Machine Learning WS 19 - Assignment 2

October 31, 2018

Adrian Gruszczynski / Yann Salimi

1 Implementation of Least-Squares Linear Regression

Using the closed-form expression from the lecture, implement Linear Regression in Python (incl. Numpy, Pandas, Matplotlib) on a Jupyter Notebook. Train on the training set of the "ZIP code"-Dataset and test on its test set.

$$\hat{\beta} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y}$$

```
In [1]: import numpy as np
        import pandas as pd

In [2]: class LinearRegression:
            def __init__(self):
                self.coefficients = None

            def fit(self, train_X, train_y):
                a = np.linalg.inv(np.dot(train_X.T, train_X))
                b = np.dot(train_X.T, train_y)
                self.coefficients = np.dot(a,b)

            def predict(self, _test_X):
                return np.dot(_test_X, self.coefficients)

In [3]: _training_data = np.array(pd.read_csv('zip.train', sep=' ', header=None),
                                  dtype=np.float32)
        _test_data = np.array(pd.read_csv('zip.test', sep=' ', header=None),
                              dtype=np.float32)
        _train_x = _training_data[:, 1:-1]
        _train_X = np.hstack((np.ones((_train_x.shape[0],1)), _train_x))
        _train_y = _training_data[:, 0].astype(np.uint8)
        _test_x = _test_data[:, 1:]
        _test_X = np.hstack((np.ones((_test_x.shape[0],1)), _test_x))
        _test_y = _test_data[:, 0].astype(np.uint8)
        _zip_classifier = LinearRegression()
        _zip_classifier.fit(_train_X, _train_y)
```

```
        _y_pred = np.round(_zip_classifier.predict(_test_X), decimals=0).astype(np.int8)
        _conf_m = pd.crosstab(pd.Series(_test_y, name='Actual'),
        pd.Series(_y_pred, name='Predicted'))
        _accuracy = np.sum(np.equal(_y_pred,_test_y)) / len(_test_y)
```

1.1 Print out the Confusion Matrix and the accuracy.

```
In [4]: print('Accuracy ', _accuracy)
        print(_conf_m)
```

```
Accuracy  0.22521175884404585
Predicted  -3   -2   -1    0    1    2    3    4    5    6    7    8    9   \
Actual
0           2   11   44   71  108   65   39   12    4    3    0    0    0
1           0    0    0    6  119   84   35   14    3    3    0    0    0
2           0    1    3   10   19   42   45   45   23    7    3    0    0
3           0    0    1    1   10   24   38   42   36   12    2    0    0
4           0    0    0    0    1    5   11   41   46   41   38   14    3
5           1    0    1    1   10   17   23   33   31   28   11    3    1
6           0    0    0    0    5    8   16   46   54   35    4    2    0
7           0    0    0    0    0    1    3   10   20   45   47   18    3
8           0    0    1    1    1    3    4   24   30   50   34   14    4
9           0    0    0    0    0    0    2    9   17   22   55   56   14

Predicted   10
Actual
0           0
1           0
2           0
3           0
4           0
5           0
6           0
7           0
8           0
9           2
```

1.2 What is a good way of encoding the labels?

Predicted labels are continous values and so they cannot be compared with the actual test values. To overcome the problem, the predicted values could be rounded and encoded as type int8. Alternatively, one could encode the training labels as linearly independent vectors in $\mathbb{R}^{10}$ and choose the best fit to determine the most likely category. An exemplary calculation using that approach is attached below.

1.3 What is the problem with using Linear Regression for Classification?

Linear regression predicts continous values whereas classification problems are of a discrete nature. It is advised to use logistic regression for classification problems.

1.4 Appendix: Encoding labels as (standard basis) vectors in $\mathbb{R}^{10}$

```
In [5]: def base(j):
            return np.eye(1, 10, j)

        def C(correct, predicted):
            assert len(correct) == len(predicted)
            c = np.zeros((10, 10), dtype=int)
            for i, j in zip(correct, predicted):
                c[i, j] += 1
            return c

        def ratio(correct, predicted):
            return np.trace(C(correct, predicted))/len(correct)
```

```
In [6]: train = np.loadtxt('zip.train')
        y, X = train.T[0].astype(int), train.T[1:].T
        y_v = np.array([base(j) for j in y]).reshape(len(X), 10)
```

With pseudo-inverse $(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}$ = np.linalg.pinv(X), for training data:

```
In [7]: beta = np.dot(np.linalg.pinv(X), y_v)
        y_mle = np.dot(X, beta)
        y_mle = [np.argmax(a) for a in y_mle]
```

Confusion matrix for training data:

```
In [8]: C(y, y_mle)
```

```
Out[8]: array([[1177,    0,    1,    4,    3,    1,    7,    0,    1,    0],
               [   0, 1002,    1,    0,    0,    0,    0,    0,    1,    1],
               [   9,    5,  648,   19,   15,    2,   13,    6,   12,    2],
               [   8,    0,    6,  611,    0,    9,    0,    9,   13,    2],
               [   2,   25,    7,    0,  589,    3,    5,    1,    4,   16],
               [  17,    1,    3,   31,   10,  477,    9,    0,    5,    3],
               [  10,    7,    9,    1,   13,    9,  608,    0,    7,    0],
               [   4,    4,    1,    0,    4,    1,    0,  593,    3,   35],
               [  24,    6,    4,   17,   14,   11,    5,    3,  452,    6],
               [   1,    7,    0,    1,   26,    6,    0,   20,    5,  578]])
```

Accuracy for training data:

```
In [9]: ratio(y, y_mle)
```

```
Out[9]: 0.9237415992319298
```

```
In [10]: test = np.loadtxt('zip.test')
         y, X = test.T[0].astype(int), test.T[1:].T
         y_v = np.array([base(j) for j in y]).reshape(len(X), 10)

In [11]: y_mle = np.dot(X, beta)
         y_mle = [np.argmax(a) for a in y_mle]
```

Confusion matrix for test data:

```
In [12]: C(y, y_mle)

Out[12]: array([[347,   2,   0,   1,   3,   1,   4,   0,   0,   1],
                [  0, 254,   0,   2,   3,   0,   3,   0,   1,   1],
                [  7,   3, 159,   8,  10,   1,   1,   1,   8,   0],
                [  5,   0,   4, 137,   2,   9,   0,   2,   3,   4],
                [  3,   7,   4,   0, 169,   1,   3,   2,   1,  10],
                [  9,   1,   0,  20,   2, 122,   0,   1,   1,   4],
                [  3,   2,   4,   0,   5,   4, 151,   0,   1,   0],
                [  3,   1,   1,   1,   7,   0,   0, 130,   0,   4],
                [  8,   3,   3,  15,   3,   8,   1,   2, 119,   4],
                [  0,   3,   0,   0,   7,   0,   0,   7,   2, 158]])
```

Accuracy for test data:

```
In [13]: ratio(y, y_mle)

Out[13]: 0.8699551569506726
```

2 Why use quadratic loss?

We have

$$y_i = \hat{y}(x_i; w) + \epsilon_i$$

With $\epsilon_i \sim \mathcal{N}(0,1) = \frac{1}{2\pi} exp(\frac{-\epsilon_0, i^2}{2})$ and some arbitrary constant factor σ we can write

$$y_i = \hat{y}(x_i; w) + \sigma\epsilon_i$$

It follows that for linear $\hat{y}(x_i; w)$, y_i also normally distributed with

$$y_i \sim \mathcal{N}(\hat{y}(x_i; w), \sigma)$$

We therefore can write for the likelihood

$$L(X, Y, w) = \prod_{i=1}^{n} p(y_i = \hat{y}(x_i; w)) \sim \prod_{i=1}^{n} exp(\frac{-1}{2\sigma^2}(y_i - \hat{y}(x_i; w))^2) = exp(\frac{-1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \hat{y}(x_i; w))^2)$$

We are interested $\hat{w}$ that maximizes the likelihood, or the logarithm of the likelihood

$$\hat{w} = \text{argmax}_w L(X, Y, w) = \text{argmax}_w \log L(X, Y, w),$$

which is equivalent because $\log L(X, Y, w)$ strictly monotone. It follows from above that with positive constant α

$$\log L(X, Y, w) = \frac{-1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \hat{y}(x_i; w))^2 + \alpha.$$

Since the first term in the sum is negative, we can immediately see that the whole term is maximized by

$$\text{argmax}_w \log L(X, Y, w) = \text{argmax}_w \frac{-1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \hat{y}(x_i; w))^2 = \text{argmin}_w \sum_{i=1}^{n} (y_i - \hat{y}(x_i; w))^2$$

It follows that

$$\hat{w} = \text{argmax}_w L(X, Y, w) = \text{argmin}_w \sum_{i=1}^{n} (y_i - \hat{y}(x_i; w))^2$$