Task 6

Preparation

First, let look at the data

Review	label	Cate	zipCode	nreview	avg_star
The baguettes and rolls are excellent, and ...	1	[Vietnamese, Sandwiches, Restaurants] ...	98118	4	4.0
I live up the street from Betty. When my ...	1	[American (New), Restaurants] ...	98109	21	4.04761904762
I'm worried about how I will review this place ...	1	[Mexican, Restaurants]	98103	14	3.11111111111
Why can't you access them on Google street view? ...	0	[Mexican, Tex-Mex, Restaurants] ...	98112	42	4.08888888889
Things to like about this place: homemade ...	0	[Mexican, Restaurants]	98102	12	3.07142857143
I had been holding off on visiting Bastille for ...	1	[Breakfast & Brunch, French, Restaurants] ...	98107	59	3.63492063492
I had gone by this place as they were moving in ...	1	[Creperies, Restaurants]	98105	6	4.16666666667
Any chance I get to eat with my hands and hav ...	0	[Vegetarian, Ethiopian, Vegan, Restaurants] ...	98102	17	3.72222222222
My favorite Thai restaurant in the ...	0	[Thai, Restaurants]	98105	16	3.58823529412
I'm pretty sure someone who was born and raised ...	0	[Barbeque, Restaurants]	98108	19	3.09523809524

[13299 rows x 6 columns]

In this task, I will first unpack *Cate* column into columns that represent each item categories. For example if a item have tag: Vietnamese, Chinese their new *Cate.Vietnamese* and *Cate.Chinese* columns will be marked 1 while other are left 0 (None).

```
data['Cate'] = data['Cate'].apply(lambda x: dict(Counter(x)))
data = data.unpack('Cate')
```

After that, I remove all non-letter charaters from reviews that I think is not necessary. Then I ran tf_idf on these reviews' bi-gram data instead of unigram. As I consider I bigram will be better for the next step.

```
data['Review'] = data['Review'].apply(lambda x: re.sub('[^a-z]+', ' ', x.lower()))

#data['tf_idf'] = gl.text_analytics.tf_idf(gl.text_analytics.count_words(data['Review']))
data['tf_idf'] = gl.text_analytics.tf_idf(gl.text_analytics.count_ngrams(data['Review']))
```

```
m = gl.topic_model.create(data['tf_idf'],method='alias',num_iterations=20)
```

Base on tf_idf data, I ran lda topic model and output the probility of each 1 in 10 topics as 10 new features to my data. So I have total 112 features: 'zipCode', 'nreview', 'avg_star', 10 probility columns and category columns.

```
data['prob'] = m.predict(data['tf_idf'],output_type='probability')

def list2dict(l):
    rep = dict()
    for i in range(10):
        rep[i] = l[i]
    return rep
data['prob'] = data['prob'].apply(list2dict)

data.remove_columns(['tf_idf','Review'])

data = data.unpack('prob')

features = data.column_names()
for col in features:
    data = data.fillna(column=col,value=0)
```

Running Machine learning:

First I left all the data that label was set to be *'[None]'*

```
leftbh = data[data['label']=='[None]']

aval = data[data['label']!='[None]']

aval['label'] = aval['label'].apply(int)
```

With the remaining data, I split them into training and testing set, training set contains 90% of all available data and 10% belong to test set.

```
train,test = aval.random_split(fraction=.9,seed=317)
```

The training set itself also be split 80-20 into a set that we will run training algorithm on and a set to valid these algorithm.

```
train,valid = train.random_split(fraction=.8,seed=317)
```

Random forest:

The first algorithm I tried is random forest. As can be seen the model has *0.86* accuracy with training set and *0.55* accuracy with validate set.

I evaluated it with test data and got **0.74** F1 score:

```
m = gl.random_forest_classifier.create(train, target='label',random_seed=317,validation_se
t=valid)

Random forest classifier:

+-----------+--------------+-----------------+--------------------+-------------------+
---------------------+

| Iteration | Elapsed Time | Training-accuracy | Validation-accuracy | Training-log_loss |
 Validation-log_loss |
```

```
+-----------+--------------+-------------------+---------------------+-------------------+
---------------------+

| 1         | 0.006018     | 0.725000          | 0.513761            | 0.544193          |
 0.861746            |

| 2         | 0.016029     | 0.792500          | 0.513761            | 0.484419          |
 0.783850            |

| 3         | 0.018034     | 0.845000          | 0.577982            | 0.463319          |
 0.735031            |

| 4         | 0.019538     | 0.860000          | 0.559633            | 0.460365          |
 0.727569            |

| 5         | 0.021042     | 0.875000          | 0.568807            | 0.446494          |
 0.719409            |

| 6         | 0.022546     | 0.867500          | 0.550459            | 0.452656          |
 0.717280            |

+-----------+--------------+-------------------+---------------------+-------------------+
---------------------+

gl.evaluation.f1_score(predictions=m.predict(test),targets=test['label'])

0.7441860465116279
```

Boosted Trees:

The second algorithm I tried is boosted tree. As can be seen the model has *0.91* accuracy with training set and *0.53* accuracy with validate set.

I evaluated it with test data and got **0.68** F1 score:

```
m = gl.boosted_trees_classifier.create(train, target='label',random_seed=317,validation_se
t=valid)

Boosted trees classifier:

+-----------+--------------+-------------------+---------------------+-------------------+
---------------------+

| Iteration | Elapsed Time | Training-accuracy | Validation-accuracy | Training-log_loss |
 Validation-log_loss |

+-----------+--------------+-------------------+---------------------+-------------------+
---------------------+

| 1         | 0.004011     | 0.810000          | 0.532110            | 0.597237          |
 0.698397            |

| 2         | 0.006516     | 0.822500          | 0.513761            | 0.534078          |
 0.699161            |

| 3         | 0.008522     | 0.905000          | 0.541284            | 0.465018          |
 0.705930            |
```

```
| 4         | 0.010527    | 0.905000        | 0.568807          | 0.429568         |
 0.733936           |

| 5         | 0.011530    | 0.907500        | 0.550459          | 0.403339         |
 0.750401           |

| 6         | 0.013035    | 0.917500        | 0.532110          | 0.375698         |
 0.770972           |

+-----------+-------------+-----------------+-------------------+------------------+
--------------------+
```

```
gl.evaluation.f1_score(predictions=m.predict(test),targets=test['label'])
```

```
0.6829268292682926
```

Nearest neighbor

The third algorithm I tried is nearest neighbor. I expected so much on this algorithm because If restaurants have near values, they may have same condition. The model gave me *0.715* accuracy with training set and *0.55* accuracy with validate set.

I evaluated it with test data and got **0.70** F1 score:

```
m = gl.nearest_neighbor_classifier.create(train, target='label',distance='squared_euclidea
n',verbose=False)
```

```
gl.evaluation.f1_score(predictions=m.predict(test),targets=test['label'])
```

```
0.7027027027027026
```

Vector model

I ran vector model classifier on this data and got *0.74* training accuracy, *0.58* validate accuracy and **0.68** F1 score.

```
m = gl.svm_classifier.create(train, target='label',validation_set=valid,max_iterations=100
00)
```

```
+-----------+----------+-----------+--------------+-------------------+-------------------
--+

| Iteration | Passes   | Step size | Elapsed Time | Training-accuracy | Validation-accurac
y |

+-----------+----------+-----------+--------------+-------------------+-------------------
--+

| 1         | 3        | 0.002500  | 0.029112     | 0.700000          | 0.513761
  |

| 2         | 5        | 1.000000  | 0.058144     | 0.715000          | 0.550459
  |

| 101       | 148      | 0.500000  | 1.528556     | 0.740000          | 0.577982
  |
```

```
| 200       | 296      | 1.000000  | 2.906220     | 0.740000          | 0.577982
  |

+-----------+----------+-----------+--------------+-------------------+------------------
--+

SUCCESS: Optimal solution found.
```

```
gl.evaluation.f1_score(predictions=m.predict(test),targets=test['label'])
```

```
0.6842105263157895
```

Logistic regression

I actually did not expected much in this algorithm as it's may seem to be too simple. I got only *0.73* training accuracy and *0.56* validation accuracy. However, when I tested with test data, I gave me **0.77** F1 score, highest in these algorithm.

```
m=gl.logistic_classifier.create(dataset=train,target='label',validation_set=valid,l1_penal
ty=.1,l2_penalty=.1)
```

```
Logistic regression:

+-----------+----------+-----------+--------------+-------------------+------------------
--+

| Iteration | Passes   | Step size | Elapsed Time | Training-accuracy | Validation-accurac
y |

+-----------+----------+-----------+--------------+-------------------+------------------
--+

Tuning step size. First iteration could take longer than subsequent iterations.

| 1         | 2        | 0.005138  | 0.099853     | 0.700000          | 0.513761
  |

| 2         | 3        | 0.002284  | 0.132901     | 0.712500          | 0.522936
  |

| 3         | 4        | 0.002284  | 0.151966     | 0.715000          | 0.522936
  |

| 11        | 12       | 0.002284  | 0.302359     | 0.727500          | 0.559633
  |

+-----------+----------+-----------+--------------+-------------------+------------------
--+

SUCCESS: Optimal solution found.
```

```
gl.evaluation.f1_score(predictions=m.predict(test),targets=test['label'])
```

```
0.7692307692307692
```

Conlusion

I think that, each freature in the data has some effect on the label. Low rate or low number of reviews may because of low number of customers because It did not pass the test. Restaurants in same categories or neighbors maybe in same condition as they serve same type of customers and may go same result of the test. I also think that some topic in topic model may be effective as custome may complain alot if there is some problem with their experience.