Kaggle

November 5, 2020

1 Fashon-MNIST

You can find my notebook published on https://www.kaggle.com/dannkim/fashon-mnist-classification-from-scratch Classification of Fashion MNIST data set, from scratch using only numpy. And comparison of the implemented model with sklearn.linear_model.LogisticRegression

[3]:
```
import numpy as np
import pandas as pd
```

1.1 Data Preparation

- Download the data set from https://www.kaggle.com/zalando-research/fashionmnist
- Load it using pandas
- Store the pixels values into X and labels into Y
- Normalize the data dividing by 255

[4]:
```
data_train = pd.read_csv('./data/fashion-mnist_train.csv')
data_test = pd.read_csv('./data/fashion-mnist_test.csv')
```

[5]:
```
X_train = data_train.values[:, 1:]
Y_train = data_train.values[:, 0]
X_test = data_test.values[:, 1:]
Y_test = data_test.values[:, 0]
```

[6]:
```
X_train, X_test = X_train.astype('float32'), X_test.astype('float32'),
X_train /= 255
X_test /= 255
```

1.2 Logistic Regression from Scratch

1.2.1 log_regression(X, Y, T, learning_rat)

- The log_regression() function optimizes the weights by minimizing a Cross Entropy Error using stochastic gradient descent
- X is a training data set
- Y is corresponding labels
- T is a number of iterations (default is 50000)

- learning_rate is a learning rate of the model (default is initially 0.01 and decreasing during the training
- Final weights are returned

[7]:
```
def log_regression(X, Y, T=50000, learning_rate=0.01):
    N = X.shape[0]
    W = np.zeros((X.shape[1],))
    for i in range(T):
        if i == 20000:
            learning_rate = 0.005
        elif i == 30000:
            learning_rate = 0.001
        rand_index = np.random.choice(N, size=1)
        x_n = X[rand_index][0]
        E_dev = -1 * (Y[rand_index] * x_n)/(1 + np.exp((Y[rand_index][0] * np.
↪dot(x_n, W))))
        W = W - learning_rate * E_dev
    return W
```

1.2.2 generate_D_Y(Y, k)

- The model uses One vs All approach for multicalss classificattion
- The generate_D_Y() function generates new labels for each class
- Y is an original set labels
- k is the number of classes

[8]:
```
def generate_D_Y(Y, k):
    Y_copy = np.copy(Y)
    for i, y in enumerate(Y_copy):
        Y_copy[i] = 2 * (int(y) == k) - 1
    return Y_copy
```

1.2.3 sigmoid(x)

- The sigmoid function is used to predict the probability that x belongs to the class

[9]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

1.2.4 calculate prob(W, X)

- The calculate_prob() function calculate the probability of each example belongs to each class
- W is the set of weights genetared by log_regression on each class k
- X is a test data set

[10]:
```
def calculate_prob(W, X):
    probs = []
    for i in range(10):
```

```
        probs.append(sigmoid(np.dot(X, W[i])))
    return probs
```

1.2.5 predict(p)

- The predict() function predicts the actual class by getting the most probable one
- p is probabilities calculated by calculate_prob()

[11]:
```
def predict(p):
    Y_pred = []
    for i in range(len(p[0])):
        Y_pred.append(np.argmax(p[:,i]))
    return Y_pred
```

1.2.6 calculate_accuracy(Y_pred, Y)

- The calculate-accuracy function evaluates the accuracy of the model by comparing predicted values Y_pred with actual labels Y

[12]:
```
def calculate_accuracy(Y_pred, Y):
    correct = 0
    for i in range(len(Y)):
        if Y_pred[i] == Y[i]:
            correct += 1
    return correct/len(Y)
```

1.2.7 Logistic Regression

[13]:
```
class logistic_regression:
    def fit(X, Y, k):
        W = []
        for i in range(k):
            W.append(log_regression(X, generate_D_Y(Y, i)))
        return W

    def score(X, Y, W):
        probs = calculate_prob(W, X)
        Y_pred = predict(np.array(probs))
        return calculate_accuracy(Y_pred, Y)
```

1.3 Training the Implemented Logisitic Regression

[14]:
```
classifier = logistic_regression
W_log = classifier.fit(X_train, Y_train, 10)
```

1.4 Testing the Implemented Logisitic Regression

[48]:
```
print(classifier.score(X_test, Y_test, W_log))
```

```
0.8323
```

1.5 Training the sklearn.linear_model.LogisticRegression

[21]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0, solver='liblinear', multi_class='auto').
 ↪fit(X_train, Y_train)
```

1.6 Testing the sklearn.linear_model.LogisticRegression

[23]:
```
print(clf.score(X_test, Y_test))
```

```
0.855
```

1.7 Summary

It can be seen that the performance of my model is approximately the same as the performance of sklearn build-in model. Therefore it can be concluded that the implementation was done correctly.