KAIST

Project #1
Multinomial logistic regression

Student : David Albert

Contents

I Task 1: Multinomial logistic regression **3**
- 1 Code explanation and results . . . 3
  - 1.1 Code explanation . . . 3
  - 1.2 Outputs . . . 4
- 2 Derivations of the optimal estimator and the optimal average MSE . . . 5
  - 2.1 Integration of the 1st part . . . 5
  - 2.2 Integration of the 2nd part . . . 5
  - 2.3 Integration of the 3rd part . . . 6
  - 2.4 Minimization of average MSE over all θ . . . 6

II Task 2 : Rain or shine **8**
- 1 Explanation and outputs of *project1_task2_generate.py* . . . 8
  - 1.1 Explanation . . . 8
  - 1.2 Outputs . . . 9
- 2 Explanation and outputs of *project1_task2.py* . . . 9
  - 2.1 Explanations . . . 9
  - 2.2 Outputs . . . 11

Part I

Task 1: Multinomial logistic regression

1 Code explanation and results

1.1 Code explanation

Creating the computational graph using softmax function as output and so *softmax cross entropy with logits* as loss function.

```
x = tf.placeholder(tf.float32, shape=[None, 3])
y_true = tf.placeholder(tf.float32, shape=[None, 3])

w = tf.Variable(tf.truncated_normal([3, 3], stddev=0.5))
b = tf.Variable(tf.constant(0.1, shape=[3]))
z = tf.matmul(x, w) + b
y_pred = tf.nn.softmax(z)

loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(
        labels=y_true,logits=z))
train = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
mse = tf.reduce_mean((y_pred - y_true)**2)
```

At each epoch, we create $n = mini\_batch\_size$ random samples of $\theta = \{\theta_0, \theta_1, \theta_2\}$ from the uniform distribution on the simplex $\theta_0 + \theta_1 + \theta_2 = 1$. It is similar to sampling from the Dirichlet(1,1,1) distribution.

```
# Sampling theta=(theta0,theta1,theta2) uniformly on the simplex
# -> theta0+theta1+theta2=1
theta = np.random.dirichlet((1, 1, 1), mini_batch_size)
```

We then generate data according to the random values of θ above mentionned. We sample values from the corresponding 3-ary distribution and transform the result $x \in \{0, 1, 2\}$ as an *one hot* vector.

```
# Generating data from distribution given by the 3-ary distribution
rand = np.random.uniform(0., 1., (mini_batch_size,1))
cumul = np.cumsum(theta, axis=1)
data = 3-np.sum(1.*(np.concatenate((rand,rand,rand),axis=1)<cumul),
                                    axis=1)
```

```python
# Change data as one_hot vector
one_hot_data = np.zeros((mini_batch_size,3))
one_hot_data[np.arange(mini_batch_size), data.astype(int)] = 1
```

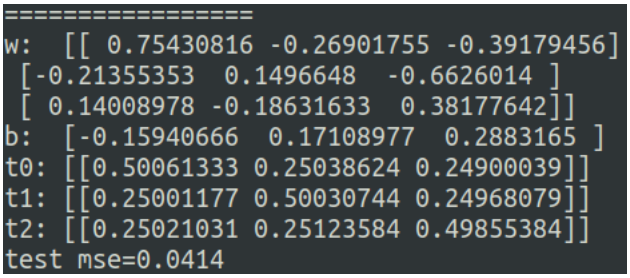
Finally, we train the model to predict θ from a single input.

```python
_, train_mse = sess.run((train, mse),
                        feed_dict={x: one_hot_data, y_true: theta})
```

And test and display the results.

```python
print('w: ', sess.run(w))
print('b: ', sess.run(b))
# estimated theta when 0 is observed
print('t0:', sess.run(y_pred, feed_dict={x:[[1,0,0]]}))
# estimated theta when 1 is observed
print('t1:', sess.run(y_pred, feed_dict={x:[[0,1,0]]}))
# estimated theta when 2 is observed
print('t2:', sess.run(y_pred, feed_dict={x:[[0,0,1]]}))
```

1.2 Outputs



```
=================
w:  [[ 0.75430816 -0.26901755 -0.39179456]
 [-0.21355353  0.1496648  -0.6626014 ]
 [ 0.14008978 -0.18631633  0.38177642]]
b:  [-0.15940666  0.17108977  0.2883165 ]
t0: [[0.50061333 0.25038624 0.24900039]]
t1: [[0.25001177 0.50030744 0.24968079]]
t2: [[0.25021031 0.25123584 0.49855384]]
test mse=0.0414
```

Figure I.1 – Outputs

The results (values of θ and MSE) are similar to the theoritical values found in the next section (see subsection 2.4).

2 Derivations of the optimal estimator and the optimal average MSE

The goal is to determine :

$$\hat{\theta}_{MSE} = argmin_{\hat{\theta}}\left[2\int_0^1\int_0^{1-\theta_0}\mathbb{E}_{X\sim p_\theta(x)}\left[\|\ \hat{\theta}(X)-\theta\|^2\right]\ d\theta_1 d\theta_0\right]$$

We have :

$$\begin{aligned}\mathbb{E}_{X\sim p_\theta(x)}\left(\|\hat{\theta}(X)-\theta\|^2\right) &= \mathbb{E}_{X\sim p_\theta(x)}\left(\sum_{i=0}^{2}\left(\hat{\theta}_i(X)-\theta_i\right)^2\right)\\ &= \sum_i\left(\mathbb{E}\left([\hat{\theta}_i(X)]^2\right)-2\theta_i\mathbb{E}\left(\hat{\theta}_i(X)\right)+\mathbb{E}\left(\theta_i^2\right)\right)\\ &= \underbrace{\sum_{i=0}^{2}\sum_{j=0}^{2}\theta_j[\hat{\theta}_i(j)]^2}_{=(1)}\underbrace{-2\sum_{i=0}^{2}\sum_{j=0}^{2}\theta_i\theta_j\hat{\theta}_i(j)}_{=(2)}+\underbrace{\sum_{i=0}^{2}\theta_i^2}_{=(3)}\end{aligned}$$

Now, we can integrate each part of the MSE over θ_0 and θ_1. Because the probability distribution of $\theta = \{\theta_0,\theta_1,\theta_2\}$ is symetric, we can determine $\hat{\theta}(0) = \left[\hat{\theta}_0(0),\hat{\theta}_1(0),\hat{\theta}_2(0)\right]$ and infer the values of $\hat{\theta}(1)$ and $\hat{\theta}(2)$.

2.1 Integration of the 1st part

Let's calculate :

$$\int_0^1\int_0^{1-\theta_0}(1)\ d\theta_1 d\theta_0$$

with :

$$(1) = \theta_0\left([\hat{\theta}_0(0)]^2+[\hat{\theta}_1(0)]^2+[\hat{\theta}_2(0)]^2\right)+\theta_1\left([\hat{\theta}_0(1)]^2+[\hat{\theta}_1(1)]^2+[\hat{\theta}_2(1)]^2\right)+\theta_2\left([\hat{\theta}_0(2)]^2+[\hat{\theta}_1(2)]^2+[\hat{\theta}_2(2)]^2\right)$$

We find that :

$$\int_0^1\int_0^{1-\theta_0}(1)\ d\theta_1 d\theta_0 = \sum_{j=0}^{2}\frac{\|\hat{\theta}(j)\|^2}{6}$$

2.2 Integration of the 2nd part

Let's now calculate :

$$\int_0^1\int_0^{1-\theta_0}(2)\ d\theta_1 d\theta_0$$

with :

$$(2) = -2[\theta_0^2\hat{\theta}_0(0)+\theta_0\theta_1\hat{\theta}_1(0)+\theta_0\theta_2\hat{\theta}_2(0)+\theta_0\theta_1\hat{\theta}_0(1)+\theta_1^2\hat{\theta}_1(1)+\theta_1\theta_2\hat{\theta}_2(1)+\theta_0\theta_2\hat{\theta}_0(2)+\theta_1\theta_2\hat{\theta}_1(2)+\theta_2^2\hat{\theta}_2(2)]$$

By keeping only the terms refering to $\hat{\theta}(0)$, we have:

$$-2\int_0^1\int_0^{1-\theta_0} \theta_0^2\hat{\theta}_0(0) + \theta_0\theta_1\hat{\theta}_1(0) + \theta_0(1-\theta_0-\theta_1)\hat{\theta}_2(0)\ d\theta_1 d\theta_0 = -\frac{1}{6}\hat{\theta}_0(0) - \frac{1}{12}\hat{\theta}_1(0) - \frac{1}{12}\hat{\theta}_2(0)$$

Because the probability distribution of θ is symetric, we can infer the results of the integral for the terms $\hat{\theta}(1)$ and $\hat{\theta}(2)$ and thus we find that:

$$\begin{aligned}\int_0^1\int_0^{1-\theta_0} (2)\ d\theta_1 d\theta_0 = &-\frac{1}{6}\hat{\theta}_0(0) - \frac{1}{12}\hat{\theta}_1(0) - \frac{1}{12}\hat{\theta}_2(0)\\ &-\frac{1}{12}\hat{\theta}_0(1) - \frac{1}{6}\hat{\theta}_1(1) - \frac{1}{12}\hat{\theta}_2(1)\\ &-\frac{1}{12}\hat{\theta}_0(2) - \frac{1}{12}\hat{\theta}_1(2) - \frac{1}{6}\hat{\theta}_2(2)\end{aligned}$$

2.3 Integration of the 3rd part

Let's finally calculate :

$$\int_0^1\int_0^{1-\theta_0} (3)\ d\theta_1 d\theta_0$$

with :

$$\begin{aligned}(3) &= \theta_0^2 + \theta_1^2 + \theta_2^2\\ &= \theta_0^2 + \theta_1^2 + (1-\theta_0-\theta_1)^2\\ &= 2\theta_0^2 + 2\theta_1^2 - 2\theta_0 - 2\theta_1 + 2\theta_0\theta_1 + 1\end{aligned}$$

We find :

$$\begin{aligned}\int_0^1\int_0^{1-\theta_0} 2\theta_0^2 + 2\theta_1^2 - 2\theta_0 - 2\theta_1 + 2\theta_0\theta_1 + 1\ d\theta_1 d\theta_0 &= \int_0^1 \left(-\frac{5}{3}\theta_0^3 + 3\theta_0^2 - 2\theta_0 + \frac{2}{3}\right)\ d\theta_0\\ &= \frac{1}{4}\end{aligned}$$

2.4 Minimization of average MSE over all θ

Thus, the estimator based one sample that minimizes the average MSE over all θ is given by:

$$\begin{aligned}\hat{\theta}_{MSE} &= argmin_{\hat{\theta}}\left[2\int_0^1\int_0^{1-\theta_0} \mathbb{E}_{X\sim p_\theta(x)}\left[\|\ \hat{\theta}(X)-\theta\|^2\right]\ d\theta_1 d\theta_0\right]\\ &= argmin_{\hat{\theta}}\left[2\int_0^1\int_0^{1-\theta_0} (1)+(2)+(3)\ d\theta_1 d\theta_0\right]\\ &= argmin_{\hat{\theta}}\left[\begin{array}{l}\frac{1}{8} + \frac{1}{6}\left(\hat{\theta}_0(0)-\frac{1}{2}\right)^2 + \frac{1}{6}\left(\hat{\theta}_1(0)-\frac{1}{4}\right)^2 + \frac{1}{6}\left(\hat{\theta}_2(0)-\frac{1}{4}\right)^2\\ \frac{1}{6}\left(\hat{\theta}_0(1)-\frac{1}{4}\right)^2 + \frac{1}{6}\left(\hat{\theta}_1(1)-\frac{1}{2}\right)^2 + \frac{1}{6}\left(\hat{\theta}_2(1)-\frac{1}{4}\right)^2\\ \frac{1}{6}\left(\hat{\theta}_0(2)-\frac{1}{4}\right)^2 + \frac{1}{6}\left(\hat{\theta}_1(2)-\frac{1}{4}\right)^2 + \frac{1}{6}\left(\hat{\theta}_2(2)-\frac{1}{2}\right)^2\end{array}\right]\end{aligned}$$

Finally,

$$\hat{\theta}_{MSE}(x) = \begin{cases} (1/2, 1/4, 1/4) \ if \ x = 0 \\ (1/4, 1/2, 1/4) \ if \ x = 1 \\ (1/4, 1/4, 1/2) \ if \ x = 2 \end{cases}$$

For this estimator, the MSE averaged over all θ is equal to $\frac{1}{8} = 3 \times \frac{1}{24}$. Thus, the error contribution for each $\hat{\theta}_i, \ i \in \{0, 1, 2\}$ is $\frac{1}{24} \approx 0.04$. We find the same result than the pratical result (section 1.2).

Part II

Task 2 : Rain or shine

1 Explanation and outputs of *project1_task2_generate.py*

1.1 Explanation

Function to approximate the steady-state by multiplying the transition matrix of the markov chain. The stop condition is $|P^{i+1} - P^i| < \epsilon$ or $i > 10000$

```
# Approximate the steady-state by multiplying the transition
# matrix of the markov chain until convergence.
def approx_steady(t_matrix, eps=0.0001):
    def dist_max(x,y):
        return max(np.abs(x-y))
    i = 0
    t0 = t_matrix
    t1 = np.matmul(t0, t_matrix)
    while (i<10000 and dist_max(t0[0],t1[0])>eps):
        t0=t1
        t1 = np.matmul(t0, t_matrix)
    return t0[0]
```

The function *generate_next_state* generates a next state X_{t+1} given previous state X_t and transition matrix P.

```
# Return a state given a list of proba :
#t_prob = [p(0), p(1), p(2), p(3)]
def generate_state(t_prob):
    p = np.random.rand(1)
    cumsum = np.cumsum(t_prob)
    for i, val in enumerate(cumsum):
        if (p<val):
            return i

# Generate next state given previous state
# and transition matrix
def generate_next_state(prev_state, tr_mat):
    return generate_state(tr_mat[prev_state])
```

We then can generate easily $n = 100000$ data given the transition matrix describes by psh, pra, pcl, psn. X_0 is generated thanks to the approximation of the steady-state probabilities.

```python
# prob. conditional of sh (shine), ra (rain), cl (cloudy),
# sn (snow)  given for each shine, rain, cloudy, snow
psh = [0.6, 0.1, 0.1, 0.1]
pra = [0.1, 0.5, 0.3, 0.1]
pcl = [0.2, 0.3, 0.4, 0.3]
psn = [0.1, 0.1, 0.2, 0.5]

# create transition matrix of the markov chain
tr_mat = np.transpose([psh, pra, pcl, psn])

# steady-state prob. of respectively shine, rain, cloudy, snow
pr = approx_steady(tr_mat)
print("Approx steady-state prob. : {}".format(pr))

n = 100000
x = np.zeros(n, dtype=np.int)

for k in range(n):
    if k == 0:
        x[k] = generate_state(pr)
    else:
        x[k] = generate_next_state(x[k-1], tr_mat)
```

1.2 Outputs

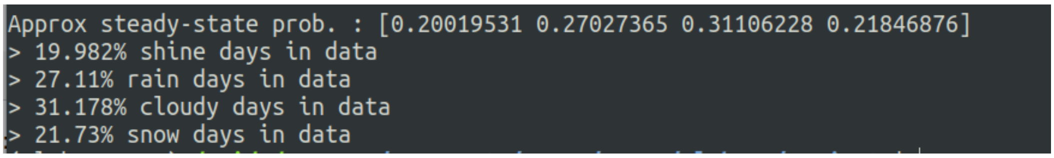

```
Approx steady-state prob. : [0.20019531 0.27027365 0.31106228 0.21846876]
> 19.982% shine days in data
> 27.11% rain days in data
> 31.178% cloudy days in data
> 21.73% snow days in data
```

Figure II.1 – Stats of data

2 Explanation and outputs of *project1_task2.py*

2.1 Explanations

We assume in this example that the weather at day t is fixed. It can take only one value given by shine (0), rain (1), cloudy (2), snow (3). Thus, we create a placeholder for the input (day t) and one for the label (day t+1) which are integers $x \in \{0, 1, 2, 3\}$ and translate these values in "one hot" vector to help our neural network training.

```python
# Create placeholder of input : the weather at day t.
# shine (0), rain (1), cloudy (2), snow (3)
x = tf.placeholder(tf.int32, shape=[None, 1])
x_hot = tf.one_hot(x, 4)
```

```
# Create placeholder of input : the weather at day t+1.
# shine (0), rain (1), cloudy (2), snow (3)
y_true = tf.placeholder(tf.int32, shape=[None, 1])
y_true_hot = tf.one_hot(y_true, 4)
```

Then, we create our neural network architecture. It consists of a simple multinomial logistic regression with four inputs and four outputs. Thus, the computed function is :

$$p(y = i \,|x; w, b) = softmax(w^T x + b)_i \;, \quad \forall i \in \{0, 1, 2, 3\}$$

with $x \in \mathbb{R}^4$, $w \in \mathbb{R}^4 \times \mathbb{R}^4$ and $b \in \mathbb{R}^4$

```
w = tf.Variable(tf.truncated_normal([4, 4], stddev=0.5))
b = tf.Variable(tf.constant(0.1, shape=[4]))
z = tf.matmul(x_hot, w) + b
y_pred = tf.nn.softmax(z)

loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_true_hot,
                                            logits=z))
train = tf.train.AdamOptimizer(0.001).minimize(loss)
```

We train the model as for the rain and shine problem.

```
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
for epoch in range(30):
    rindex = np.random.permutation(n)
    s = s[rindex,:]
    p = p[rindex,:]
    for i in range(n // m):
        sess.run(   train,
                    feed_dict={
                        x: s[i*m:i*m+m,:],
                        y_true: p[i*m:i*m+m,:]
                        })
```

Finnaly, we test the model and display results. We display the probability of weather at day $t + 1$ given weather at day t for all possible weather at day t.

```
s_test = np.zeros((4, 1))
s_test[0] = 0
s_test[1] = 1
s_test[2] = 2
s_test[3] = 3

# Evalute prediction for each weather at day t
pred = y_pred.eval(feed_dict={x: s_test})

# Display results
print()
print("------ Resultats pred prob. cond ------")
```

```python
print("Proba[ X(t+1) | X(t) = 'shine' ] = {}".format(pred[0]))
print("Proba[ X(t+1) | X(t) = 'rain'  ] = {}".format(pred[1]))
print("Proba[ X(t+1) | X(t) = 'cloudy' ] = {}".format(pred[2]))
print("Proba[ X(t+1) | X(t) = 'snow' ] = {}".format(pred[3]))
```

2.2 Outputs

```
----- Resultats pred prob. cond -----
Proba[ X(t+1) | X(t) = 'shine' ] = [[0.59664243 0.10369433 0.20219867 0.09746451]]
Proba[ X(t+1) | X(t) = 'rain'  ] = [[0.09989394 0.49063125 0.30890697 0.1005678 ]]
Proba[ X(t+1) | X(t) = 'cloudy' ] = [[0.09801193 0.29907972 0.40050235 0.20240599]]
Proba[ X(t+1) | X(t) = 'snow' ] = [[0.10040201 0.10248841 0.30068332 0.49642625]]
```

Figure II.2 – Prediction prob. of weather at day t+1 given weather at day t

We can see that the neural network have approximately learnt the conditional probabilities given in the table of proba. There is a small error probably due to the small number of initial data.