CS 334: Homework #3 Solutions

1. **Feature Extraction and Feature Selection**: The full code can be viewed in `selFeat.py`

   (a) There are a variety of features that can be extracted from the date and timestamp. For the solutions, we will extract the hour and the day of the week (i.e., Monday, Tuesday, Wednesday) as well as whether it is morning, afternoon, or sleeping time.

```
def is_morning(row):
    # calculate morning or not
    if row['hour'] > 5 and row['hour'] <= 12:
        return 1
    return 0

def is_afternoon(row):
    # calculate afternoon or not
    if row['hour'] > 12 and row['hour'] <= 18:
        return 1
    return 0


def is_sleep(row):
    # calculate afternoon or not
    if row['hour'] > 22 or row['hour'] <= 5:
        return 1
    return 0


def extract_features(df):
    # convert it to date
    df['date'] = pd.to_datetime(df['date'])
    # create an hour feature
    df['hour'] = df.date.dt.hour
    df['morning'] = df.apply(lambda row: is_morning(row), axis=1)
    df['afternoon'] = df.apply(lambda row: is_afternoon(row), axis=1)
    df['sleep'] = df.apply(lambda row: is_sleep(row), axis=1)
    # extract day of the week
    df['dow'] = df.date.dt.dayofweek
    df = df.drop(columns=['date'])
    return df
```

   (b) The heatmap of the Pearson correlation between the features and the response is shown in Figure 1. The code to calculate the correlation and plot it is shown below:

```
df = pd.read_csv("eng_xTrain.csv")
# do the feature extraction
df = selFeat.extract_features(df)
# merge xfeat with y
y = pd.read_csv('eng_yTrain.csv')
df['y'] = y

# calculate the correlation and perform the heatmap
corrMat = df.corr()
snsPlot = sns.heatmap(corrMat)
snsPlot.figure.savefig('corr_heatmap.eps', format='eps')
```

   (c) Based on the heatmap and the correlation matrix, any two features whose magnitude of the correlation was above the threshold 0.5 was removed (and reduced to a single feature).

   (d) For preprocessing data, we can choose MinMax to standardize the data.

```
def select_features(df):
    """
```

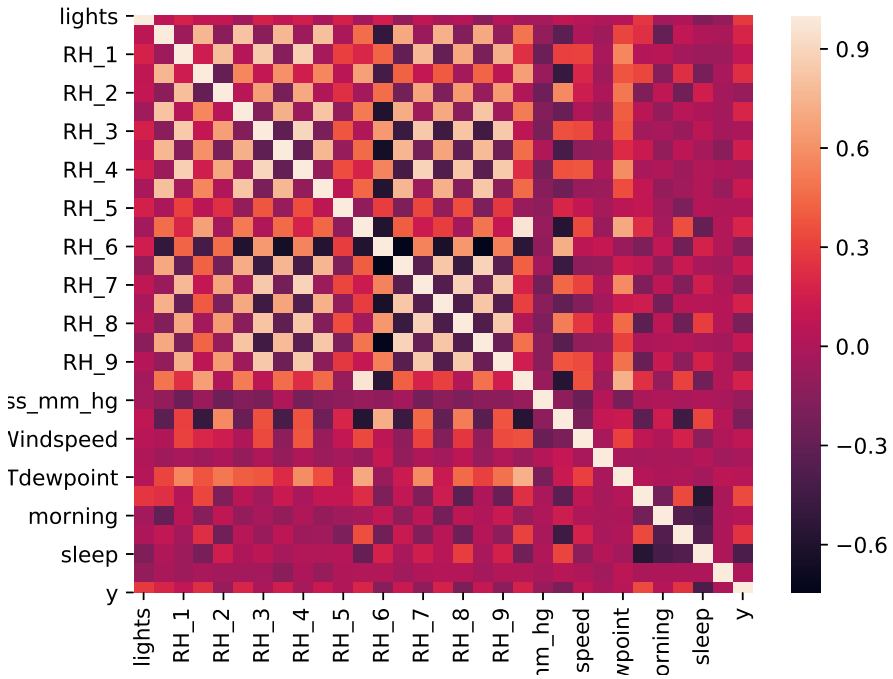

Figure 1: Correlation heatmap

```
    Select the features to keep
    """
    df.drop(columns=["T1", "RH_1", "RH_2",
                     "T3", "RH_3", "T4",
                     "RH_4", "T5", "RH_5",
                     "RH_6", "T7", "RH_7", "RH_9",
                     "T9", "T_out", "Press_mm_hg",
                     "Windspeed", "Visibility", "Tdewpoint",
                     "dow", "morning"],
            inplace=True)
    return df


def preprocess_data(trainDF, testDF):
    # standardize the two
    scaler = preprocessing.MinMaxScaler()
    scaler.fit(trainDF)
    trainTr = pd.DataFrame(scaler.transform(trainDF),
                           columns=trainDF.columns)
    testTr = pd.DataFrame(scaler.transform(testDF),
                           columns=testDF.columns)
    return trainTr, testTr
```

2. **Linear Regression: Single Unique Solution**: The full code can be viewed in `lr.py`, `standardLR.py`

   (a) The prediction can be calculated as $\hat{\mathbf{y}} = \mathbf{X}\beta$

```
    def predict(self, xFeat):
        """
        Given the feature set xFeat, predict
        what class the values will have.
```

```
        Parameters
        ----------
        xFeat : nd-array with shape m x d
            The data to predict.

        Returns
        -------
        yHat : 1d array or list with shape m
            Predicted response per sample
        """
        return np.matmul(xFeat, self.beta)
```

(b) The unique solution for multivariate linear regression is: $\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$

```
    def train_predict(self, xTrain, yTrain, xTest, yTest):
        """
        See definition in LinearRegression class
        """
        trainStats = {}
        startTime = time.time()
        temp = np.ones((len(xTrain),1), dtype=int)
        Int = np.ones((len(xTest),1),dtype = int)
        xTest = np.concatenate((xTest,Int),axis =1)
        xTrain = np.concatenate((xTrain, temp), axis=1)
        xt = np.transpose(xTrain)
        # calculate (X^T)X
        xtx = np.matmul(xt, xTrain)
        # calculate the inverse of xtx
        xtxi = np.linalg.inv(xtx)
        # xty
        xty = np.matmul(xt, yTrain)
        self.beta = np.matmul(xtxi, xty)
        trainMse = self.mse(xTrain, yTrain)
        testMse = self.mse(xTest, yTest)
        elapsedTime = time.time() - startTime
```

3. Linear Regression using SGD

(a) For the `train_prediction` function, the key is to sort it and keep track of the batch size. The code assumes the batch size splits the dataset evenly (in other words the batch sizes aren't partial).

```
    def train_predict(self, xTrain, yTrain, xTest, yTest):
        # randomly initialize the weights
        self.beta = np.random.rand(xTrain.shape[1],1)
        # set things up quickly
        trainStats = {}
        n = xTrain.shape[0]
        # get the number of batches
        nBatch = int(float(n) / self.bs)
        startTime = time.time()
        for epoch in range(self.mEpoch):
            # shuffle the dataset
            idx = np.random.permutation(n)
            # iterate through each batch
            for i in range(nBatch):
                bIdx = idx[i*self.bs:(i+1)*self.bs]
                grad = grad_pt(self.beta, xTrain[bIdx], yTrain[bIdx]) / self.bs
                self.beta = self.beta + self.lr*grad
                grad = np.zeros(xTrain.shape[1])
                trainMse = self.mse(xTrain, yTrain)
                testMse = self.mse(xTest, yTest)
                elapsedTime = time.time() - startTime
```

```
trainStats[epoch*nBatch + i] = {'time':elapsedTime,
                                'train-mse':trainMse,
                                'test-mse':testMse}
```

(b) In Figure. 2, we can see learning rate = 0.001 should give the best MSE. The full code can be viewed in `q3b.py`

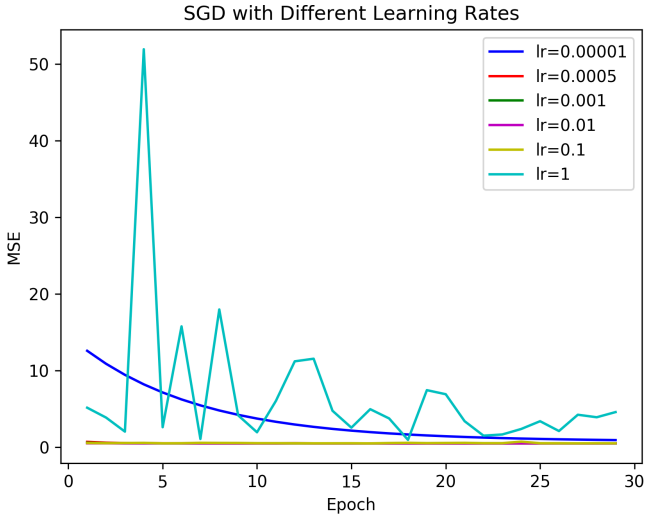

Figure 2: SGD with Different Learning Rates

(c) In Figure 3, we can see the test mse drops along with the increment of the epoch number. The full code can be viewed in `q3c.py`

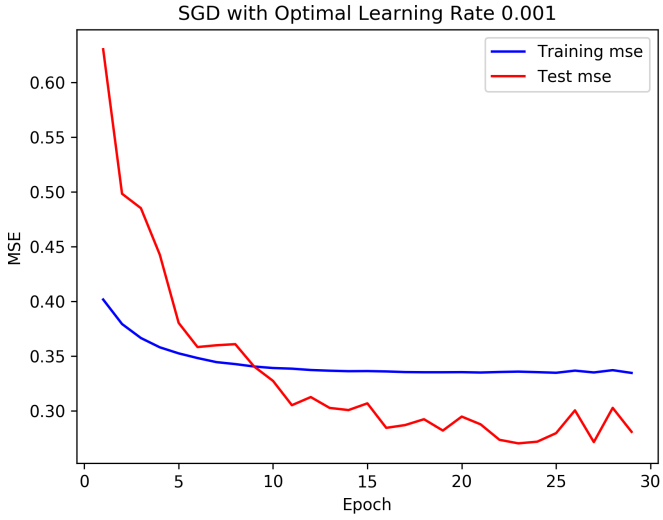

Figure 3: Correlation heatmap

4. **Comparison of LR with SGD and Closed form**

(a) For each batch size (1, 86, 215, 5990, 16770), we will test it on these learning rate (1, 0.1, 0.001, 0.0005). The results are shown in Figure 4 5 6. The full code can be viewed in `q4lr2bs.py`.

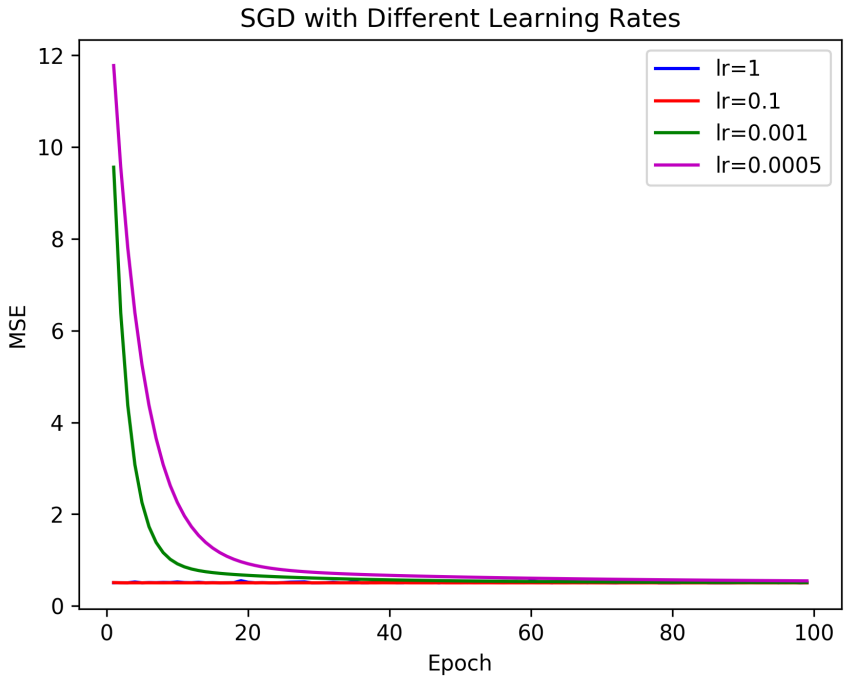

Figure 4: Batch size is 86

After choosing best learning rate for each batch size, we can obtain the graphs on mse vs time for different batch size. The results are shown in Figure 7 8. The full code can be viewed in `q4bs2ti.py`.

(b) Possible illustrations.

From Figure 7 and Figure 8, we can see the smaller the batch size is, the longer the process takes. The results of Closed Form Solution is better than those of all SGD solutions. Smaller batch size can achieve better results. In future, according to the requirements of the problem, we could choose different configurations to solve it.

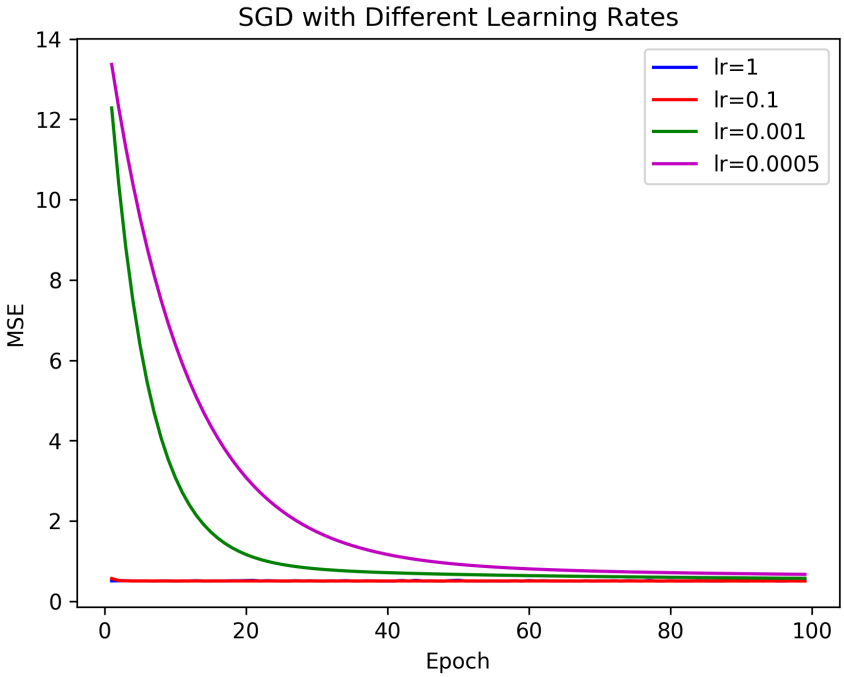

Figure 5: Batch size is 215

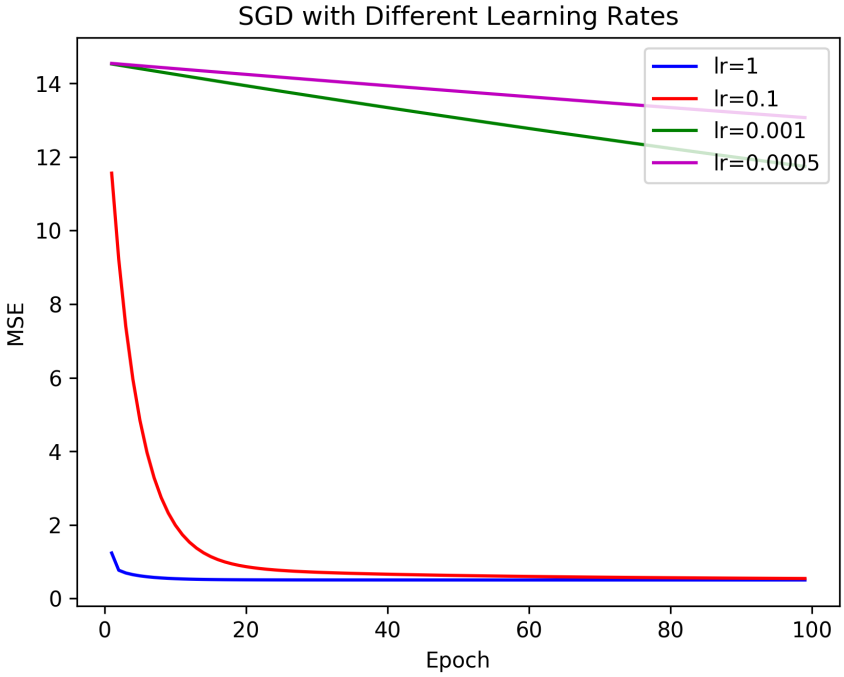

Figure 6: Batch size is 16770

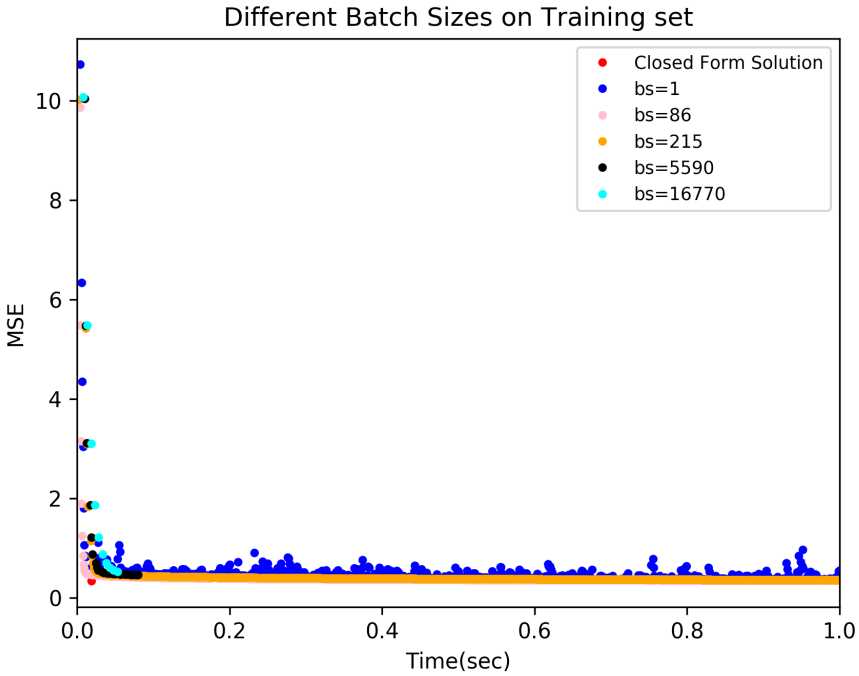

Figure 7: Different Batch Sizes on Training Set

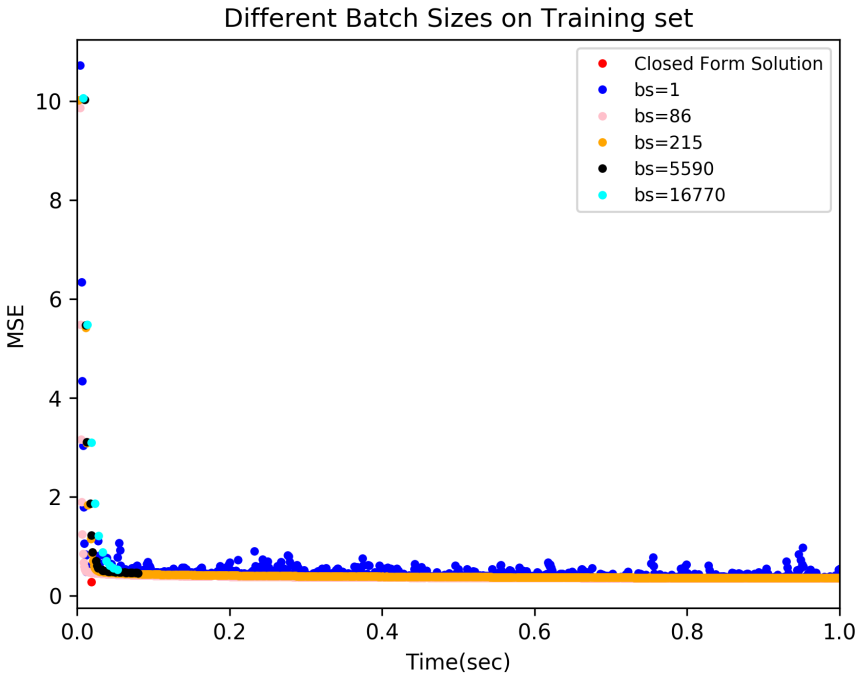

Figure 8: Different Batch Sizes on Training Set