Hand-in 1

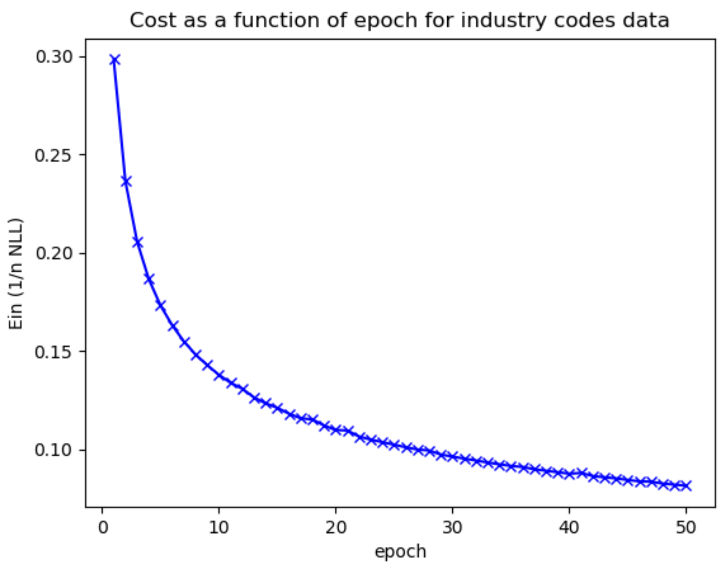
Mads-Peter Verner Christiansen, au616397
Zeyuan Tang, au597881
Douwe Tjeerd Schotanus, au600876

September 28, 2018

1 Part I: Logistic regression

1.1 Code

1.1.1 Summary and results



In sample score achieved for the run in the figure was: 0.97 and the test score was 0.96. The plot shows that cost decreases rapidly asymptotically approaching zero at high epochs.

1.1.2 Code snippets

The gradient and the cost for logistic regression are calculated as follows

```
cost = -(y @ np.log(logistic(w @ X.T)) + (1 - y) @ np.log(1 - logistic(w @ X.T)))
    grad = -X.T @ (y - logistic(X @ w))
```

```
        n, _ = X.shape
        cost /= n; grad /= n
```

The gradient-descent fitting algorithm is shown below.

```
    if w is None: w = np.zeros(X.shape[1])
        history = []
        for i in range(epochs):
            print('epochs {} j {}'.format(i, int(X.shape[0]/batch_size)))
            random_indices = np.random.permutation(X.shape[0])
            X, y = X[random_indices], y[random_indices]
            for j in range(int(X.shape[0]/batch_size) - 1):
                Xj = X[j*batch_size:(j+1)*batch_size]
                yj = y[j*batch_size:(j+1)*batch_size]
                #loss = (Xj.T @ Xj @ w - Xj.T @ yj) * 2 / batch_size
                cost, grad = self.cost_grad(Xj, yj, w)
                w = w - lr * grad
            cost, grad = self.cost_grad(X, y, w)
            history.append(cost)
        ### END CODE
        self.w = w
        self.history = history
```

1.2 Theory

1.2.1 Question 1: Running time

The run time of the the cost and the gradient are given by:

$$t_{cost} = O(nd) \tag{1}$$

$$t_{gradient} = O(nd) \tag{2}$$

For the mini-batch gradient descent algorithm the total run time is

$$t_{total} = O(epochs \cdot n \cdot d) \tag{3}$$

Because loops 'epochs' times and each iteration it does n/mini_batch_size gradient calculations where the data matrix has shape (mini_batch_size, d)

1.2.2 Question 2: Sanity check

The algorithm would become worse, as locality can no longer be exploited, a random permutation of the pixels will remove locality dependent features. As an example, the distance between the nose and the eyes can no longer be determined, this feature is similar to the symmetry feature of hand-written digits which is also local.

1.2.3 Question 3: Linearly separable data

For linearly separable data all points will be correctly classified. For a single data point that should classified 0 the cost is:

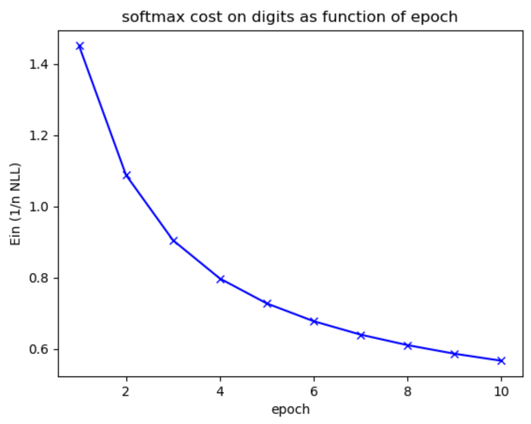
$$NLL = \ln(1 - \sigma(w^T x)) \tag{4}$$

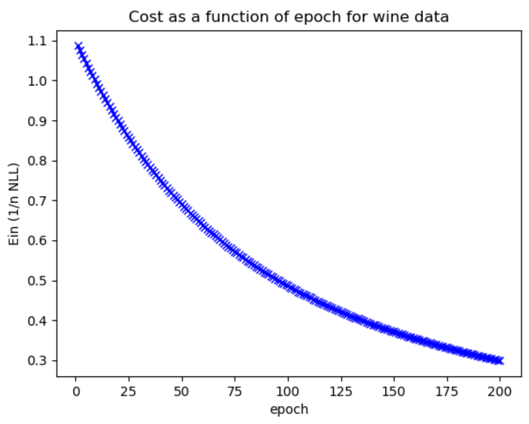
This is minimised if the sigmoid gives zero, which it approaches for large negative numbers, thus increasing the magnitude of the weights will decrease the cost.

2 Part II: Softmax

2.1 Code

2.1.1 Summary and results





For the digits dataset an in-sample score of 0.85 and a test sample of 0.85 was achieved. For the wine data set the in sample score was 1.0 and the test score was 0.90.

2.1.2 Code snippets

The cost-grad function has been implemented as

```
sm = softmax(X@W)
grad = -X.transpose()@(Yk-sm)
grad /= n
# Cost:
_, idx = np.nonzero(Yk)
cost = 0
for i in range(n):
    z = W.transpose()@X[i, :]
    cost -= (z[idx[i]]-np.log(np.sum(np.exp(z))))
cost /= n
```

and the fit function as

```
 n, _ = X.shape
     idxs = np.arange(n)
     num_batches = n//batch_size
     for ep in range(epochs):
         np.random.shuffle(idxs)
         X = X[idxs, :]; Y = Y[idxs]
         batches = [(X[i*batch_size:(i+1)*batch_size], Y[i*batch_size:(i+1)*batch_siz
         for batch in batches:
             cost, grad = self.cost_grad(batch[0], batch[1], W)
             W -= lr*1/batch_size*grad
         self.W = W
         cost, _ = self.cost_grad(X, Y, W)
         history[ep] = cost
     self.history = history
```

2.2 Theory

2.2.1 Running time

For the softmax algorithm the run time of calculating the cost and the gradient are:

$$t_{cost} = O(ndK) \tag{5}$$

$$t_{gradient} = O(ndK) \tag{6}$$

Where the run time of the softmax function has not been included. The gradient-descent algorithm has run time:

$$t_{total} = O(epochs \cdot ndK) \tag{7}$$