Introduction

The topic of this project is to implement logistic regression algorithm to the fashion_train.cvs dataset to build the predict model, and give the accuracy score of the model by implementing prediction fashion_test.cvs dataset. The dataset belongs to 70000 instances of 10 clothing pictures, each of which is identified with a numeric label (0 through 9). 60000 are training set and 10000 are testing set.

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

This report is mainly focus on the process and a critical analysis of the results in steps and final conclusions, along with the reasons behind some important commands, i.e. why the reported numbers behave the way they do and some code/formula that displayed logistic regression algorithm. In this report, we will also compare the result with KNN algorithm model, which we got in last week's project.

Step

1. Import data and do the cleaning
2. Relabel 10 different digits with one-hot encoding
3. Set training X, training Y, testing X and testing Y
4. Fit the model and make predict based on logistic regression with dataset prepared in step 3
5. Build confusion matrix to look up the accuracy roughly
6. Check the possibility matrix and accuracy score

Work

Step 1 Import dataset and do the cleaning

```
[50] # Import the orginial dataset
     train = pd.read_csv('/content/sample_data/fashion_train.csv', header = None)
     test = pd.read_csv('/content/sample_data/fashion_test.csv', header = None)
```

Here is the following dataset that will be applied into the training algorithm:

Train##

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	2
0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	1	0	0	0	0	41	188	103	54	48	43	87	168	133	16	0	
2	0	0	0	0	0	0	0	0	0	0	22	118	24	0	0	0	0	0	48	88	5	0	
3	3	0	0	0	0	0	0	0	0	33	96	175	156	64	14	54	137	204	194	102	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0	0	0	

5 rows × 785 columns

##Test##

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	0	0	0	0	0	0	0	0	0	0	13	67	0	0	0	0	50	38	0	0	0
2	1	0	0	0	0	0	0	0	0	1	0	67	177	129	153	117	129	146	141	175	0	0
3	1	0	0	0	0	0	0	0	0	0	21	123	108	99	99	84	83	86	92	70	6	0
4	6	0	0	0	2	0	1	1	0	0	0	0	57	67	73	76	76	83	62	0	0	0

5 rows × 785 columns

The first column in each dataset stands for labels that listed in the introduction part. The integers in 1 to 785 columns stands for pixels in each instance by row. Here only 21 columns are displayed to give a look.

```
# Rename the first column as 'initial' to stands for original labels
train.rename(columns={0:'initial'}, inplace=True)
test.rename(columns={0:'initial'}, inplace=True)
print(train)
print(test)
```

```
       initial  1  2  3  4  5  6  7  ...  777  778  779  780  781  782  783  784
0            9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
1            0  0  0  0  0  0  1  0  ...  130   76    0    0    0    0    0    0
2            0  0  0  0  0  0  0  0  ...    1    0    0    0    0    0    0    0
3            3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
4            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
...        ... .. .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  ...  ...  ...
59995        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
59996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
59997        3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
59998        0  0  0  0  0  0  0  0  ...   50    5    0    1    0    0    0    0
59999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0

[60000 rows x 785 columns]
      initial  1  2  3  4  5  6  7  ...  777  778  779  780  781  782  783  784
0           9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
1           2  0  0  0  0  0  0  0  ...    0    3  174  189   67    0    0    0
2           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
3           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
4           6  0  0  0  2  0  1  1  ...    0    0    0    0    0    0    0    0
...       ... .. .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  ...  ...  ...
9995        9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
9996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
9997        8  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
9998        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0
9999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0    0

[10000 rows x 785 columns]
```

Step 2 Relabel 10 different digits with one-hot encoding

```
######### relabel 0 ########
### train_zero
train_zero = train
train_zero['relabel'] = 0
train_zero.loc[train_zero['initial']==0,'relabel']=1 # replace label 0 as 1, label m as 0
print(train_zero)
### test_zero
test_zero = test
test_zero['relabel'] = 0
test_zero.loc[test_zero['initial']==0,'relabel']=1 # set label 0 as 1, label m as 0
print(test_zero)
```

One-hot encoding is where the integer encoded variable is removed and one new binary variable is added for each unique integer value in the variable. In this example where train_zero and test_zero has been made, train and test sets are managed respectively. First, set an all-zero column as ‘relabel’, when the initial label is equal to the model’s name (zero at here), ‘relabel’ will change 0 into 1 accordingly. Otherwise it will keep being zero.

```
       initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0            9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1            0  0  0  0  0  0  1  0  ...   76    0    0    0    0    0    0        1
2            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
3            3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
...        ... .. .. .. .. .. .. .. ...  ...  ...  ...  ...  ...  ...  ...      ...
59995        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59997        3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59998        0  0  0  0  0  0  0  0  ...    5    0    1    0    0    0    0        1
59999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[60000 rows x 786 columns]
      initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0           9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1           2  0  0  0  0  0  0  0  ...    3  174  189   67    0    0    0        0
2           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4           6  0  0  0  2  0  1  1  ...    0    0    0    0    0    0    0        0
...       ... .. .. .. .. .. .. .. ...  ...  ...  ...  ...  ...  ...  ...      ...
9995        9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9997        8  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9998        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
```

```
#########  relabel  1  ########
###  train_one
train_one  =  train
train_one['relabel']  =  0
train_one.loc[train_one['initial']==1,'relabel']=1  #  replace  label  1  as  1,  label  m  as  0
print(train_one)
###  test_one
test_one  =  test
test_one['relabel']  =  0
test_one.loc[test_one['initial']==1,'relabel']=1  #  set  label  1  as  1,  label  m  as  0
print(test_one)
```

```
       initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0            9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1            0  0  0  0  0  0  1  0  ...   76    0    0    0    0    0    0        0
2            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3            3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
...        ... .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  ...  ...      ...
59995        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
59997        3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59998        0  0  0  0  0  0  0  0  ...    5    0    1    0    0    0    0        0
59999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[60000 rows x 786 columns]
      initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0           9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1           2  0  0  0  0  0  0  0  ...    3  174  189   67    0    0    0        0
2           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
3           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
4           6  0  0  0  2  0  1  1  ...    0    0    0    0    0    0    0        0
...       ... .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  ...  ...      ...
9995        9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
9997        8  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9998        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
9999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[10000 rows x 786 columns]
```

```
#########  relabel  2  ########
###  train_two
train_two  =  train
train_two['relabel']  =  0
train_two.loc[train_two['initial']==2,'relabel']=1  #  replace  label  2  as  1,  label  m  as  0
print(train_two)
###  test_two
test_two  =  test
test_two['relabel']  =  0
test_two.loc[test_two['initial']==2,'relabel']=1  #  set  label  2  as  1,  label  m  as  0
print(test_two)
```

```
       initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0            9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1            0  0  0  0  0  0  1  0  ...   76    0    0    0    0    0    0        0
2            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3            3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
...        ... .. .. .. .. .. .. .. ...  ...  ...  ...  ...  ...  ...  ...      ...
59995        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59997        3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59998        0  0  0  0  0  0  0  0  ...    5    0    1    0    0    0    0        0
59999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[60000 rows x 786 columns]
      initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0           9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1           2  0  0  0  0  0  0  0  ...    3  174  189   67    0    0    0        1
2           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4           6  0  0  0  2  0  1  1  ...    0    0    0    0    0    0    0        0
...       ... .. .. .. .. .. .. .. ...  ...  ...  ...  ...  ...  ...  ...      ...
9995        9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9997        8  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9998        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[10000 rows x 786 columns]
```

```
#########  relabel  3  ########
###  train_three
train_three  =  train
train_three['relabel']  =  0
train_three.loc[train_three['initial']==3,'relabel']=1  #  replace  label  3  as  1,  label  m  as  0
print(train_three)
###  test_three
test_three  =  test
test_three['relabel']  =  0
test_three.loc[test_three['initial']==3,'relabel']=1  #  set  label  3  as  1,  label  m  as  0
print(test_three)
```

```
       initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0            9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1            0  0  0  0  0  0  1  0  ...   76    0    0    0    0    0    0        0
2            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3            3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
4            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
...        ... .. .. .. .. .. .. .. ...  ...  ...  ...  ...  ...  ...  ...      ...
59995        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59997        3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        1
59998        0  0  0  0  0  0  0  0  ...    5    0    1    0    0    0    0        0
59999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[60000 rows x 786 columns]
      initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0           9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1           2  0  0  0  0  0  0  0  ...    3  174  189   67    0    0    0        0
2           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4           6  0  0  0  2  0  1  1  ...    0    0    0    0    0    0    0        0
...       ... .. .. .. .. .. .. .. ...  ...  ...  ...  ...  ...  ...  ...      ...
9995        9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9997        8  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9998        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[10000 rows x 786 columns]
```

```
#########  relabel  4  ########
###  train_four
train_four  =  train
train_four['relabel']  =  0
train_four.loc[train_four['initial']==4,'relabel']=1  #  replace  label  4  as  1,  label  m  as  0
print(train_four)
###  test_four
test_four  =  test
test_four['relabel']  =  0
test_four.loc[test_four['initial']==4,'relabel']=1  #  set  label  4  as  1,  label  m  as  0
print(test_four)
```

```
       initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0            9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1            0  0  0  0  0  0  1  0  ...   76    0    0    0    0    0    0        0
2            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3            3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4            0  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
...        ... .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  ...  ...      ...
59995        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59997        3  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
59998        0  0  0  0  0  0  0  0  ...    5    0    1    0    0    0    0        0
59999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[60000 rows x 786 columns]
      initial  1  2  3  4  5  6  7  ...  778  779  780  781  782  783  784  relabel
0           9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
1           2  0  0  0  0  0  0  0  ...    3  174  189   67    0    0    0        0
2           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
3           1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
4           6  0  0  0  2  0  1  1  ...    0    0    0    0    0    0    0        0
...       ... .. .. .. .. .. .. ..  ...  ...  ...  ...  ...  ...  ...  ...      ...
9995        9  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9996        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9997        8  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9998        1  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0
9999        5  0  0  0  0  0  0  0  ...    0    0    0    0    0    0    0        0

[10000 rows x 786 columns]
```

Only label 0 to 4 are displayed here and below. Label 5 to 9 are the same process with similar codes.

Step 3 define training X, training Y, testing X and testing Y

```
###  train  x
train_zero_x  =  train_zero.iloc[:,1:785]
train_one_x  =  train_one.iloc[:,1:785]
train_two_x  =  train_two.iloc[:,1:785]
train_three_x  =  train_three.iloc[:,1:785]
train_four_x  =  train_four.iloc[:,1:785]
train_five_x  =  train_five.iloc[:,1:785]
train_six_x  =  train_six.iloc[:,1:785]
train_seven_x  =  train_seven.iloc[:,1:785]
train_eight_x  =  train_eight.iloc[:,1:785]
train_nine_x  =  train_nine.iloc[:,1:785]
```

```
###  train  y
train_zero_y  =  train_zero['relabel']
train_one_y  =  train_one['relabel']
train_two_y  =  train_two['relabel']
train_three_y  =  train_three['relabel']
train_four_y  =  train_four['relabel']
train_five_y  =  train_five['relabel']
train_six_y  =  train_six['relabel']
train_seven_y  =  train_seven['relabel']
train_eight_y  =  train_eight['relabel']
train_nine_y  =  train_nine['relabel']
```

```
#  test  x
test_zero_x  =  test_zero.iloc[:,1:785]
test_one_x  =  test_one.iloc[:,1:785]
test_two_x  =  test_two.iloc[:,1:785]
test_three_x  =  test_three.iloc[:,1:785]
test_four_x  =  test_four.iloc[:,1:785]
test_five_x  =  test_five.iloc[:,1:785]
test_six_x  =  test_six.iloc[:,1:785]
test_seven_x  =  test_seven.iloc[:,1:785]
test_eight_x  =  test_eight.iloc[:,1:785]
test_nine_x  =  test_nine.iloc[:,1:785]
```

```
#  test  y
test_zero_y  =  test_zero['relabel']
test_one_y  =  test_one['relabel']
test_two_y  =  test_two['relabel']
test_three_y  =  test_three['relabel']
test_four_y  =  test_four['relabel']
test_five_y  =  test_five['relabel']
test_six_y  =  test_six['relabel']
test_seven_y  =  test_seven['relabel']
test_eight_y  =  test_eight['relabel']
test_nine_y  =  test_nine['relabel']
```

So far it has all been in preparation for the unfolding of logistic regression algorithms. Once the explanatory

and interpreted variables of the training and test sets have been confirmed, the third part is to perform standard logistic regression using python's built-in algorithms.

Step 4 Logistics Regression

```
clf_zero = LR(random_state=0).fit(train_zero_x, train_zero_y)
pred_zero = clf_zero.predict(test_zero_x)
clf_one = LR(random_state=0).fit(train_one_x, train_one_y)
pred_one = clf_one.predict(test_one_x)
clf_two = LR(random_state=0).fit(train_two_x, train_two_y)
pred_two = clf_two.predict(test_two_x)
clf_three = LR(random_state=0).fit(train_three_x, train_three_y)
pred_three = clf_three.predict(test_three_x)
clf_four = LR(random_state=0).fit(train_four_x, train_four_y)
pred_four = clf_four.predict(test_four_x)
clf_five = LR(random_state=0).fit(train_five_x, train_five_y)
pred_five = clf_five.predict(test_five_x)
clf_six = LR(random_state=0).fit(train_six_x, train_six_y)
pred_six = clf_six.predict(test_six_x)
clf_seven = LR(random_state=0).fit(train_seven_x, train_seven_y)
pred_seven = clf_seven.predict(test_seven_x)
clf_eight = LR(random_state=0).fit(train_eight_x, train_eight_y)
pred_eight = clf_eight.predict(test_eight_x)
clf_nine = LR(random_state=0).fit(train_nine_x, train_nine_y)
pred_nine = clf_nine.predict(test_nine_x)
```

In order to show the accuracy of the regression model, it needs to be examined using different metrics and presentations. Steps five and six are used to examine its accuracy in three ways. The first is the confusion matrix, which presents in a two-dimensional matrix the number of errors between the test set y-values obtained by model prediction and the true y-values of the test set under different labels. In this confusion matrix, we shared 10 models, each with labels of 0 and 1, so the output of the confusion matrix is a 2*2 matrix.

Step 5 Confusion Matrix (CM)

```
[49] CM_zero  =  CM(pred_zero, test_zero_y)
     CM_one  =  CM(pred_one, test_one_y)
     CM_two  =  CM(pred_two, test_two_y)
     CM_three  =  CM(pred_three, test_three_y)
     CM_four  =  CM(pred_four, test_four_y)
     CM_five  =  CM(pred_five, test_five_y)
     CM_six  =  CM(pred_six, test_six_y)
     CM_seven  =  CM(pred_seven, test_seven_y)
     CM_eight  =  CM(pred_eight, test_eight_y)
     CM_nine  =  CM(pred_nine, test_nine_y)
     print(CM_zero)
     print(CM_one)
     print(CM_two)
     print(CM_three)
     print(CM_four)
     print(CM_five)
     print(CM_six)
     print(CM_eight)
     print(CM_nine)
```

```
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
[[8913   74]
 [  87  926]]
```

Python automatically filters for the optimal result, so the 10 models get the same confusion matrix, which simplifies our process.

The sum of the numbers in the obfuscation matrix should be equal to the sum of the instances under each label.

74+87 predictions were inaccurate and 8913+926 predictions were accurate.

Step 6 Check the possibility matrix and accuracy score

```
## Check the possibility matrix for each pixcel ##
print(clf_zero.predict_proba(test_zero_x))
print(clf_one.predict_proba(test_one_x))
print(clf_two.predict_proba(test_two_x))
print(clf_three.predict_proba(test_three_x))
print(clf_four.predict_proba(test_four_x))
print(clf_five.predict_proba(test_five_x))
print(clf_six.predict_proba(test_six_x))
print(clf_seven.predict_proba(test_seven_x))
print(clf_eight.predict_proba(test_eight_x))
print(clf_nine.predict_proba(test_nine_x))
```

```
[[3.41672575e-01 6.58327425e-01]
 [1.00000000e+00 1.00691419e-29]
 [1.00000000e+00 1.49089472e-10]
 ...
 [9.99999998e-01 1.83874960e-09]
 [9.99974458e-01 2.55419417e-05]
 [9.99995748e-01 4.25213059e-06]]
```

The prediction accuracy of each pixel in each model is shown in the likelihood matrix, and is shown in the final step as a verification of whether the results are consistent. The same results were obtained, again verifying the correctness of the algorithm

```
####### Check the accuracy score #######
print(accuracy_score(pred_zero, test_zero_y))
print(accuracy_score(pred_one, test_one_y))
print(accuracy_score(pred_two, test_two_y))
print(accuracy_score(pred_three, test_three_y))
print(accuracy_score(pred_four, test_four_y))
print(accuracy_score(pred_five, test_five_y))
print(accuracy_score(pred_six, test_six_y))
print(accuracy_score(pred_seven, test_seven_y))
print(accuracy_score(pred_eight, test_eight_y))
print(accuracy_score(pred_nine, test_nine_y))
```

0. 9839

As can already be seen from the confusion matrix, the evaluation indicators will be the same for all 10 models. Therefore the report is shown only once. The accuracy score is 0.9839, which is close to zero, indicating that the models are predicting well.

Findings

	Logistic Regression	K-nearest Neighbors (K = 1)	Review
Encoding Method	One-hot encoding	Integer encoding	Logistic regression has significant advantages when applied to multiple variables. Because the one-hot coding is binary for the classification of multiple labels, it is consistent with the treatment of logistic regression.
Confusion Matrix	[[8913 74] [87 926]]	[[800 7 15 35 5 0 160 0 5 0] [2 975 2 9 2 0 1 0 1 0] [20 2 782 14 127 0 117 0 9 0] [26 8 10 850 34 0 27 0 3 0] [5 4 97 42 734 0 69 0 2 0] [0 0 0 0 0 863 0 5 0 2] [142 3 94 48 97 2 619 0 17 0] [1 0 0 0 0 68 0 949 4 30] [4 1 0 2 1 1 7 0 958 1] [0 0 0 0 0 66 0 46 1 967]	1. By simply reencoding and use 0 and 1 to numerous labels, logistic regression has less complexity confusion matrix 2. Here k lists only the optimal one, selected by comparing exact value scores at different k values. The results of KNN are obtained by human selection. For logistic regression, python is able to give the optimized matrix and the exact worth score at once. Logistic regression requires significantly fewer adjustments than KNN.
Accuracy Score	0. 9839	0. 8497	Logistic regression is more accurate

References

Brownlee J. (2020, June 12). Ordinal and One-Hot Encodings for Categorical Data. Retrieved from, https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/

N.N. (n.d.) w3schools. https://www.w3schools.com/python/python_arrays.asp