Linear Regression - Forest Fires

Subhendu Mishra

September 22, 2021

1. Linear Regression

The closed form/analytical solution and an iterative solution using minibatch gradient descent for **L2**-regularized linear regression is implemented that minimizes the mean squared error **(MSE)** function. For a data set, $\mathbf{D} = \{(x_i, y_i)\}_{i=1}^{N}$, of **N** training samples where each $x_i \in R^d$, the linear regression model is defined as:

$$f_w(x) = w^T x$$

and the batch gradient descent weight update rule for the k^{th} element in $\mathbf{w}$, w^k, is:

$$w_k = w_k - \eta \frac{1}{N} \Sigma_{i=1}^{N} (f_w(x_i) y_i) x_i, k$$

where η is the learning rate and x_i, k is the value of the k^{th} dimension in x_i.

2. Gradient descent stopping criteria

The stopping criteria used in Gradient Descent is the average MSE loss over both the training set and the validation set.

3. Effect of regularization:- Setting C to 0 in do_ gradient _ descent yields the unregularized least squares solution. MSE on the dev.set instances for different values of C (including C=0) is plotted in the following manner -

train_features, train_targets = get_features('train.csv',True), get_targets('train.c

dev_features, dev_targets = get_features('dev.csv',False), get_targets('dev.csv')

dev_loss=do_evaluation(dev_features, dev_targets, a_solution)

train_loss=do_evaluation(train_features, train_targets, a_solution)

C = np.zeros((20))

loss = np.zeros((20))

for i in range(0,20):

print('training LR using gradient descent...')

gradient_descent_soln = do_gradient_descent(train_features,

train_targets,

dev_features,

dev_targets,

$1e^{-2}$,

C[i],

32,

39000,

100)

print('evaluating iterative_solution...')

dev_loss=do_evaluation(dev_features, dev_targets, gradient_descent_soln)

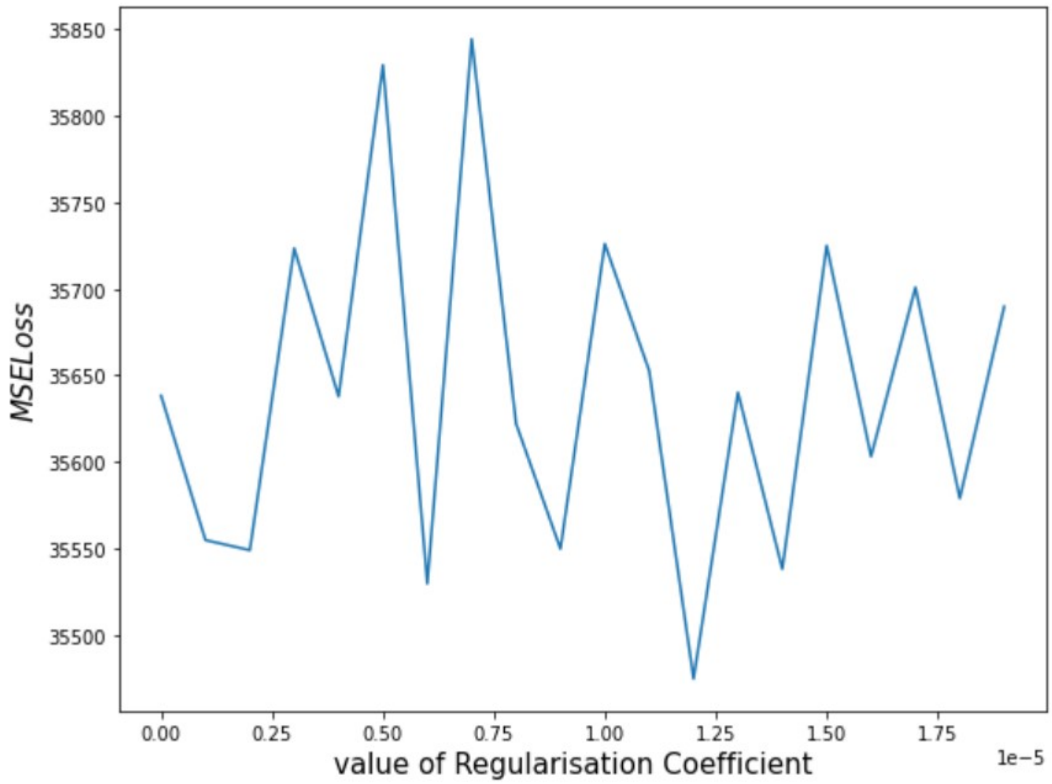
train_loss=do_evaluation(train_features, train_targets,

gradient_descent_soln)

print('gradient_descent_soln $\frown$train loss: , dev_loss: '.format(train_loss,dev_loss))

loss[i] = dev_loss

C[i+1] = C[i] + 0.000001



4. Basis functions :-

Two different basis functions are implemented that will be applied to input features with the L2-regularized model and optimized using gradient descent. and the **MSE** is calculated on the development set samples using both basis functions.

Basis function

$$\phi_(x_1) = x^2$$
$$\Phi_(x_2) = x^4$$

For no basis function :

$$\text{basis1} = \text{False}$$
$$\text{basis2} = \text{False}$$
$$\text{MSE Loss} = 38933.769$$

For Basis Function 1 :

$$\text{basis1} = \text{True}$$
$$\text{basis2} = \text{False}$$
$$\text{MSE Loss} = 35088.528$$

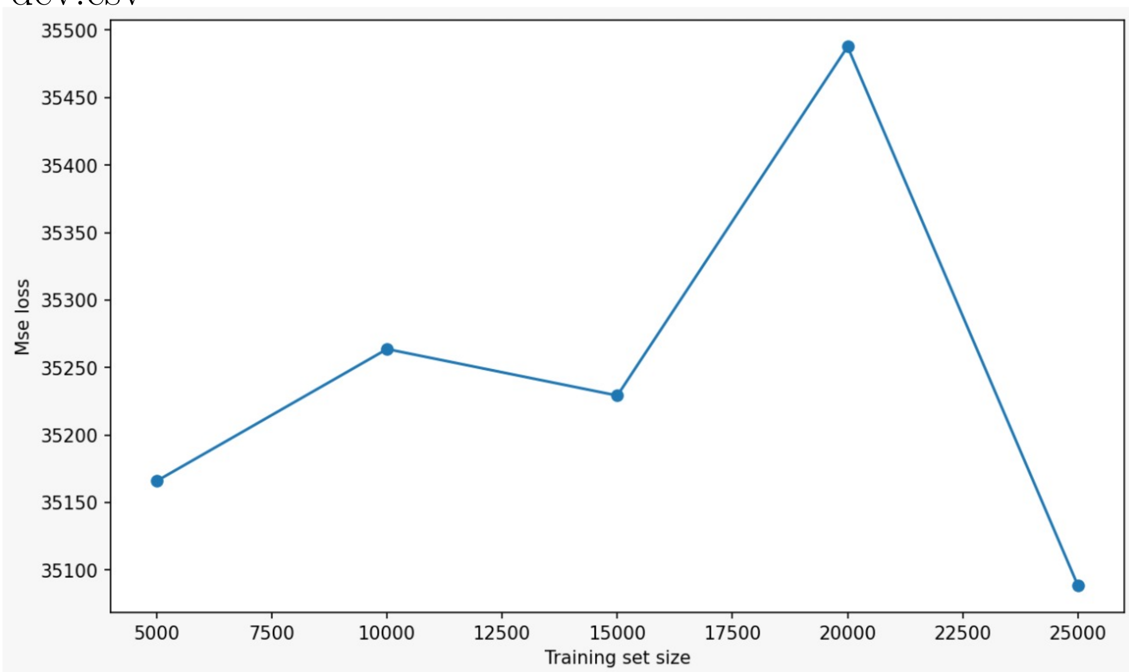
For Basis Function 2 :

$$\text{basis1} = \text{False}$$
$$\text{basis2} = \text{True}$$
$$\text{MSE Loss} = 38083.768$$

Under get_features subroutine we can set basis1 = True for using $\phi(x_1)$ and we can set basis2 = True for using $\phi(x_2)$ as our basis function.

5. Training Plots :-

Plot containing Training set size in X-axis vs. MSE loss on the Y-axis of dev.csv



.

6.Feature Importance:-

The most important feature is the "brightness" as this has the highest weight of all. Some features are not changing in the test data like "instrument" and "version". However, to invert the matrix we found the co-variance of each feature with respect to others. The highly co-related features were removed. The "scan" and "track" were highly co-related. Thus one of these features could be considered least important