HW3_programQuestion

October 14th, 2020

1 HW3_programQuestion

Due to 11:59 pm, 28th, October 2020

This is an individual assignment.

```
In [ ]: """
        Import libraries that you might require.
        """

        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.metrics import mean_squared_error, accuracy_score
        from sklearn.linear_model import LogisticRegression
```

1.1 Logistic Regression and Gradient Descent

In this question, we will try to use logistic regression to solve a binary classification problem. Given some information of the house, such as area and the number of living rooms, would it be expensive? We would like to predict 1 if it is expensive, and 0 otherwise.

We will first implement it with a python package, and then try to implement it by updating weights with gradient descent.Batch gradient descent (since we are using all samples at each iteration) and AdaGrad will be implemented. We will also derive the gradient formula.

1.1.1 a) Implement logistic regression with Scikit learn package.

First load data and observe data.

```
In [ ]: """
        Reads the data.
        """

        X_train = pd.read_csv('hw3_house_sales/X_train.csv')
        X_test = pd.read_csv('hw3_house_sales/X_test.csv')
        y_train = pd.read_csv('hw3_house_sales/y_train.csv')
        y_test = pd.read_csv('hw3_house_sales/y_test.csv')
        print(X_train.shape)
        print(X_test.shape)
        print(y_train.shape)
        print(y_test.shape)

        print(X_test.head(5))
        print(y_test.head(5))
```

Fill in the logisticRegressionScikit() function. Report the weights, training accuracy, and the test accuracy.

```
In [ ]: def LogisticRegressionScikit(X_train, y_train, X_test, y_test):
            """
            Computes logistic regression with scikit-learn.

            Args:
                X_train: feature matrix of training set
                y_train: truth value of training set
                X_test: feature matrix of test set
                y_test: truth value of test set

            Returns:
                w: numpy array of learned coefficients
                y_pred: numpy array of predicted labels for the test data
                score: accuracy of test data
            """
            return coef, y_pred, score
```

```
In [ ]: coef_scikit, y_pred_scikit, acc_scikit = LogisticRegressionScikit(X_train, y_train, X_

        print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(acc_scik
        print('logistic regression coefficient:', coef_scikit)
```

1.1.2 b) Gradient derivation

Calculate the maximum likelihood estimation $L(w) = P(Y|X; w)$, then formulate the stochastic gradient ascent rule.

1.1.3 c) Logistic regression with simple gradient descent

Fill in the LogisticRegressionSGD() function. To do that, two helper functions, sigmoid_activation() (to calculate the sigmoid function result), and model_optimize() (to calculate the gradient of w), will be needed. Both helper functions can be used in the following AdaGrad optimization function. Use a learning rate of 10^4, run with 2000 iterations. Report the weights and accuracy. Keep track of the accuracy every 100 iterations in the training set. It will be used later.

```
In [ ]: def sigmoid_activation(x):
            """
            Calculates the sigmoid function.
```

```
        Args:
            x: numpy array of input

        Returns:
            final_result: numpy array of sigmoid result
        """
        return final_result
```

Remember to derive the gradient (Question 4.2), write down the weight update formula, and hand it in with your latex submission!

```
In [ ]: def model_optimize(w, X, Y):
            """
            Calculates gradient of the weights.

            Args:
                X: numpy array of training samples
                Y: numpy array of training labels
                w: numpy array of weights
            Returns:
                dw: the gradient of the weights

            """
            return dw
```

```
In [ ]: def LogisticRegressionSGD(w, X, Y, learning_rate, num_iterations):
            """
            Uses SGD to update weights for logistic regression.

            Args:
                w: numpy array of initial weights
                X: numpy array of training samples
                Y: numpy array of training labels
                learning_rate: float learning rate to update w
                num_iterations: int number of iterations to update w

            Returns:
                coeff: numpy array of weights after optimization
                accuracies: a list of accuracy at each hundred's iteration. With 2000 iteratio
                            accuracies should be a list of size 20
            """

            return coeff, accuracies
```

1.1.4 d) Logistic regression with AdaGrad

Fill in the LogisticRegressionAda() function. Use a learning rate of 10^4, run with 2000 iterations. Report the weights and accuracy. Keep track of the accuracy every 100 iterations in the training set. It will be used later.

```
In [ ]: def LogisticRegressionAda(w, X, Y, learning_rate, num_iterations):
            """
            Use AdaGrad to update weights.

            Args:
                w: numpy array of initial weights
                X: numpy array of training samples
                Y: numpy array of training labels
                learning_rate: float learning rate to update w
                num_iterations: int number of iterations to update w

            Returns:
                coeff: numpy array of weights after optimization
                accuracies: a list of accuracy at each hundred's iteration
            """
            accuracies = []

            return coeff, accuracies
```

We add a predict() function here to threshold probability prediction into binary classification

```
In [ ]: def predict(final_pred, m):
            """
            Predict labels from probability to 0/1 label, threshold 0.5.

            Args:
                final_pred: m x 1 vector, probabilty of each sample belonging to class 1
                m: number of samples

            Returns:
                y_pred: m x 1 vector, label of each sample, can be 0/1
            """

            return y_pred
```

Now we start to use our dataset and construct model.

```
In [ ]: # Do some data preparation, convert dataframe to numpy array
        n_features = X_train.shape[1]

        w = np.zeros((1, n_features))

        X_train = X_train.values
        X_test = X_test.values

        y_train = y_train.values
        y_test = y_test.values
```

```
m_train =  X_train.shape[0]
m_test =  X_test.shape[0]
```

Model construction for SGD logistic regression.

```
In [ ]: #Gradient Descent
        coeff_SGD, acc_SGD = LogisticRegressionSGD(w, X_train, y_train, learning_rate=0.0001,nu

        # TODO: predict probability
        final_train_pred_SGD = ...
        final_test_pred_SGD = ...
        # predict label
        y_train_pred_SGD = predict(final_train_pred_SGD, m_train)
        y_test_pred_SGD = predict(final_test_pred_SGD, m_test)

        print('Optimized weights for SGD', coeff_SGD[:-1])
        print('Optimized intercept for SGD', coeff_SGD[-1])

        print('Training Accuracy for SGD', accuracy_score(y_train_pred_SGD.T, y_train))
        print('Test Accuracy for SGD', accuracy_score(y_test_pred_SGD.T, y_test))
```

Model construction for AdaGrad logistic regression.

```
In [ ]: #AdaGrad Descent
        coeff_Ada, acc_Ada = LogisticRegressionSGD(w, X_train, y_train, learning_rate=0.0001,nu

        # TODO: predict probability
        final_train_pred_Ada = ...
        final_test_pred_Ada = ...
        # predict label
        y_train_pred_Ada = predict(final_train_pred_Ada, m_train)
        y_test_pred_Ada = predict(final_test_pred_Ada, m_test)

        print('Optimized weights for Ada', coeff_Ada[:-1])
        print('Optimized intercept for Ada', coeff_Ada[-1])

        print('Training Accuracy for Ada', accuracy_score(y_train_pred_Ada.T, y_train))
        print('Test Accuracy for Ada', accuracy_score(y_test_pred_Ada.T, y_test))
```

Plot accuracy vs iteration for SGD and AdaGrad. Compare the performance difference. Briefly explain the reason.

```
In [ ]: # Plot accuracy vs iteration for SGD and AdaGrad

        plt.plot(acc_SGD, label='SGD')
        plt.plot(acc_Ada, label='AdaGrad')
        plt.ylabel('Accuracy')
        plt.xlabel('iterations (per hundreds)')
```

```
    plt.title('Accuracy improvement over time')
    plt.legend(loc='lower right')
    plt.show()
```

1.1.5 e) Comparision of Scikit, SGD and AdaGrad convergence

Plot the loss function of SGD and AdaGrad over 2000 iterations on both the training and test data. What do you observe? Which one has better accuracy on the test dataset? Why might that be the case?