Privacy secure stochastic gradient descent

Revar Vladimir
Optimization Class Project. MIPT

Introduction

Nowdays secure is not really defended. Many algorithms do not think about using some modifications to prevent a leak of data, that's why I tested a modification of SGD that change original dataset by adding some noise vectors for every string. This vector has a normal uniform distribution from -1 to 1. The idea of this method is to find a vector of weights that is the best for linear regression. Also this found weights is compared with analytical solution.

Stochastic gradient descent

One approach for minimization function is stochastic gradient descent:

minimize $L = \sum_{i=1}^{N} (Y_i - Y_{pred})^2$

where minimization is provided by movement to the direction of

$$w := w - \frac{1}{n} * \theta \sum_{1}^{n} f_w$$

and $Y_{pred} = X * w$, where X is our dataset and w is weights vector.

- the idea to find derivative L_w

Derivative of loss function

I used a mean squared error $MSE = \sum_{i=1}^{dimX} (Y - X@w)^2$

minimize $\sum_{i=1}^{dimX} (Y - X@w)^2$

- $(\sum_{i=1}^{dimX} (Y - X@w)^2)' = \sum_{1}^{dimX} ((Y_i - \sum_{1}^{dimX^*} x_{ij} * w_i)^2)' =$
- $2 \sum_{1}^{dimX} x_{ji} (\sum_{1}^{dimX^*} x_{ij} * w_i - Y_i) =$
- $2X^T \sum_{1}^{dimX} (X@w - Y)$
- Where $dimX$ is number of strings
- And $dimX^*$ is a number of rows

Algorithm

We solve this problem using an Stochastic gradient descent:

```
X_train, X_test, Y_train, Y_test = train_test_split(X_b, y,
 test_size = 0.2, random_state = 23)

def generate_theta(x):
    return np.random.rand(x.shape[1],1)

def add_some_noise(x):
    for j in range(x.shape[1]):
    x[j,:] += np.random.uniform(-1, 1, 14)
    return x

def sgd(x,y,alpha,w,iterations):
    save_w = w
    previous_mse = mse(x@w, y)
    mse_array = []
    iterations_array = []
    for i in range(1, iterations):
        mse_array.append(mse(x@w, y))
        iterations_array.append(i)
        if i % 10000 == 0:
        current_mse = mse(x@w, y)
        if current_mse > previous_mse:
        print('curr > prev', current_mse)
        alpha /= 2
        w = save_w
        if 0 <= previous_mse - current_mse <= 10**-3:
            print('alpha is ', alpha)
            alpha *= 2
        print('iter is {} and mse is {}'.format(i, mse(x@w, y)))
        save_w = w
        previous_mse = current_mse
    w -= alpha * sgd_gradient(x,y,w)
    return w, mse_array, iterations_array
```

Normalize dataset

For normal convergence we should normalize our dataset by using algorithm

```
norm_df = df.apply(lambda x: (x - np.mean(x)) / (np.max(x) - np.min(x)))
```

Convergence proof

To prove that this modifacition of stochastic gradient descent convergse to a minimum, we compare a result with analitical solution that is pseudo-inverse matrix $(X^T X)^{-1} X^T Y$

Numerical example

Consider a numerical example with $L(w) = \sum_{1}^{dimX} (Y - X@w)^2$ with $X \in \mathbf{R}^{404 \times 14}$ and $Y \in \mathbf{R}^{404}$. Entries of X and Y are generated as tested matrix. Here, we have chosen w using sgd.

Results

On this numerical example, modification of sgd has a really awesome convergense that is really close to analytical solution

```
Mse for simple sgd is 0.018235973074556364
Mse for modificated sgd is 0.013209349763651826
Mse for analytical solution is 0.012332762679044384
```

Conclusion

In this project were compared results of classic sgd and modificated sgd on 500000 iterations to convergence and noise dataset has a less loss function than original algorithm. Also as noise vector is chosen randomly it ensures privacy of original data.

Acknowledgements

This material is based upon work written by Reid M. Bixler
Computer Science Dept. Virginia Tech Blacksburg, VA 24060.