk_score_val:.4f}. Rerun.")

    if score_test < sk_score_test * (1 - threshold):
        print(f"\nTest score is more than 8% worse. Ours: {score_test:.4f},␣
↪Scikit: {sk_score_test:.4f}. Rerun.")

    if (score_train >= sk_score_train * (1 - threshold)) and \
       (score_val >= sk_score_val * (1 - threshold)) and \
       (score_test >= sk_score_test * (1 - threshold)):
        print("\nPerformance is acceptable.\n")
```

[10]:
```
# Load breast cancer, digits, and tree coverage datasets
datasets = [
    skdata.load_breast_cancer(),
    skdata.load_digits(),
    skdata.fetch_covtype()
]
dataset_names = [
    'breast cancer',
    'digits greater or less than 5',
    'fir and pine coverage',
]

# Loss functions to minimize
dataset_optimizer_types = [
    # For breast cancer dataset
    [
        'gradient_descent',
        'momentum_gradient_descent',
        'stochastic_gradient_descent',
        'momentum_stochastic_gradient_descent'
    # For digits greater than or less than 5 dataset
    ], [
        'gradient_descent',
        'momentum_gradient_descent',
        'stochastic_gradient_descent',
        'momentum_stochastic_gradient_descent'
    # For fir and pine coverage dataset
    ], [
        'stochastic_gradient_descent',
        'momentum_stochastic_gradient_descent'
    ]
]

# DONE: Select hyperparameters

# Step size (always used)
dataset_alphas = [
    # For breast cancer dataset
    [1e-6, 1e-6, 0.001, 0.01],
    # For digits greater than or less than 5 dataset
    [1e-4, 1e-4, 0.01, 0.1],
    # For fir and pine coverage dataset
    [0.001, 0.01]
]

# How much the learning rate should decrease over time (only for SGD)
dataset_eta_decay_factors = [
```

```
    # For breast cancer dataset
    [None, None, 0.99, 0.99],
    # For digits greater than or less than 5 dataset
    [None, None, 0.95, 0.95],
    # For fir and pine coverage dataset
    [0.99, 0.99]
]

# Momentum (only for momentum-based ..duh)
# None for no momentum
dataset_betas = [
    # For breast cancer dataset
    [None, 0.9, None, 0.9],
    # For digits greater than or less than 5 dataset
    [None, 0.9, None, 0.9],
    # For fir and pine coverage dataset
    [None, 0.9]
]

# Samples (only for SGD)
dataset_batch_sizes = [
    # For breast cancer dataset
    # # N = 569
    [None, None, 24, 24],
    # For digits greater than or less than 5 dataset
    # # N = 1797
    [None, None, 32, 32],
    # For fir and pine coverage dataset
    # # N = 495141
    [1000, 1000]
]


# Iterations
dataset_Ts = [
    # For breast cancer dataset
    [10000, 10000, 10000, 10000],
    # For digits greater than or less than 5 dataset
    [10000, 10000, 10000, 10000],
    # For fir and pine coverage dataset
    [3000, 3000]
]

# Zip up all dataset options
dataset_options = zip(
    datasets,
    dataset_names,
```

```
    dataset_optimizer_types,
    dataset_alphas,
    dataset_eta_decay_factors,
    dataset_betas,
    dataset_batch_sizes,
    dataset_Ts)


for options in dataset_options:

    # Unpack dataset options
    dataset, \
        dataset_name, \
        optimizer_types, \
        alphas, \
        eta_decay_factors, \
        betas, \
        batch_sizes, \
        Ts = options

    '''
    Create the training, validation and testing splits
    '''
    x = dataset.data
    y = dataset.target

    if dataset_name == 'digits greater or less than 5':
        y[y < 5] = 1
        y[y >= 5] = 0
    elif dataset_name == 'fir and pine coverage':

        idx_fir_or_pine = np.where(np.logical_or(y == 1, y == 2))[0]

        x = x[idx_fir_or_pine, :]
        y = y[idx_fir_or_pine]

        # Pine class: 0; Fir class: 1
        y[y == 2] = 0

    print('Preprocessing the {} dataset ({} samples, {} feature dimensions)'.
 ↪format(dataset_name, x.shape[0], x.shape[1]))

    # Shuffle the dataset based on sample indices
    shuffled_indices = np.random.permutation(x.shape[0])

    # Choose the first 60% as training set, next 20% as validation and the rest␣
 ↪as testing
```

```
    train_split_idx = int(0.60 * x.shape[0])
    val_split_idx = int(0.80 * x.shape[0])

    train_indices = shuffled_indices[0:train_split_idx]
    val_indices = shuffled_indices[train_split_idx:val_split_idx]
    test_indices = shuffled_indices[val_split_idx:]

    # Select the examples from x and y to construct our training, validation, testing sets
    x_train, y_train = x[train_indices, :], y[train_indices]
    x_val, y_val = x[val_indices, :], y[val_indices]
    x_test, y_test = x[test_indices, :], y[test_indices]

    '''
    Trains and tests logistic regression model from scikit-learn
    '''
    model_scikit = LogisticRegressionSciKit(penalty=None, fit_intercept=False)

    # DONE: Train scikit-learn logistic regression model
    model_scikit.fit(x_train, y_train)


    print('***** Results on the {} dataset using scikit-learn logistic regression model *****'.format(dataset_name))

    # DONE: Score model using mean accuracy on training set
    predictions_train = model_scikit.predict(x_train)
    sk_score_train = skmetrics.accuracy_score(y_train, predictions_train)
    print('Training set mean accuracy: {:.4f}'.format(sk_score_train))

    # DONE: Score model using mean accuracy on validation set
    predictions_val = model_scikit.predict(x_val)
    sk_score_val = skmetrics.accuracy_score(y_val, predictions_val)
    print('Validation set mean accuracy: {:.4f}'.format(sk_score_val))

    # DONE: Score model using mean accuracy on testing set
    predictions_test = model_scikit.predict(x_test)
    sk_score_test = skmetrics.accuracy_score(y_test, predictions_test)
    print('Testing set mean accuracy: {:.4f}'.format(sk_score_test))

    '''
    Trains, validates, and tests our logistic regression model for binary classification
    '''
    # Take the transpose of the dataset to match the dimensions discussed in lecture
    # i.e., (N x d) to (d x N)
```

```
    x_train = np.transpose(x_train, axes=(1, 0))
    x_val = np.transpose(x_val, axes=(1, 0))
    x_test = np.transpose(x_test, axes=(1, 0))
    y_train = np.expand_dims(y_train, axis=0)
    y_val = np.expand_dims(y_val, axis=0)
    y_test = np.expand_dims(y_test, axis=0)

    # DONE: Set the ground truth to the appropriate classes (integers) according to lecture
    # -1 and +1
    y_train = np.where(y_train == 0, -1, 1)
    y_val = np.where(y_val == 0, -1, 1)
    y_test = np.where(y_test == 0, -1, 1)

    model_options = zip(optimizer_types, alphas, eta_decay_factors, betas, batch_sizes, Ts)

    for optimizer_type, alpha, eta_decay_factor, beta, batch_size, T in model_options:

        # DONE: Initialize our logistic regression model
        model_ours = LogisticRegression()

        print('***** Results of our logistic regression model trained on {} dataset *****'.format(dataset_name))
        print('\t optimizer_type={} \n\t alpha={} \n\t eta_decay_factor={} \n\t beta={} \n\t batch_size={} \n\t T={}'.format(
            optimizer_type, alpha, eta_decay_factor, beta, batch_size, T))

        time_start = time.time()

        # DONE: Train model on training set
        model_ours.fit(
            x_train,
            y_train,
            T,
            alpha,
            eta_decay_factor,
            beta,
            batch_size,
            optimizer_type
        )

        time_elapsed = time.time() - time_start
        print('Total training time: {:3f} seconds'.format(time_elapsed))

        # DONE: Score model using mean accuracy on training set
```

```
        predictions_train = model_ours.predict(x_train)
        score_train = np.mean(predictions_train == y_train)
        # print("Predictions", predictions_train)
        # print("Ground truth", y_train)
        print('Training set mean accuracy: {:.4f}'.format(score_train))

        # DONE: Score model using mean accuracy on training set
        predictions_val = model_ours.predict(x_val)
        score_val = np.mean(predictions_val == y_val)
        print('Validation set mean accuracy: {:.4f}'.format(score_val))

        # DONE: Score model using mean accuracy on training set
        predictions_test = model_ours.predict(x_test)
        score_test = np.mean(predictions_test == y_test)
        print('Testing set mean accuracy: {:.4f}'.format(score_test))

        compare_scores(
            score_train, score_val, score_test,
            sk_score_train, sk_score_val, sk_score_test
        )

    print('')
```

```
Preprocessing the breast cancer dataset (569 samples, 30 feature dimensions)
***** Results on the breast cancer dataset using scikit-learn logistic
regression model *****
Training set mean accuracy: 0.9355
Validation set mean accuracy: 0.9386
Testing set mean accuracy: 0.9211
***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=gradient_descent
         alpha=1e-06
         eta_decay_factor=None
         beta=None
         batch_size=None
         T=10000
Step=1000  Loss=0.4448779736881357
Step=2000  Loss=0.3599476894609853
Step=3000  Loss=0.3240723977754247
Step=4000  Loss=0.3027336798429431
Step=5000  Loss=0.2881630181897578
Step=6000  Loss=0.2774524239317973
Step=7000  Loss=0.26919230856031323
Step=8000  Loss=0.2625955604324603
Step=9000  Loss=0.2571833276928684
Step=10000  Loss=0.2526466689717918
Total training time: 0.270742 seconds
```

```
Training set mean accuracy: 0.9179
Validation set mean accuracy: 0.9035
Testing set mean accuracy: 0.9298

Performance is acceptable.

***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=momentum_gradient_descent
         alpha=1e-06
         eta_decay_factor=None
         beta=0.9
         batch_size=None
         T=10000
Step=1000  Loss=0.5149999181760895
Step=2000  Loss=0.43788449458213824
Step=3000  Loss=0.3880642607855869
Step=4000  Loss=0.3521563599732627
Step=5000  Loss=0.32503127450325864
Step=6000  Loss=0.3041348058099797
Step=7000  Loss=0.2883105038337543
Step=8000  Loss=0.27685687328477215
Step=9000  Loss=0.268464584095708
Step=10000  Loss=0.26196525963686834
Total training time: 0.290265 seconds
Training set mean accuracy: 0.9179
Validation set mean accuracy: 0.9035
Testing set mean accuracy: 0.9298

Performance is acceptable.

***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=stochastic_gradient_descent
         alpha=0.001
         eta_decay_factor=0.99
         beta=None
         batch_size=24
         T=10000
Step=1000  Loss=0.39260816721134717
Step=2000  Loss=0.37600847814341
Step=3000  Loss=0.36583787815897106
Step=4000  Loss=0.35768376582000916
Step=5000  Loss=0.3523432719526187
Step=6000  Loss=0.34772405807707846
Step=7000  Loss=0.3442223361839243
Step=8000  Loss=0.34198879235529717
Step=9000  Loss=0.3384262742498414
```

```
Step=10000  Loss=0.3355524398889064
Total training time: 0.486495 seconds
Training set mean accuracy: 0.9091
Validation set mean accuracy: 0.8860
Testing set mean accuracy: 0.9211

Performance is acceptable.

***** Results of our logistic regression model trained on breast cancer dataset
*****
         optimizer_type=momentum_stochastic_gradient_descent
         alpha=0.01
         eta_decay_factor=0.99
         beta=0.9
         batch_size=24
         T=10000
Step=1000  Loss=0.7088008151052103
Step=2000  Loss=0.6344943256524684
Step=3000  Loss=0.5989428366963758
Step=4000  Loss=0.575321883320887
Step=5000  Loss=0.5586694251653267
Step=6000  Loss=0.5436818632638798
Step=7000  Loss=0.5331701222941695
Step=8000  Loss=0.5245188979144549
Step=9000  Loss=0.5185704885089195
Step=10000  Loss=0.512027355560007
Total training time: 0.503408 seconds
Training set mean accuracy: 0.9179
Validation set mean accuracy: 0.9123
Testing set mean accuracy: 0.9298

Performance is acceptable.


Preprocessing the digits greater or less than 5 dataset (1797 samples, 64
feature dimensions)
***** Results on the digits greater or less than 5 dataset using scikit-learn
logistic regression model *****
Training set mean accuracy: 0.9184
Validation set mean accuracy: 0.8969
Testing set mean accuracy: 0.9056
***** Results of our logistic regression model trained on digits greater or less
than 5 dataset *****
         optimizer_type=gradient_descent
         alpha=0.0001
         eta_decay_factor=None
         beta=None
         batch_size=None
```

```
         T=10000
Step=1000  Loss=0.42243358489012917
Step=2000  Loss=0.365406449561807
Step=3000  Loss=0.3396825736973276
Step=4000  Loss=0.32429855760286636
Step=5000  Loss=0.31376168287040695
Step=6000  Loss=0.3059655126049028
Step=7000  Loss=0.2999083516517843
Step=8000  Loss=0.2950429018696543
Step=9000  Loss=0.29103962114440135
Step=10000  Loss=0.2876853666569619
Total training time: 0.539896 seconds
Training set mean accuracy: 0.9017
Validation set mean accuracy: 0.8858
Testing set mean accuracy: 0.9083

Performance is acceptable.

***** Results of our logistic regression model trained on digits greater or less
than 5 dataset *****
         optimizer_type=momentum_gradient_descent
         alpha=0.0001
         eta_decay_factor=None
         beta=0.9
         batch_size=None
         T=10000
Step=1000  Loss=0.42381913380425984
Step=2000  Loss=0.36727560221534
Step=3000  Loss=0.34173251868892324
Step=4000  Loss=0.3262775582950837
Step=5000  Loss=0.31556929925703303
Step=6000  Loss=0.30757381441809944
Step=7000  Loss=0.301320652013386
Step=8000  Loss=0.29627491499150266
Step=9000  Loss=0.2921109855401281
Step=10000  Loss=0.28861588924500164
Total training time: 0.555441 seconds
Training set mean accuracy: 0.8998
Validation set mean accuracy: 0.8858
Testing set mean accuracy: 0.9056

Performance is acceptable.

***** Results of our logistic regression model trained on digits greater or less
than 5 dataset *****
         optimizer_type=stochastic_gradient_descent
         alpha=0.01
         eta_decay_factor=0.95
```

```
         beta=None
         batch_size=32
         T=10000
Step=1000  Loss=0.4381091189518277
Step=2000  Loss=0.4279522537459438
Step=3000  Loss=0.4223388392140144
Step=4000  Loss=0.4185545727303249
Step=5000  Loss=0.4157515289069932
Step=6000  Loss=0.41352644623049184
Step=7000  Loss=0.4117023283100634
Step=8000  Loss=0.41013140564400696
Step=9000  Loss=0.4087859318084856
Step=10000  Loss=0.40759377410954245
Total training time: 0.879757 seconds
Training set mean accuracy: 0.8646
Validation set mean accuracy: 0.8384
Testing set mean accuracy: 0.8639

Performance is acceptable.

***** Results of our logistic regression model trained on digits greater or less
than 5 dataset *****
         optimizer_type=momentum_stochastic_gradient_descent
         alpha=0.1
         eta_decay_factor=0.95
         beta=0.9
         batch_size=32
         T=10000
Step=1000  Loss=0.3012558887703501
Step=2000  Loss=0.29622609716718573
Step=3000  Loss=0.29323566830754894
Step=4000  Loss=0.29145839529143164
Step=5000  Loss=0.29021587642423935
Step=6000  Loss=0.28924456788000136
Step=7000  Loss=0.288391038255442
Step=8000  Loss=0.2877133912595621
Step=9000  Loss=0.2871702117042914
Step=10000  Loss=0.2866464690253258
Total training time: 0.897409 seconds
Training set mean accuracy: 0.9035
Validation set mean accuracy: 0.8802
Testing set mean accuracy: 0.9028

Performance is acceptable.


Preprocessing the fir and pine coverage dataset (495141 samples, 54 feature
dimensions)
```

```
***** Results on the fir and pine coverage dataset using scikit-learn logistic
regression model *****
Training set mean accuracy: 0.7515
Validation set mean accuracy: 0.7547
Testing set mean accuracy: 0.7504
***** Results of our logistic regression model trained on fir and pine coverage
dataset *****
         optimizer_type=stochastic_gradient_descent
         alpha=0.001
         eta_decay_factor=0.99
         beta=None
         batch_size=1000
         T=3000
Step=1000  Loss=1.9366960232466492
Step=2000  Loss=0.5910159502489213
Step=3000  Loss=0.5636007001792679
Total training time: 164.761738 seconds
Training set mean accuracy: 0.7111
Validation set mean accuracy: 0.7139
Testing set mean accuracy: 0.7096

Performance is acceptable.

***** Results of our logistic regression model trained on fir and pine coverage
dataset *****
         optimizer_type=momentum_stochastic_gradient_descent
         alpha=0.01
         eta_decay_factor=0.99
         beta=0.9
         batch_size=1000
         T=3000
Step=1000  Loss=1.004568538582225
Step=2000  Loss=0.8951011591354504
Step=3000  Loss=0.8345623921460444
Total training time: 163.795974 seconds
Training set mean accuracy: 0.7202
Validation set mean accuracy: 0.7220
Testing set mean accuracy: 0.7180

Performance is acceptable.
```

[ ]:

[ ]: