Amazon fine food- Logistic regression

December 26, 2018

1 Amazon fine food- Logistic regression

```
In [194]: import sqlite3
          import pandas as pd
          import numpy as np
          import nltk
          import string
          import matplotlib.pyplot as plt
          import seaborn as sns

          con = sqlite3.connect('./final.sqlite')

          review= pd.read_sql_query('''select * from Reviews''',con)
```

```
In [195]: review=review.sort_values('Time')
```

2 Bow

```
In [70]: from sklearn.feature_extraction.text import TfidfTransformer
         from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn import datasets, neighbors
         from matplotlib.colors import ListedColormap

         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.metrics import confusion_matrix
         from sklearn import metrics
         from sklearn.metrics import roc_curve, auc
         from nltk.stem.porter import PorterStemmer
```

```
In [71]: #BoW
         count_vect = CountVectorizer() #in scikit-learn
         final_counts=count_vect.fit_transform(review['Text'].values)
```

```
In [72]: final_counts.get_shape()

Out[72]: (9982, 22825)
```

In [196]:
```
per=int(0.7*final_counts.shape[0])
a=final_counts[0:per]
b=final_counts[per:]
label=review.Score[0:per]
label1=review.Score[per:]
```

In [197]:
```
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=False).fit_transform(a)
print(standardized_data.shape)
```

```
c:\users\vishal\appdata\local\programs\python\python36\lib\site-packages\sklearn\utils\validat
  warnings.warn(msg, DataConversionWarning)


(6987, 22825)
```

In [198]:
```
from sklearn.preprocessing import StandardScaler
standardized_data1 = StandardScaler(with_mean=False).fit_transform(b)
print(standardized_data.shape)
```

```
c:\users\vishal\appdata\local\programs\python\python36\lib\site-packages\sklearn\utils\validat
  warnings.warn(msg, DataConversionWarning)


(6987, 22825)
```

In [199]:
```
def partition(x):
    if x =='negative':
        return 0
    return 1
```

In [200]:
```
#changing reviews with score less than 3 to be positive and vice-versa
actualScore = label
positiveNegative = actualScore.map(partition)
label= positiveNegative
```

In [201]:
```
actualScore = label1
positiveNegative = actualScore.map(partition)
label1= positiveNegative
```

In [202]:
```
# ============================== loading libraries ==================================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

```
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
# ==========================================================================
```

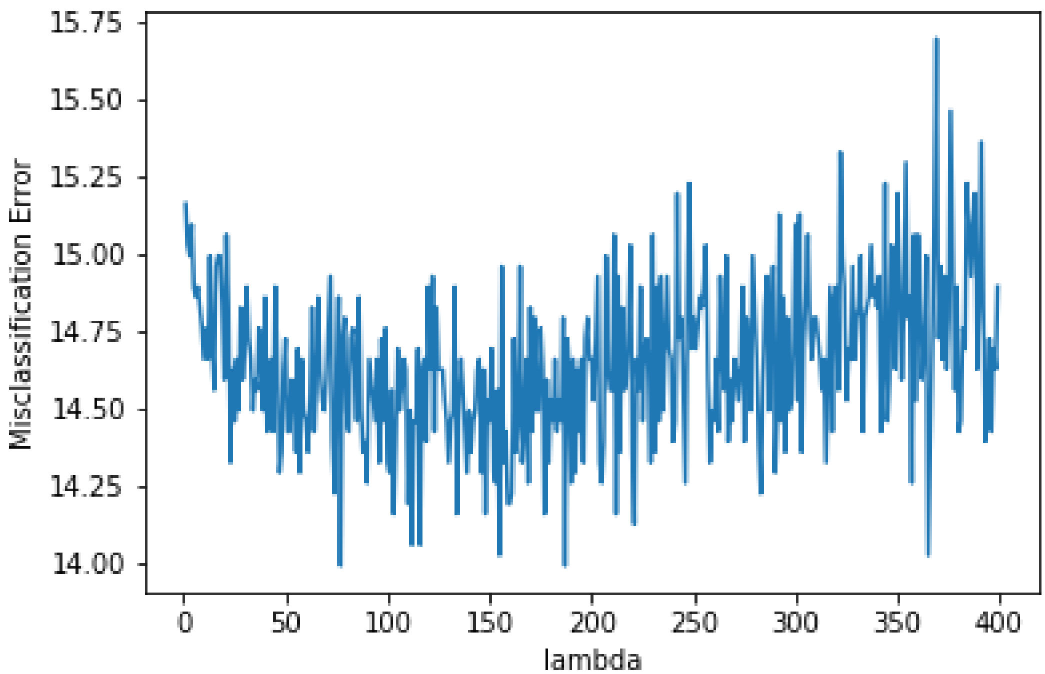
In [203]:
```
X_test, y_test = standardized_data1, label1

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(standardized_data, label,
```

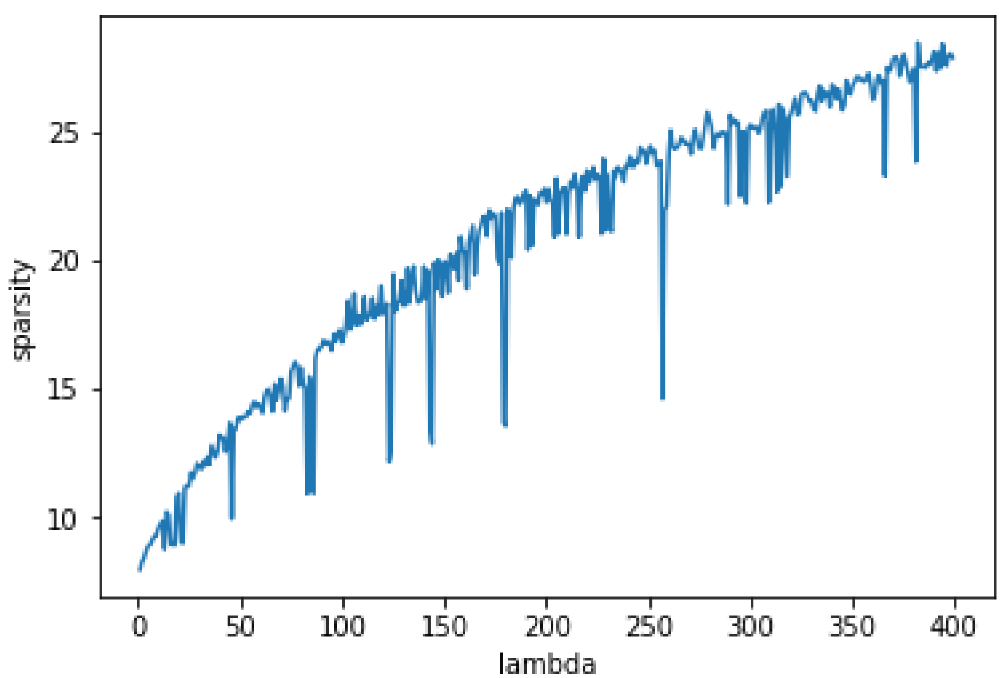
In [204]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection  import GridSearchCV
from sklearn.linear_model import LogisticRegressionCV
```

2.1 L1 regularizer

In [82]:
```
#a=166.81
scores = []
sparsity=[]
neighbors=range(1,400)
for a in neighbors:
    lr_optimal = LogisticRegression(penalty='l1',C=a)

    # fitting the model
    lr_optimal.fit(X_tr, y_tr)

    # predict the response
    pred = lr_optimal.predict(X_test)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    scores.append(acc)

    coef_l1_LR = lr_optimal.coef_.ravel()

    sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100

    sparsity.append(sparsity_l1_LR)


MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('lambda')
plt.ylabel('Misclassification Error')
plt.show()
```

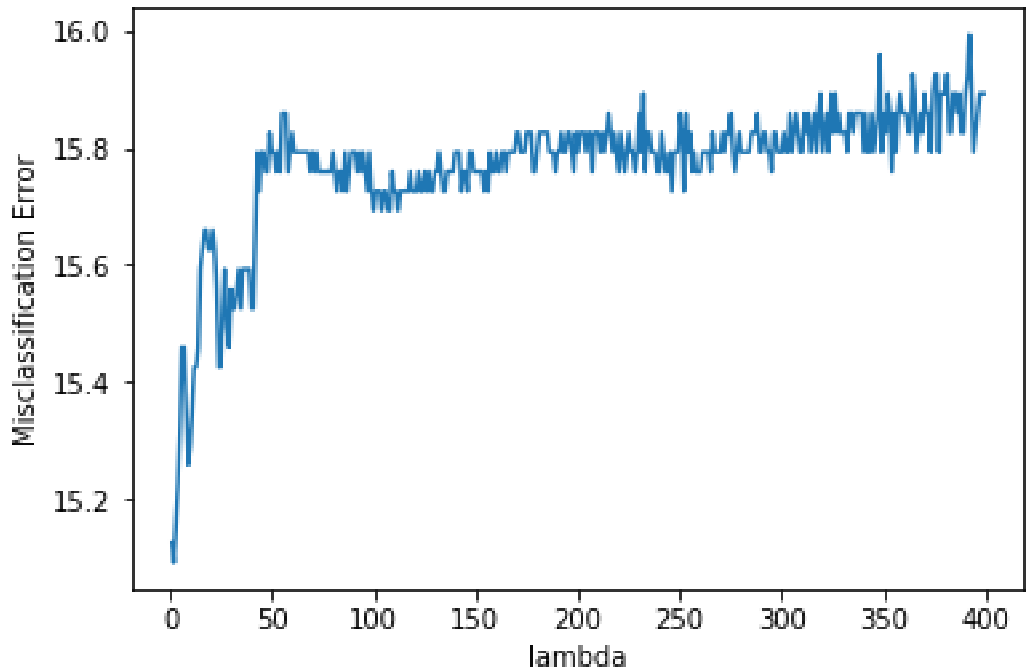
```
plt.plot(neighbors,sparsity)
plt.xlabel('lambda')
plt.ylabel('sparsity')
plt.show()
```

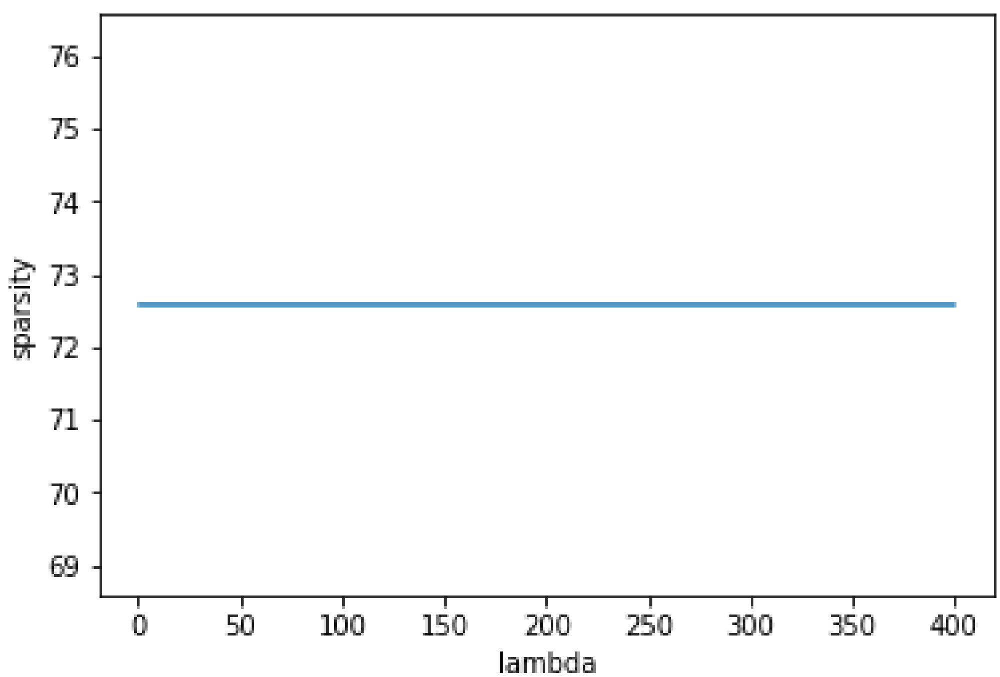




2.2 L2 regularizer

In [83]:
```
#a=166.81
scores = []
sparsity=[]
neighbors=range(1,400)
for a in neighbors:
    lr_optimal = LogisticRegression(penalty='l2',C=a)

    # fitting the model
    lr_optimal.fit(X_tr, y_tr)

    # predict the response
    pred = lr_optimal.predict(X_test)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    scores.append(acc)

    coef_l1_LR = lr_optimal.coef_.ravel()

    sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100

    sparsity.append(sparsity_l1_LR)


MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('lambda')
plt.ylabel('Misclassification Error')
plt.show()

plt.plot(neighbors,sparsity)
plt.xlabel('lambda')
plt.ylabel('sparsity')
plt.show()
```


16.0
15.8
15.6
15.4
15.2
Misclassification Error
0
50
100
150
200
250
300
350
400
lambda


76
75
74
73
72
71
70
69
sparsity
0
50
100
150
200
250
300
350
400
lambda

2.3 GridSearchCV

In [ ]:
```
knn = LogisticRegression()
grid_values = {'penalty': ['l1','l2'], 'C': range(1,500)}
clf = GridSearchCV(knn,param_grid=grid_values)

# fitting the model on crossvalidation train
best_model=clf.fit(X_tr, y_tr)

# predict the response on the crossvalidation train
```

In [85]:
```
print('Best Penalty:', best_model.best_estimator_.get_params()['penalty'])
print('Best C:', best_model.best_estimator_.get_params()['C'])
a=best_model.best_estimator_.get_params()['C']
```

```
Best Penalty: l1
Best C: 341
```

In [86]:
```
pred = best_model.predict(X_cv)
pred
```

Out[86]:
```
array([1, 0, 0, ..., 0, 1, 1], dtype=int64)
```

In [207]:
```
lr_optimal = LogisticRegression(penalty='l1',C=a)

# fitting the model
lr_optimal.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal.predict(X_cv)

# evaluate accuracy
acc = accuracy_score(y_cv, pred) * 100
print('\nThe train accuracy of the knn classifier for  = %d with l1 regularizer is %
```

```
The train accuracy of the knn classifier for  = 341 with l1 regularizer is 87.982833%
```

In [87]:
```
lr_optimal = LogisticRegression(penalty='l1',C=a)

# fitting the model
lr_optimal.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe test accuracy of the knn classifier for  = %d with l1 regularizer is %f%
```

```
The test accuracy of the knn classifier for  = 341 with l1 regularizer is 85.175292%
```

```
In [88]: lr_optimal1 = LogisticRegression(penalty='l2',C=a)

         # fitting the model
         lr_optimal1.fit(X_tr, y_tr)

         # predict the response
         pred = lr_optimal1.predict(X_test)

         # evaluate accuracy
         acc = accuracy_score(y_test, pred) * 100
         print('\nThe test accuracy of the knn classifier for  = %d with l2 regularizer is %f%
```

```
The test accuracy of the knn classifier for  = 341 with l2 regularizer is 84.140234%
```

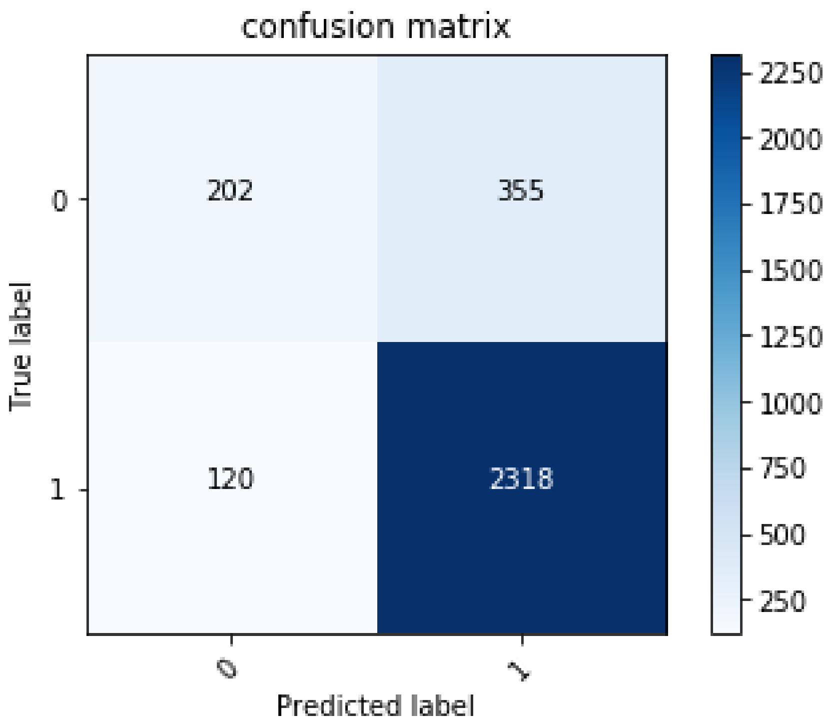
```
In [89]: coef_l1_LR = lr_optimal.coef_.ravel()
         coef_l2_LR = lr_optimal1.coef_.ravel()

         sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100
         sparsity_l2_LR = np.mean(coef_l2_LR != 0) * 100

         print("Sparsity with L1 penalty: %.2f%%" % sparsity_l1_LR)
         print("Sparsity with L2 penalty: %.2f%%" % sparsity_l2_LR)
```

```
Sparsity with L1 penalty: 26.31%
Sparsity with L2 penalty: 72.58%
```

```
In [90]: import itertools

         confusion = confusion_matrix(y_test, pred)
         print(confusion)
         plt.imshow(confusion,cmap=plt.cm.Blues)
         plt.title('confusion matrix')

         plt.colorbar()
         tick_marks = np.arange(2)
         plt.xticks(tick_marks, rotation=45)
         plt.yticks(tick_marks)

         fmt = 'd'
         thresh = confusion.max() / 2
         for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
             plt.text(j, i, format(confusion[i, j], fmt),
```

```
                      horizontalalignment="center",
                      color="white" if confusion[i, j] > thresh else "black")


plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.show()
```

```
[[ 202  355]
 [ 120 2318]]
```



2.4 RandomSearch

```
In [91]: from sklearn.model_selection import RandomizedSearchCV
         knn = LogisticRegression()
         grid_values = {'penalty': ['l1','l2'], 'C': range(1,500)}
         clf = RandomizedSearchCV(knn,param_distributions=grid_values)

         # fitting the model on crossvalidation train
         best_model=clf.fit(X_tr, y_tr)

         # predict the response on the crossvalidation train
```

In [92]:
```
print('Best C:', best_model.best_estimator_.get_params()['C'])
a=best_model.best_estimator_.get_params()['C']
```

```
Best C: 37
```

In [212]:
```
lr_optimal = LogisticRegression(penalty='l1',C=a)

# fitting the model
lr_optimal.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal.predict(X_cv)

# evaluate accuracy
acc = accuracy_score(y_cv, pred) * 100
print('\nThe train accuracy of the knn classifier for  = %d with l1 regularizer is %
```

```
The train accuracy of the knn classifier for  = 37 with l1 regularizer is 88.841202%
```

In [93]:
```
lr_optimal = LogisticRegression(penalty='l1',C=a)

# fitting the model
lr_optimal.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe test accuracy of the knn classifier for  = %d with l1 regularizer is %f%
```

```
The test accuracy of the knn classifier for  = 37 with l1 regularizer is 85.342237%
```

In [94]:
```
# ============================== KNN with k = optimal_k ==============================
# instantiate learning model k = optimal_k

lr_optimal1 = LogisticRegression(penalty='l2',C=a)

# fitting the model
lr_optimal1.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal1.predict(X_test)
```

```
    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    print('\nThe test accuracy of the knn classifier for  = %d with l2 regularizer is %f%
```

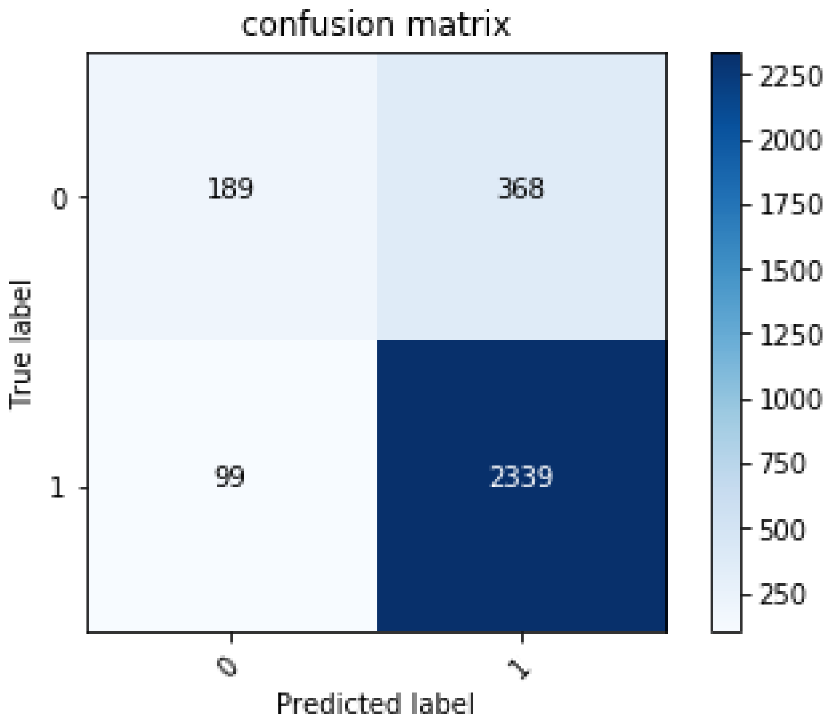
```
The test accuracy of the knn classifier for  = 37 with l2 regularizer is 84.407346%
```

```
In [95]: coef_l1_LR = lr_optimal.coef_.ravel()
         coef_l2_LR = lr_optimal1.coef_.ravel()

         sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100
         sparsity_l2_LR = np.mean(coef_l2_LR != 0) * 100

         print("Sparsity with L1 penalty: %.2f%%" % sparsity_l1_LR)
         print("Sparsity with L2 penalty: %.2f%%" % sparsity_l2_LR)
```

```
Sparsity with L1 penalty: 12.31%
Sparsity with L2 penalty: 72.58%
```

```
In [96]: import itertools

         confusion = confusion_matrix(y_test, pred)
         print(confusion)
         plt.imshow(confusion,cmap=plt.cm.Blues)
         plt.title('confusion matrix')

         plt.colorbar()
         tick_marks = np.arange(2)
         plt.xticks(tick_marks, rotation=45)
         plt.yticks(tick_marks)

         fmt = 'd'
         thresh = confusion.max() / 2
         for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
             plt.text(j, i, format(confusion[i, j], fmt),
                      horizontalalignment="center",
                      color="white" if confusion[i, j] > thresh else "black")


         plt.ylabel('True label')
         plt.xlabel('Predicted label')

         plt.show()
```

```
[[ 189  368]
 [  99 2339]]
```



2.5 feature Importance

2.5.1 Multi-collinearity

```
In [97]: count=0
         for i in lr_optimal.coef_[0]:
             count+=1

In [98]: print(count)

22825

In [99]: import copy
         new=copy.copy(final_counts)

In [100]: new=np.reshape(np.matrix(np.random.normal(loc=0,scale=0.01,size=final_counts.shape[0]

In [101]: import tqdm

In [102]: new.shape

Out[102]: (9982, 22825)

In [103]: final_counts.shape
```

Out[103]: (9982, 22825)

In [104]: mat=new+final_counts

In [105]: mat

Out[105]: matrix([[0.00214642, 0.0052462 , -0.00192545, ..., -0.00203866,
          0.00330998, 0.00623343],
        [0.00782629, 0.00549193, -0.00400983, ..., -0.0273219 ,
         -0.0024257 , -0.00084757],
        [-0.00656472, 0.00427547, -0.00232741, ..., -0.00889617,
          0.00840334, 0.01181426],
        ...,
        [-0.00313937, -0.01172267, -0.01157534, ..., -0.00894108,
          0.00136741, 0.01043953],
        [-0.00055448, -0.00504435, 0.00187086, ..., 0.00010495,
         -0.02093107, -0.01826474],
        [0.01500703, 0.00727658, -0.00668015, ..., -0.00297361,
         -0.00812724, -0.00058376]])

In [106]:
```
per=int(0.7*mat.shape[0])
c=mat[0:per]
d=mat[per:]
```

In [107]:
```
from sklearn.preprocessing import StandardScaler
standardized_data2 = StandardScaler(with_mean=False).fit_transform(c)
print(standardized_data2.shape)
standardized_data3 = StandardScaler(with_mean=False).fit_transform(d)
print(standardized_data3.shape)
```

```
(6987, 22825)
(2995, 22825)
```

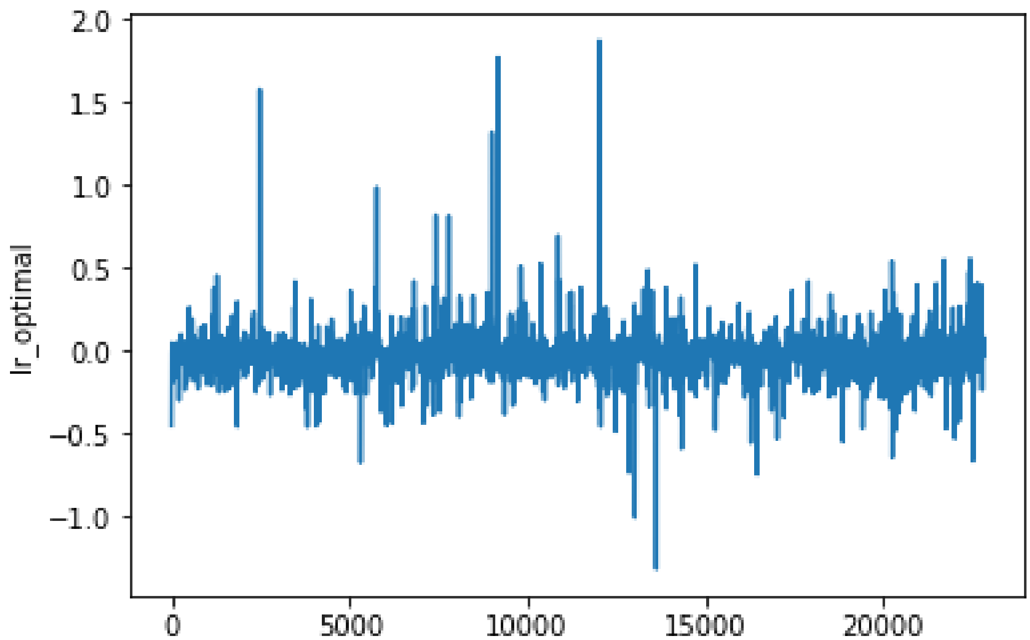
In [108]:
```
X_test1, y_test1 = standardized_data3, label1

# split the train data set into cross validation train and cross validation test
X_tr1, X_cv1, y_tr1, y_cv1 = cross_validation.train_test_split(standardized_data2, la
```

In [213]:
```
lr_optimal1 = LogisticRegression(penalty='l1',C=a)

# fitting the model
lr_optimal1.fit(X_tr1, y_tr1)

# predict the response
pred = lr_optimal1.predict(X_test1)

# evaluate accuracy
acc = accuracy_score(y_test1, pred) * 100
print('\nThe test accuracy of the knn classifier for  = 37 with l1 regularizer is %f%
```

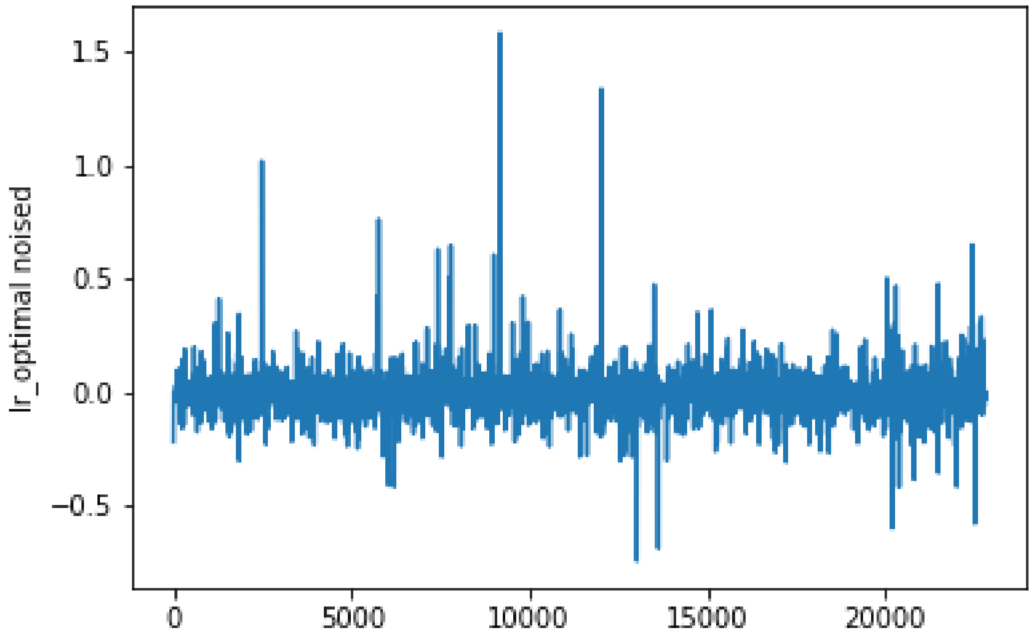
```
The test accuracy of the knn classifier for  = 37 with l1 regularizer is 81.818182%
```

```
In [110]: diff=((lr_optimal.coef_-lr_optimal1.coef_))
```

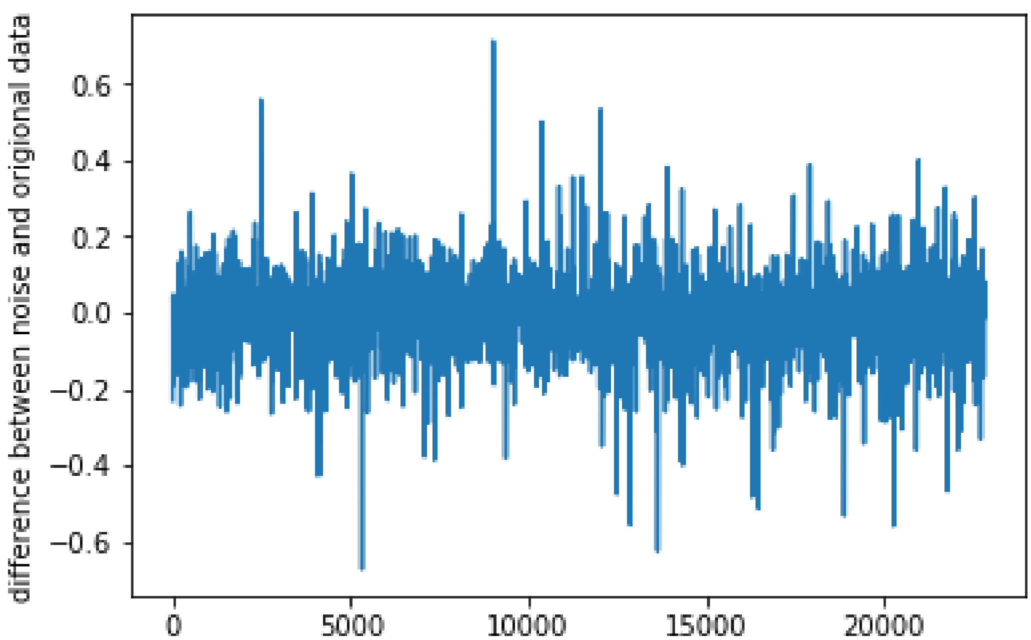
```
In [111]: import matplotlib.pyplot as plt
          plt.plot(lr_optimal.coef_[0])
          plt.ylabel('lr_optimal')
          plt.show()
```




```
In [112]: plt.plot(lr_optimal1.coef_[0])
          plt.ylabel('lr_optimal noised')
          plt.show()
```



In [113]:
```
plt.plot(diff[0])
plt.ylabel('difference between noise and origional data')
plt.show()
```



They are not collinear as weight doesn't change lot. So we can find important features

```
In [114]: dict ={k: v for v, k in enumerate(lr_optimal.coef_[0])}

In [115]: imp = [dict[k] for k in sorted(dict.keys())[:50]]

In [116]: print(list(count_vect.vocabulary_.keys())[list(count_vect.vocabulary_.values()).inde
```

```
money
```

2.5.2 These are the top 50 features from BoW

```
In [117]: for i in imp:
              for j in count_vect.vocabulary_.values():
                  if i==j:
                      print(list(count_vect.vocabulary_.keys())[list(count_vect.vocabulary_.va
                      pass
```

```
not
money
refund
minute
crunchy
worst
thats
paid
receive
spend
rip
waste
maybe
thought
popped
vinegar
stuck
changed
awful
luck
chocolates
died
00
did
weak
disappointed
ended
cinnamon
didn
weird
rubbery
```

```
flat
own
event
gumballs
disappointing
thus
worse
description
expiration
triple
retail
were
chalk
terrible
ok
lox
neither
don
hates
```

3 Tf-idf

In [118]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(review['Text'].values)
```

In [214]:
```
per=int(0.7*final_tf_idf.shape[0])
a=final_tf_idf[0:per]
b=final_tf_idf[per:]
label=review.Score[0:per]
label1=review.Score[per:]
```

In [215]:
```
actualScore = label
positiveNegative = actualScore.map(partition)
label= positiveNegative
```

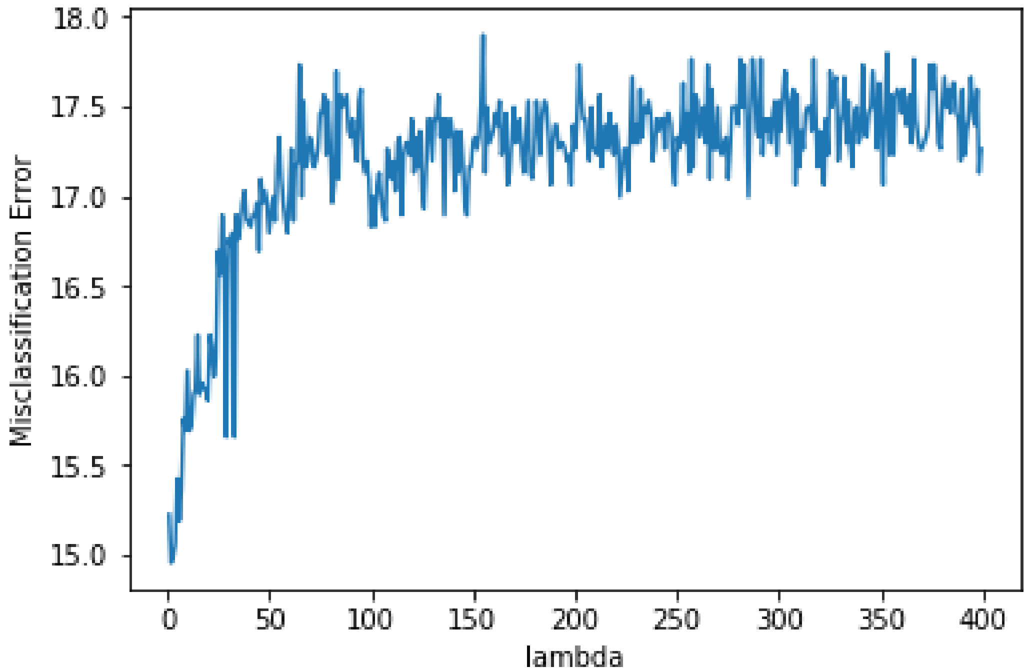
In [216]:
```
actualScore = label1
positiveNegative = actualScore.map(partition)
label1= positiveNegative
```

In [217]:
```
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=False).fit_transform(a)
standardized_data1 = StandardScaler(with_mean=False).fit_transform(b)
```

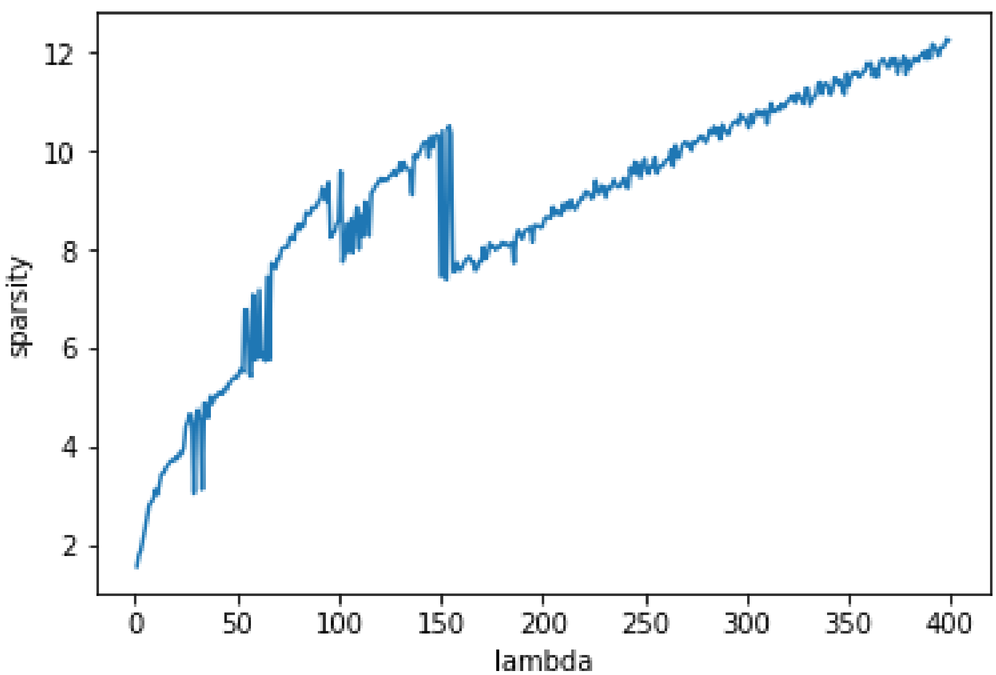
In [218]:
```
X_test, y_test = standardized_data1, label1

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(standardized_data, label,
```

3.1 L1 regularizer

In [124]:
```
#a=166.81
scores = []
sparsity=[]
neighbors=range(1,400)
for a in neighbors:
    lr_optimal = LogisticRegression(penalty='l1',C=a)

    # fitting the model
    lr_optimal.fit(X_tr, y_tr)

    # predict the response
    pred = lr_optimal.predict(X_test)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    scores.append(acc)

    coef_l1_LR = lr_optimal.coef_.ravel()

    sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100

    sparsity.append(sparsity_l1_LR)


MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('lambda')
plt.ylabel('Misclassification Error')
plt.show()

plt.plot(neighbors,sparsity)
plt.xlabel('lambda')
plt.ylabel('sparsity')
plt.show()
```



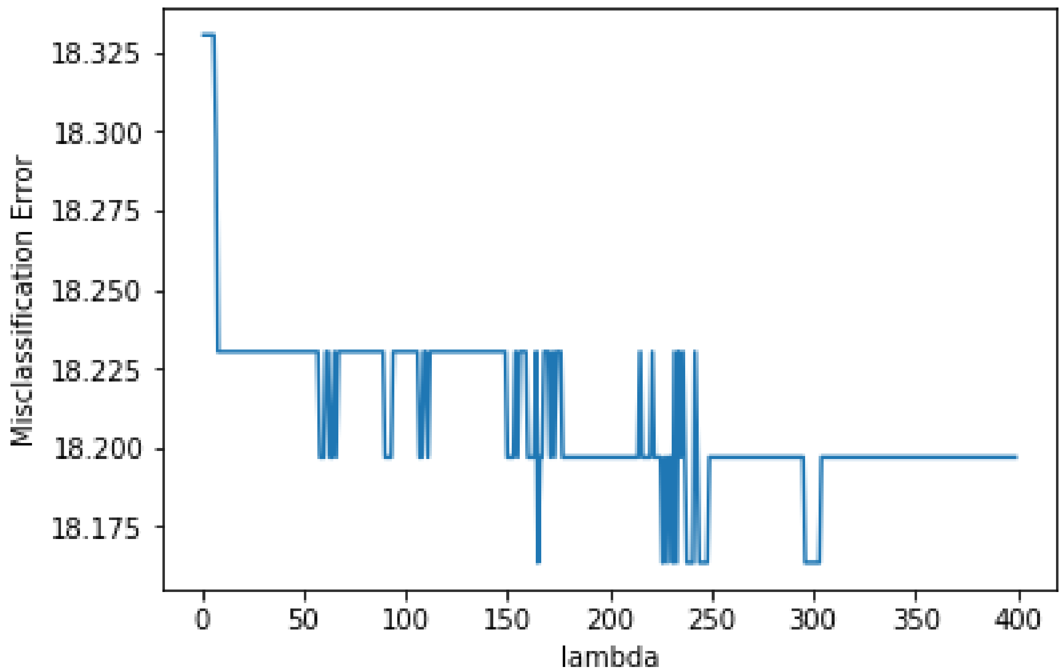
Misclassification Error
18.0
17.5
17.0
16.5
16.0
15.5
15.0
0
50
100
150
200
250
300
350
400
lambda

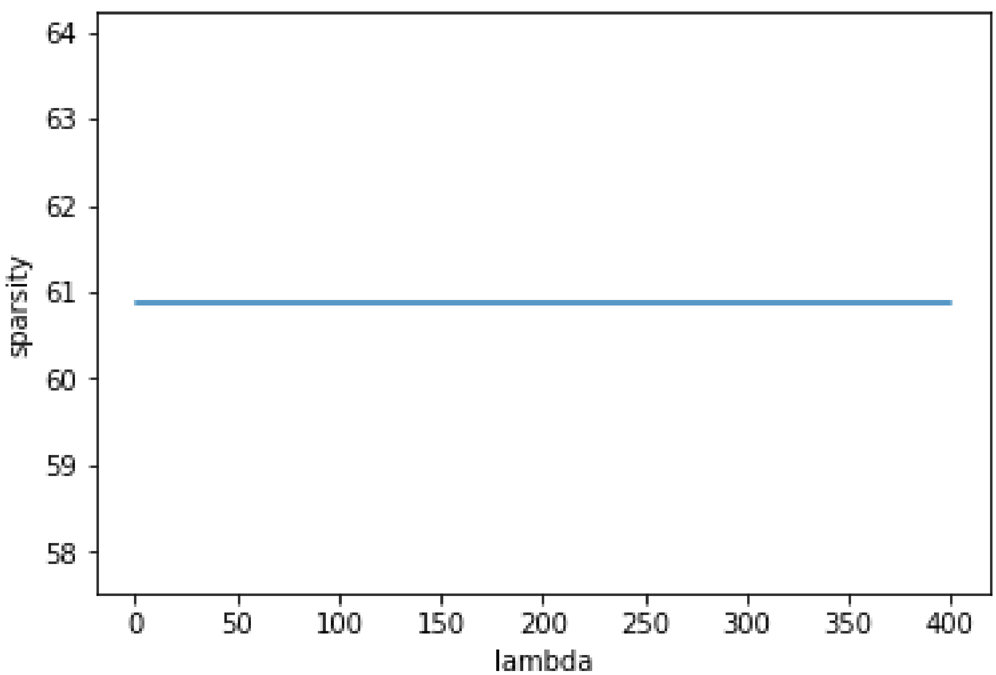


sparsity
12
10
8
6
4
2
0
50
100
150
200
250
300
350
400
lambda

3.2 L2 regularizer

In [125]:
```
#a=166.81
scores = []
sparsity=[]
neighbors=range(1,400)
for a in neighbors:
    lr_optimal = LogisticRegression(penalty='l2',C=a)

    # fitting the model
    lr_optimal.fit(X_tr, y_tr)

    # predict the response
    pred = lr_optimal.predict(X_test)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    scores.append(acc)

    coef_l1_LR = lr_optimal.coef_.ravel()

    sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100

    sparsity.append(sparsity_l1_LR)


MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('lambda')
plt.ylabel('Misclassification Error')
plt.show()

plt.plot(neighbors,sparsity)
plt.xlabel('lambda')
plt.ylabel('sparsity')
plt.show()
```


18.325
18.300
18.275
18.250
18.225
18.200
18.175
Misclassification Error
0
50
100
150
200
250
300
350
400
lambda


64
63
62
61
60
59
58
sparsity
0
50
100
150
200
250
300
350
400
lambda

3.3 Gridsearch

In [126]:
```
knn = LogisticRegression()
grid_values = {'penalty': ['l1','l2'], 'C': range(1,500)}
clf = GridSearchCV(knn,param_grid=grid_values)


best_model=clf.fit(X_tr, y_tr)


print('Best Penalty:', best_model.best_estimator_.get_params()['penalty'])
print('Best C:', best_model.best_estimator_.get_params()['C'])
a=best_model.best_estimator_.get_params()['C']
```

```
Best Penalty: l1
Best C: 3
```

In [219]:
```
lr_optimal = LogisticRegression(penalty='l1',C=3)

# fitting the model
lr_optimal.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal.predict(X_cv)

# evaluate accuracy
acc = accuracy_score(y_cv, pred) * 100
print('\nThe train accuracy of the knn classifier for  = %d with l1 regularizer is %
```

```
The train accuracy of the knn classifier for  = 3 with l1 regularizer is 88.078207%
```

In [127]:
```
lr_optimal = LogisticRegression(penalty='l1',C=a)

# fitting the model
lr_optimal.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe test accuracy of the knn classifier for  = %d with l1 regularizer is %f%
```

```
The test accuracy of the knn classifier for  = 3 with l1 regularizer is 84.974958%
```

In [128]:
```
lr_optimal1 = LogisticRegression(penalty='l2',C=a)

# fitting the model
lr_optimal1.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal1.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe test accuracy of the knn classifier for  = %d with l2 regularizer is %f%
```

```
The test accuracy of the knn classifier for  = 3 with l2 regularizer is 81.669449%
```

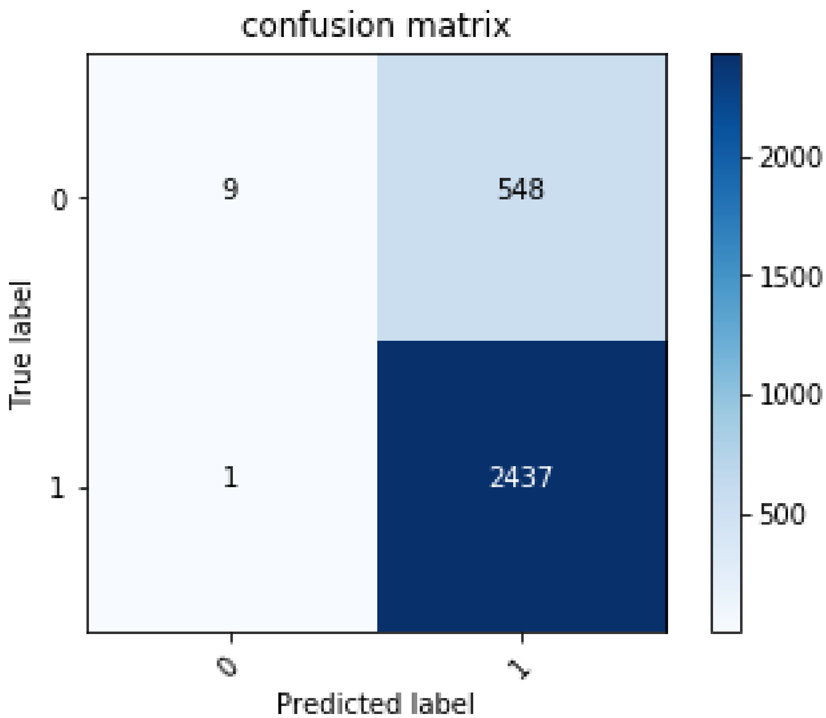
In [129]:
```
coef_l1_LR = lr_optimal.coef_.ravel()
coef_l2_LR = lr_optimal1.coef_.ravel()

sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100
sparsity_l2_LR = np.mean(coef_l2_LR != 0) * 100

print("Sparsity with L1 penalty: %.2f%%" % sparsity_l1_LR)
print("Sparsity with L2 penalty: %.2f%%" % sparsity_l2_LR)
```

```
Sparsity with L1 penalty: 2.06%
Sparsity with L2 penalty: 60.87%
```

In [130]:
```
import itertools

confusion = confusion_matrix(y_test, pred)
print(confusion)
plt.imshow(confusion,cmap=plt.cm.Blues)
plt.title('confusion matrix')

plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, rotation=45)
plt.yticks(tick_marks)

fmt = 'd'
thresh = confusion.max() / 2
for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
    plt.text(j, i, format(confusion[i, j], fmt),
             horizontalalignment="center",
             color="white" if confusion[i, j] > thresh else "black")
```

```
plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.show()
```

```
[[   9  548]
 [   1 2437]]
```



3.4 RandomSearch

```
In [131]: from sklearn.model_selection import RandomizedSearchCV
          knn = LogisticRegression()
          grid_values = {'penalty': ['l1','l2'], 'C': range(1,500)}
          clf = RandomizedSearchCV(knn,param_distributions=grid_values)

          # fitting the model on crossvalidation train
          best_model=clf.fit(X_tr, y_tr)

          # predict the response on the crossvalidation train

          print('Best C:', best_model.best_estimator_.get_params()['C'])
          a=best_model.best_estimator_.get_params()['C']
```

```
Best C: 64
```

```
In [220]: lr_optimal = LogisticRegression(penalty='l1',C=64)

          # fitting the model
          lr_optimal.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe train accuracy of the knn classifier for  = %d with l1 regularizer is %
```

```
The train accuracy of the knn classifier for  = 64 with l1 regularizer is 83.005008%
```

```
In [132]: lr_optimal = LogisticRegression(penalty='l1',C=a)

          # fitting the model
          lr_optimal.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe test accuracy of the knn classifier for  = %d with l1 regularizer is %f
```

```
The test accuracy of the knn classifier for  = 64 with l1 regularizer is 82.904841%
```

```
In [133]: # ============================== KNN with k = optimal_k ===========================
          # instantiate learning model k = optimal_k

          lr_optimal1 = LogisticRegression(penalty='l2',C=a)

          # fitting the model
          lr_optimal1.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal1.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe test accuracy of the knn classifier for  = %d with l2 regularizer is %f
```

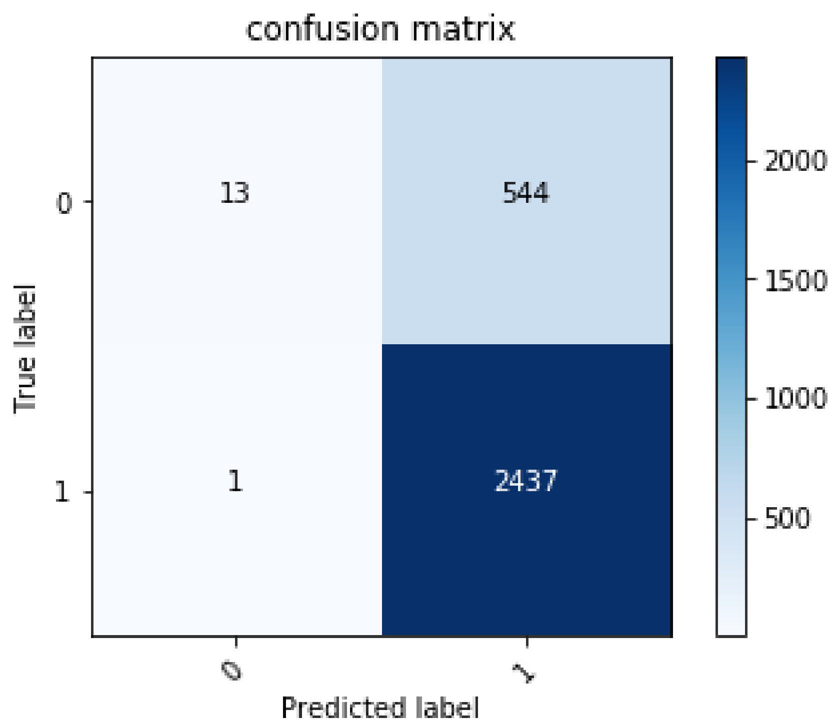
```
The test accuracy of the knn classifier for  = 64 with l2 regularizer is 81.803005%
```

```
In [134]: coef_l1_LR = lr_optimal.coef_.ravel()
          coef_l2_LR = lr_optimal1.coef_.ravel()

          sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100
          sparsity_l2_LR = np.mean(coef_l2_LR != 0) * 100

          print("Sparsity with L1 penalty: %.2f%%" % sparsity_l1_LR)
          print("Sparsity with L2 penalty: %.2f%%" % sparsity_l2_LR)
```

```
Sparsity with L1 penalty: 7.44%
Sparsity with L2 penalty: 60.87%
```

```
In [135]: import itertools

          confusion = confusion_matrix(y_test, pred)
          print(confusion)
          plt.imshow(confusion,cmap=plt.cm.Blues)
          plt.title('confusion matrix')

          plt.colorbar()
          tick_marks = np.arange(2)
          plt.xticks(tick_marks, rotation=45)
          plt.yticks(tick_marks)

          fmt = 'd'
          thresh = confusion.max() / 2
          for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
              plt.text(j, i, format(confusion[i, j], fmt),
                        horizontalalignment="center",
                        color="white" if confusion[i, j] > thresh else "black")


          plt.ylabel('True label')
          plt.xlabel('Predicted label')

          plt.show()
```

```
[[  13  544]
 [   1 2437]]
```



3.5 Feature Importance

3.5.1 Multi-collinearity

In [136]:
```
origional=final_tf_idf[0:500]
```

In [137]:
```
new=np.reshape(np.matrix(np.random.normal(loc=0,scale=0.01,size=origional.shape[0]*o
```

In [138]:
```
mat=new+origional
```

In [139]:
```
per=int(0.7*mat.shape[0])
c=mat[0:per]
d=mat[per:]
```

In [140]:
```
from sklearn.preprocessing import StandardScaler
standardized_data2 = StandardScaler(with_mean=False).fit_transform(c)
print(standardized_data2.shape)
standardized_data3 = StandardScaler(with_mean=False).fit_transform(d)
print(standardized_data3.shape)
```

```
(350, 284906)
(150, 284906)
```

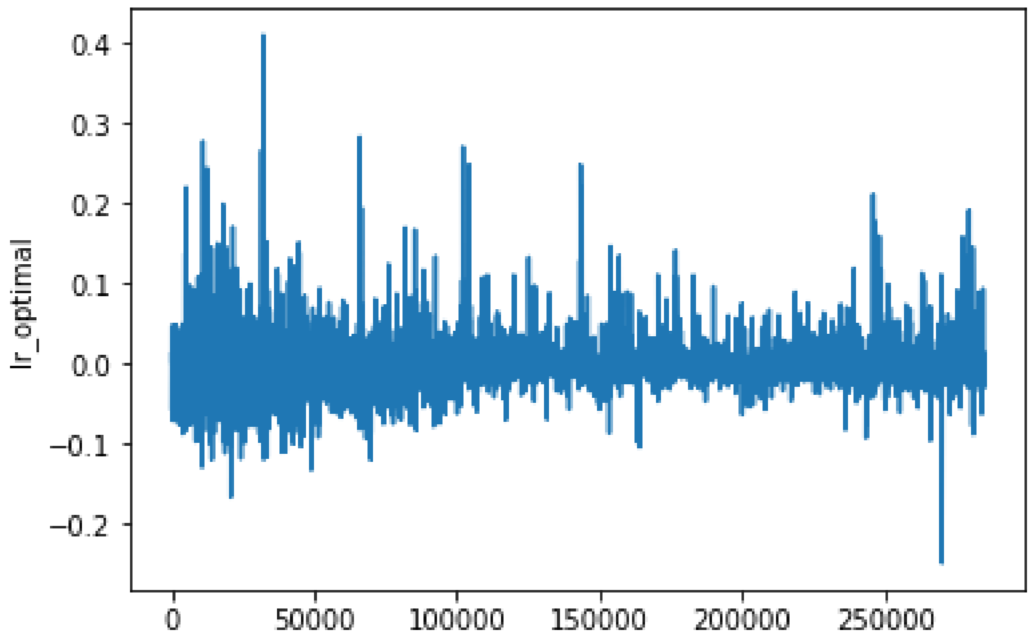
In [141]:
```
X_test1, y_test1 = standardized_data3, label1[0:150]

# split the train data set into cross validation train and cross validation test
X_tr1, X_cv1, y_tr1, y_cv1 = cross_validation.train_test_split(standardized_data2, la
```

In [221]:
```
lr_optimal1 = LogisticRegression(penalty='l1',C=a)

# fitting the model
lr_optimal1.fit(X_tr1, y_tr1)

# predict the response
pred = lr_optimal1.predict(X_test1)

# evaluate accuracy
acc = accuracy_score(y_test1, pred) * 100
print('\nThe test accuracy of the knn classifier for  = 64 with l1 regularizer is %f%
```

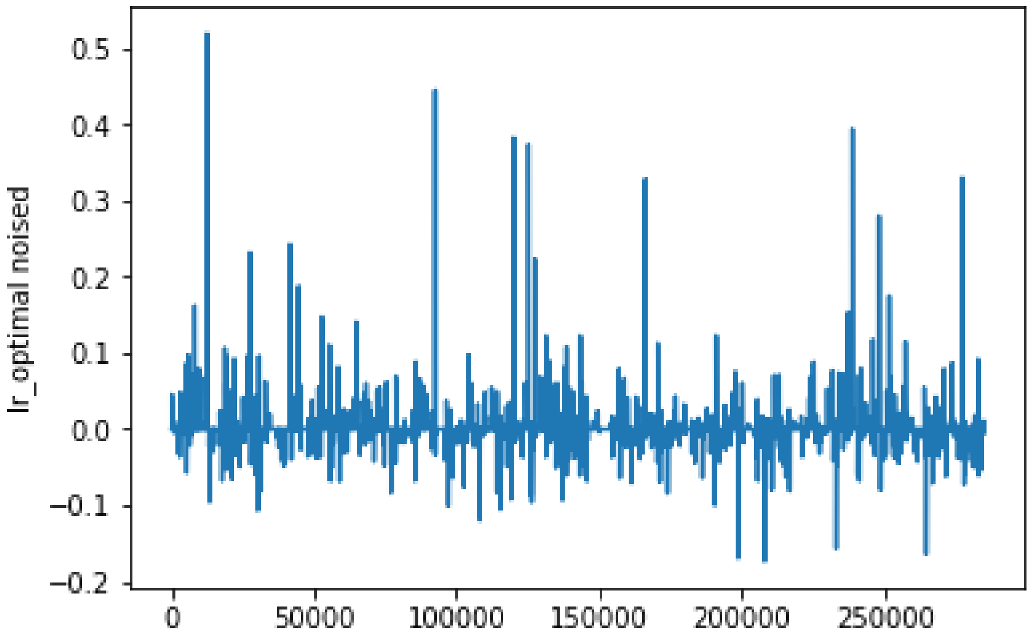
```
The test accuracy of the knn classifier for  = 64 with l1 regularizer is 81.818182%
```

In [143]:
```
diff=((lr_optimal.coef_-lr_optimal1.coef_))
```

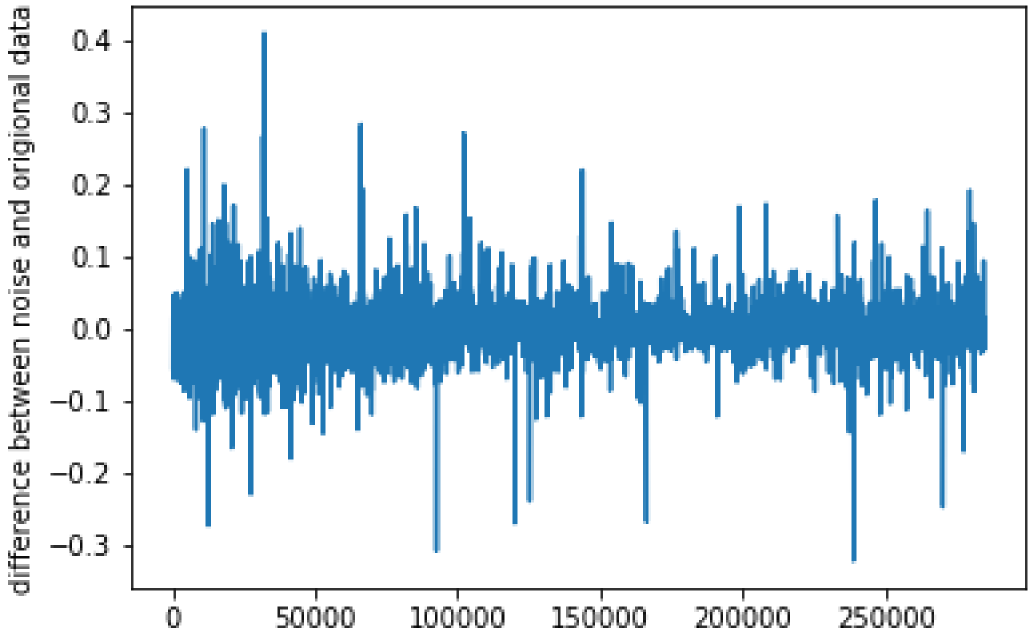
In [144]:
```
import matplotlib.pyplot as plt
plt.plot(lr_optimal.coef_[0])
plt.ylabel('lr_optimal')
plt.show()
```



In [145]:
```
import matplotlib.pyplot as plt
plt.plot(lr_optimal1.coef_[0])
plt.ylabel('lr_optimal noised')
plt.show()
```



In [146]:
```
plt.plot(diff[0])
plt.ylabel('difference between noise and origional data')
plt.show()
```



They are not collinear as weight doesn't change lot. So we can find important features

3.5.2 Top 50 features from TF-idf

```
In [147]: dict ={k: v for v, k in enumerate(lr_optimal.coef_[0])}
          imp = [dict[k] for k in sorted(dict.keys())[:50]]

In [148]: imp

Out[148]: [270203,
           20736,
           49011,
           10726,
           69870,
           14125,
           32289,
           24286,
           33427,
           18685,
           38629,
           39972,
           69875,
           20754,
           164160,
           45360,
           20546,
           42612,
```

```
         13358,
         25652,
         163113,
         17921,
         8478,
         31227,
         9061,
         266110,
         25608,
         68226,
         49389,
         243502,
         31531,
         51505,
         46339,
         14083,
         30861,
         19497,
         281125,
         3922,
         153478,
         69943,
         44713,
         31020,
         15039,
         11118,
         14376,
         20777,
         41235,
         4395,
         13072,
         85809]
```

```
In [149]: print(list(tf_idf_vect.vocabulary_.keys())[list(tf_idf_vect.vocabulary_.values()).inc
```

```
arrived and
```

```
In [150]: for i in imp:
              for j in tf_idf_vect.vocabulary_.values():
                  if i==j:
                      print(list(tf_idf_vect.vocabulary_.keys())[list(tf_idf_vect.vocabulary_.v
                      pass
```

```
waste
arrived and
charged
am extremely
```

disappointed
and hate
better off
awful
bit weird
apple syrup
brand again
broken when
disappointed and
arrived dead
not worth
candy rather
around everywhere
buy anything
and crunchy
bags and
not buy
anymore loved
all just
bellies
almonds in
very disappointed
bag you
didn enjoy
check to
the trash
berries is
chocolate wanted
care about
and gummy
being diagnosed
are high
would not
absolutely ruins
money
disappointing
can even
being rubbery
and persistent
amazing popcorn
and jar
arrived half
burns
acidic better
and caution
favorite kinda

4 W2V

In [151]:
```
import codecs
import glob
import logging
import multiprocessing
import os
import pprint

import gensim.models.word2vec as w2v
import sklearn.manifold

import re

model = w2v.Word2Vec.load("food2vec.w2v")
```

In [152]:
```
# Train your own Word2Vec model using your own text corpus
import gensim
import re
i=0
list_of_sent=[]

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

def cleanpunc(sentence): #function to clean the word of any punctuation or special c
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned


for sent in review['Text'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

In [153]:
```
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [154]:
```
words = list(w2v_model.wv.vocab)
print(len(words))
```

6928

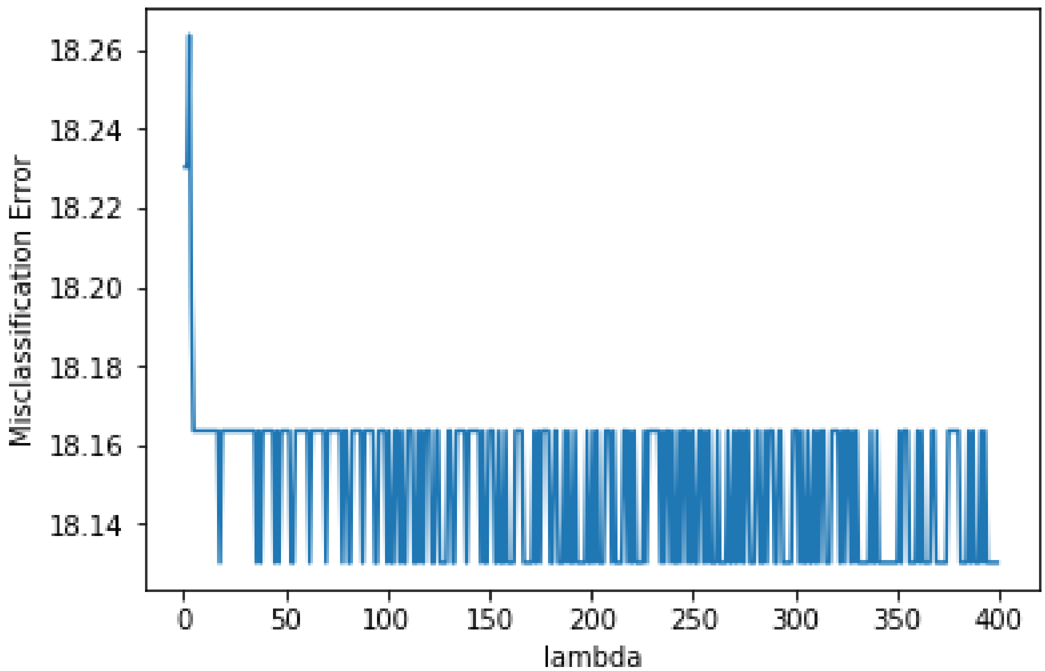
```
In [155]: all_word_vectors_matrix = w2v_model.wv.syn0
```

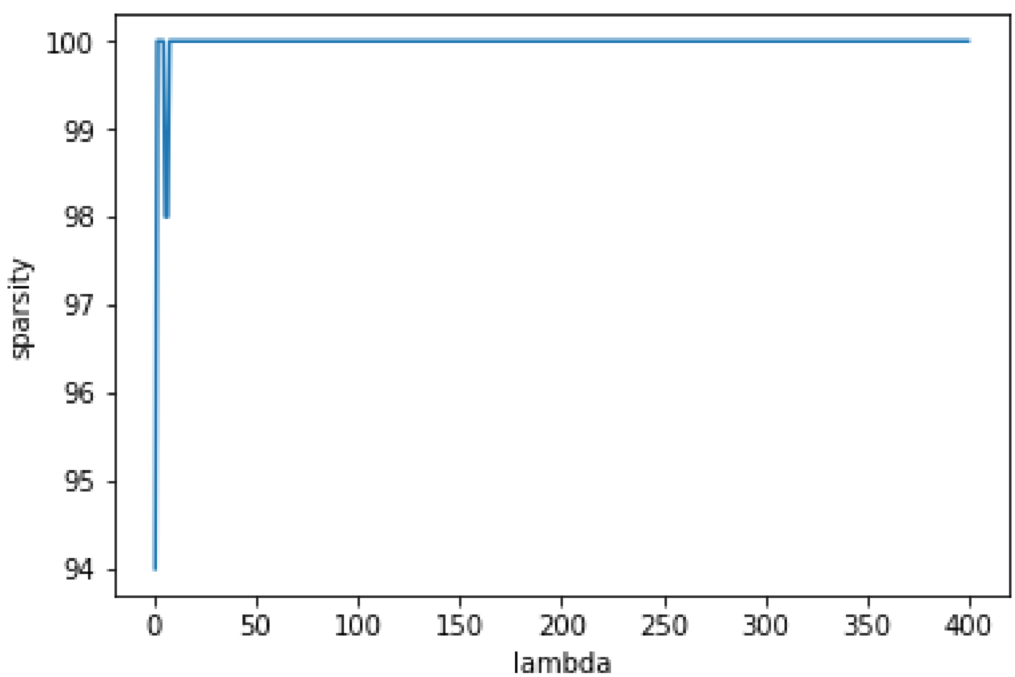
```
c:\users\vishal\appdata\local\programs\python\python36\lib\site-packages\ipykernel_launcher.py
  """Entry point for launching an IPython kernel.
```

```
In [156]: count=0
          temp=np.zeros((1,50))
          revect=np.zeros((1,50))

In [157]: np.seterr(divide='ignore', invalid='ignore')
          count=0
          for i in review['Text']:
              for j in cleanpunc(i).split():
                  if j in words:
                      temp=np.add(temp, all_word_vectors_matrix[words.index(j)])
                      count+=1
              if count!=0:
                  temp=[i/count for i in temp]
              count=0
              revect=np.concatenate((revect,temp))
              temp=np.zeros((1,50))

In [158]: revect=np.delete(revect,0,axis=0)

In [222]: per=int(0.7*revect.shape[0])
          a=revect[0:per]
          b=revect[per:]
          label=review.Score[0:per]
          label1=review.Score[per:9982]

In [223]: actualScore = label
          positiveNegative = actualScore.map(partition)
          label= positiveNegative

In [224]: actualScore = label1
          positiveNegative = actualScore.map(partition)
          label1= positiveNegative

In [225]: from sklearn.preprocessing import StandardScaler
          standardized_data = StandardScaler(with_mean=False).fit_transform(a)
          standardized_data1 = StandardScaler(with_mean=False).fit_transform(b)

In [226]: X_test, y_test = standardized_data1, label1

          # split the train data set into cross validation train and cross validation test
          X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(standardized_data, label,
```

4.1 L1 regularizer

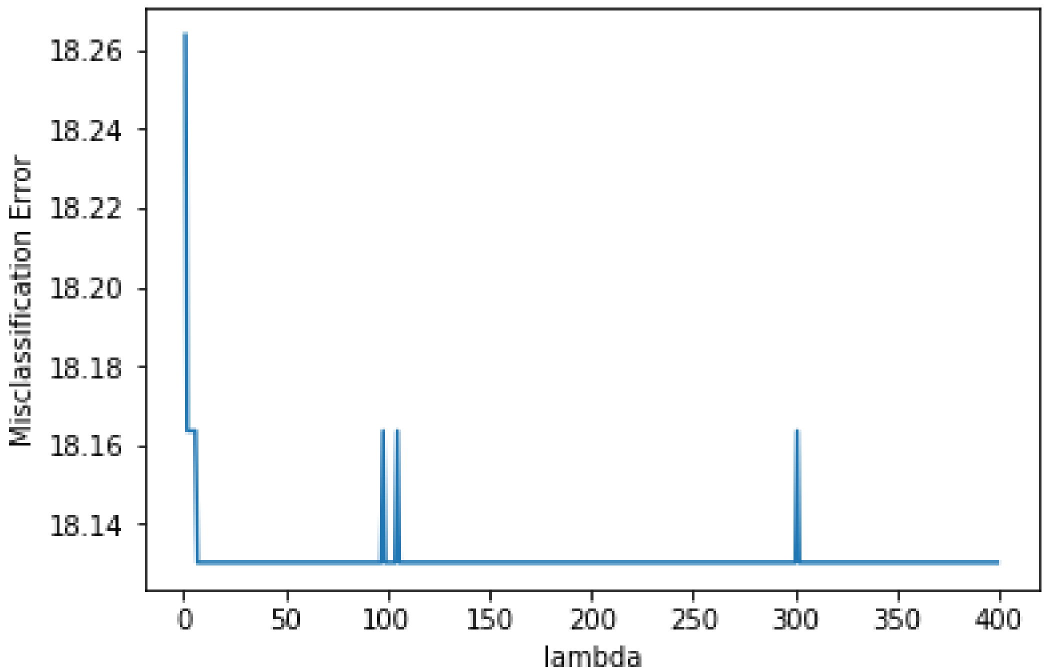
```
In [164]: #a=166.81
          scores = []
          sparsity=[]
          neighbors=range(1,400)
          for a in neighbors:
              lr_optimal = LogisticRegression(penalty='l1',C=a)

              # fitting the model
              lr_optimal.fit(X_tr, y_tr)

              # predict the response
              pred = lr_optimal.predict(X_test)

              # evaluate accuracy
              acc = accuracy_score(y_test, pred) * 100
              scores.append(acc)

              coef_l1_LR = lr_optimal.coef_.ravel()

              sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100

              sparsity.append(sparsity_l1_LR)


          MSE = [100 - x for x in scores]

          plt.plot(neighbors, MSE)

          plt.xlabel('lambda')
          plt.ylabel('Misclassification Error')
          plt.show()

          plt.plot(neighbors,sparsity)
          plt.xlabel('lambda')
          plt.ylabel('sparsity')
          plt.show()
```

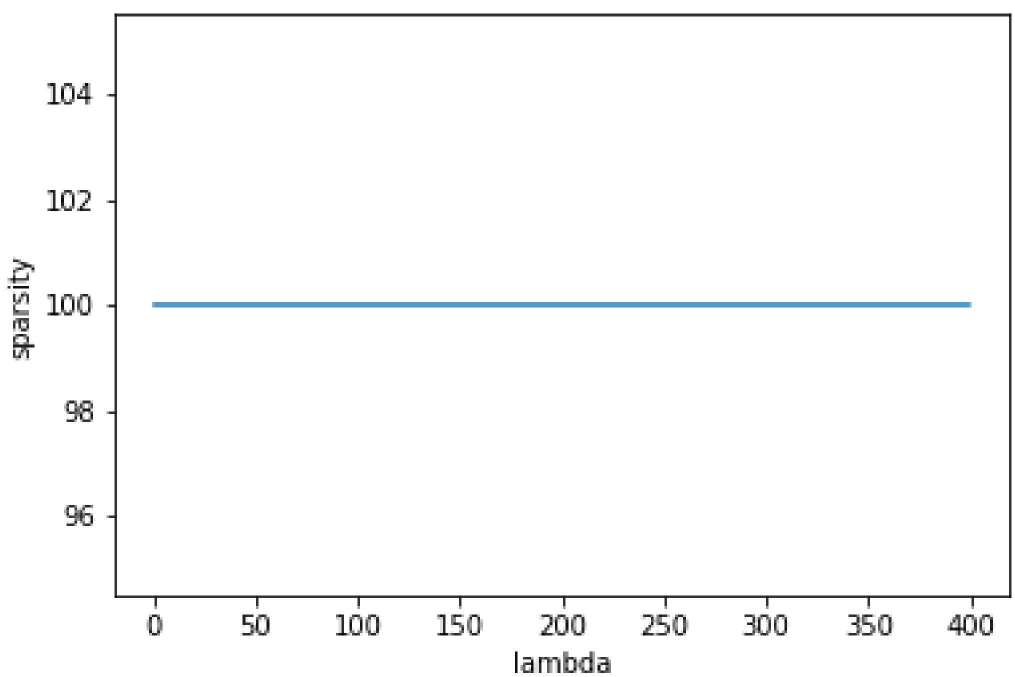

18.26
18.24
18.22
18.20
18.18
18.16
18.14
Misclassification Error
0
50
100
150
200
250
300
350
400
lambda


100
99
98
97
96
95
94
sparsity
0
50
100
150
200
250
300
350
400
lambda

4.2 L2 regularizer

In [165]:
```
#a=166.81
scores = []
sparsity=[]
neighbors=range(1,400)
for a in neighbors:
    lr_optimal = LogisticRegression(penalty='l2',C=a)

    # fitting the model
    lr_optimal.fit(X_tr, y_tr)

    # predict the response
    pred = lr_optimal.predict(X_test)

    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    scores.append(acc)

    coef_l1_LR = lr_optimal.coef_.ravel()

    sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100

    sparsity.append(sparsity_l1_LR)


MSE = [100 - x for x in scores]

plt.plot(neighbors, MSE)

plt.xlabel('lambda')
plt.ylabel('Misclassification Error')
plt.show()

plt.plot(neighbors,sparsity)
plt.xlabel('lambda')
plt.ylabel('sparsity')
plt.show()
```


18.26
18.24
18.22
18.20
18.18
18.16
18.14
Misclassification Error
0
50
100
150
200
250
300
350
400
lambda


104
102
100
98
96
sparsity
0
50
100
150
200
250
300
350
400
lambda

4.3 GridSearch

```
In [166]: knn = LogisticRegression()
          grid_values = {'penalty': ['l1','l2'], 'C': range(1,500)}
          clf = GridSearchCV(knn,param_grid=grid_values)


          best_model=clf.fit(X_tr, y_tr)


          print('Best Penalty:', best_model.best_estimator_.get_params()['penalty'])
          print('Best C:', best_model.best_estimator_.get_params()['C'])
          a=best_model.best_estimator_.get_params()['C']
```

```
Best Penalty: l1
Best C: 1
```

```
In [229]: lr_optimal = LogisticRegression(penalty='l1',C=a)

          # fitting the model
          lr_optimal.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe train accuracy of the knn classifier for  = %d with l1 regularizer is %
```

```
The train accuracy of the knn classifier for  = 1 with l1 regularizer is 81.602671%
```

```
In [167]: lr_optimal = LogisticRegression(penalty='l1',C=a)

          # fitting the model
          lr_optimal.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe test accuracy of the knn classifier for  = %d with l1 regularizer is %f
```

```
The test accuracy of the knn classifier for  = 1 with l1 regularizer is 81.769616%
```

In [168]:
```python
lr_optimal1 = LogisticRegression(penalty='l2',C=a)

# fitting the model
lr_optimal1.fit(X_tr, y_tr)

# predict the response
pred = lr_optimal1.predict(X_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe test accuracy of the knn classifier for  = %d with l2 regularizer is %f%
```

```
The test accuracy of the knn classifier for  = 1 with l2 regularizer is 81.736227%
```

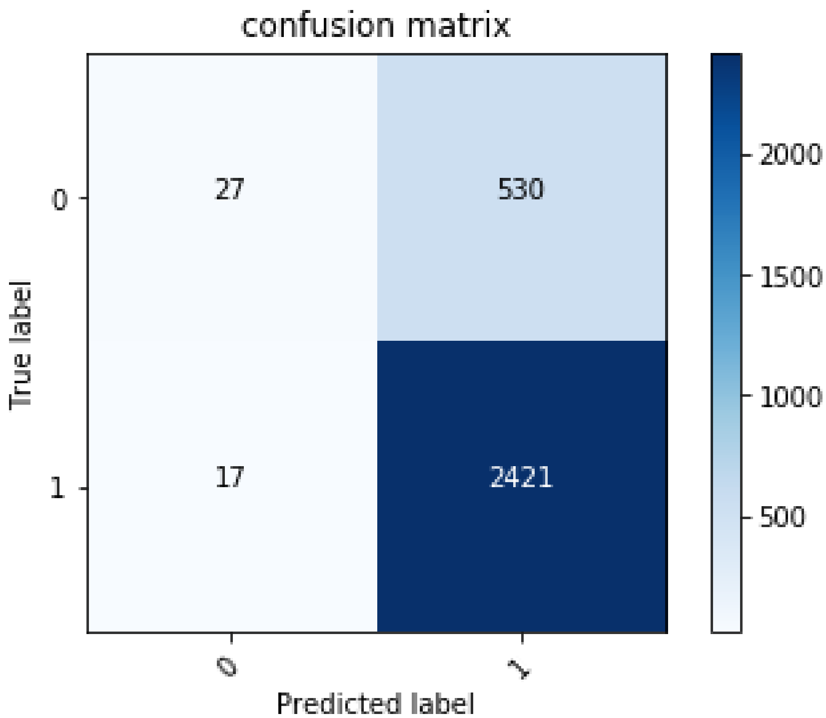
In [169]:
```python
coef_l1_LR = lr_optimal.coef_.ravel()
coef_l2_LR = lr_optimal1.coef_.ravel()

sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100
sparsity_l2_LR = np.mean(coef_l2_LR != 0) * 100

print("Sparsity with L1 penalty: %.2f%%" % sparsity_l1_LR)
print("Sparsity with L2 penalty: %.2f%%" % sparsity_l2_LR)
```

```
Sparsity with L1 penalty: 94.00%
Sparsity with L2 penalty: 100.00%
```

In [170]:
```python
import itertools

confusion = confusion_matrix(y_test, pred)
print(confusion)
plt.imshow(confusion,cmap=plt.cm.Blues)
plt.title('confusion matrix')

plt.colorbar()
tick_marks = np.arange(2)
plt.xticks(tick_marks, rotation=45)
plt.yticks(tick_marks)

fmt = 'd'
thresh = confusion.max() / 2
for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
    plt.text(j, i, format(confusion[i, j], fmt),
             horizontalalignment="center",
             color="white" if confusion[i, j] > thresh else "black")
```

```
plt.ylabel('True label')
plt.xlabel('Predicted label')

plt.show()
```

```
[[  27  530]
 [  17 2421]]
```



4.4 RandomSearch

```
In [171]: from sklearn.model_selection import RandomizedSearchCV
          knn = LogisticRegression()
          grid_values = {'penalty': ['l1','l2'], 'C': range(1,500)}
          clf = RandomizedSearchCV(knn,param_distributions=grid_values)

          # fitting the model on crossvalidation train
          best_model=clf.fit(X_tr, y_tr)

          # predict the response on the crossvalidation train

          print('Best C:', best_model.best_estimator_.get_params()['C'])
          a=best_model.best_estimator_.get_params()['C']
```

```
Best C: 240
```

```
In [228]: lr_optimal = LogisticRegression(penalty='l1',C=240)

          # fitting the model
          lr_optimal.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe train accuracy of the knn classifier for  = %d with l1 regularizer is %
```

```
The train accuracy of the knn classifier for  = 240 with l1 regularizer is 81.836394%
```

```
In [172]: lr_optimal = LogisticRegression(penalty='l1',C=a)

          # fitting the model
          lr_optimal.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe test accuracy of the knn classifier for  = %d with l1 regularizer is %f
```

```
The test accuracy of the knn classifier for  = 240 with l1 regularizer is 81.869783%
```

```
In [173]: # ============================== KNN with k = optimal_k ============================
          # instantiate learning model k = optimal_k

          lr_optimal1 = LogisticRegression(penalty='l2',C=a)

          # fitting the model
          lr_optimal1.fit(X_tr, y_tr)

          # predict the response
          pred = lr_optimal1.predict(X_test)

          # evaluate accuracy
          acc = accuracy_score(y_test, pred) * 100
          print('\nThe test accuracy of the knn classifier for  = %d with l2 regularizer is %f
```

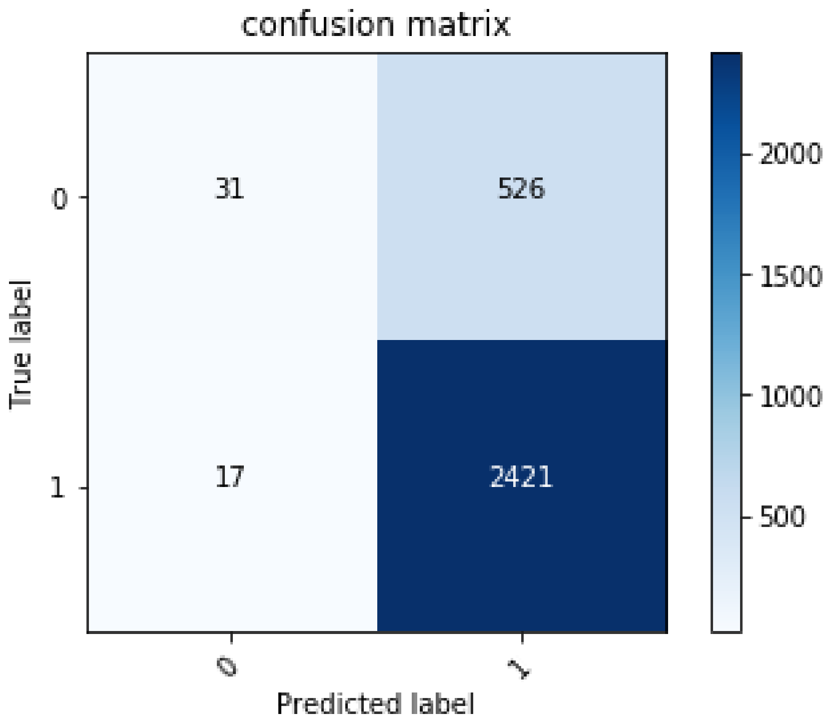
```
The test accuracy of the knn classifier for  = 240 with l2 regularizer is 81.869783%
```

```
In [174]: coef_l1_LR = lr_optimal.coef_.ravel()
          coef_l2_LR = lr_optimal1.coef_.ravel()

          sparsity_l1_LR = np.mean(coef_l1_LR != 0) * 100
          sparsity_l2_LR = np.mean(coef_l2_LR != 0) * 100

          print("Sparsity with L1 penalty: %.2f%%" % sparsity_l1_LR)
          print("Sparsity with L2 penalty: %.2f%%" % sparsity_l2_LR)
```

```
Sparsity with L1 penalty: 100.00%
Sparsity with L2 penalty: 100.00%
```

```
In [175]: import itertools

          confusion = confusion_matrix(y_test, pred)
          print(confusion)
          plt.imshow(confusion,cmap=plt.cm.Blues)
          plt.title('confusion matrix')

          plt.colorbar()
          tick_marks = np.arange(2)
          plt.xticks(tick_marks, rotation=45)
          plt.yticks(tick_marks)

          fmt = 'd'
          thresh = confusion.max() / 2
          for i, j in itertools.product(range(confusion.shape[0]), range(confusion.shape[1])):
              plt.text(j, i, format(confusion[i, j], fmt),
                       horizontalalignment="center",
                       color="white" if confusion[i, j] > thresh else "black")


          plt.ylabel('True label')
          plt.xlabel('Predicted label')

          plt.show()
```

```
[[  31  526]
 [  17 2421]]
```



4.5 Feature Importance

4.5.1 Multi-collinearity

```
In [176]: origional=all_word_vectors_matrix

In [177]: origional.shape

Out[177]: (6928, 50)

In [178]: new=np.reshape(np.matrix(np.random.normal(loc=0,scale=0.01,size=all_word_vectors_mat

In [179]: mat=new+origional

In [180]: mat.shape

Out[180]: (6928, 50)

In [184]: per=int(0.7*mat.shape[0])
          c=mat[0:per]
          d=mat[per:]
          label=review.Score[0:per]
          label1=review.Score[per:6928]
```

```
In [185]: from sklearn.preprocessing import StandardScaler
          standardized_data2 = StandardScaler(with_mean=False).fit_transform(c)
          print(standardized_data2.shape)
          standardized_data3 = StandardScaler(with_mean=False).fit_transform(d)
          print(standardized_data3.shape)
```

```
(4849, 50)
(2079, 50)
```

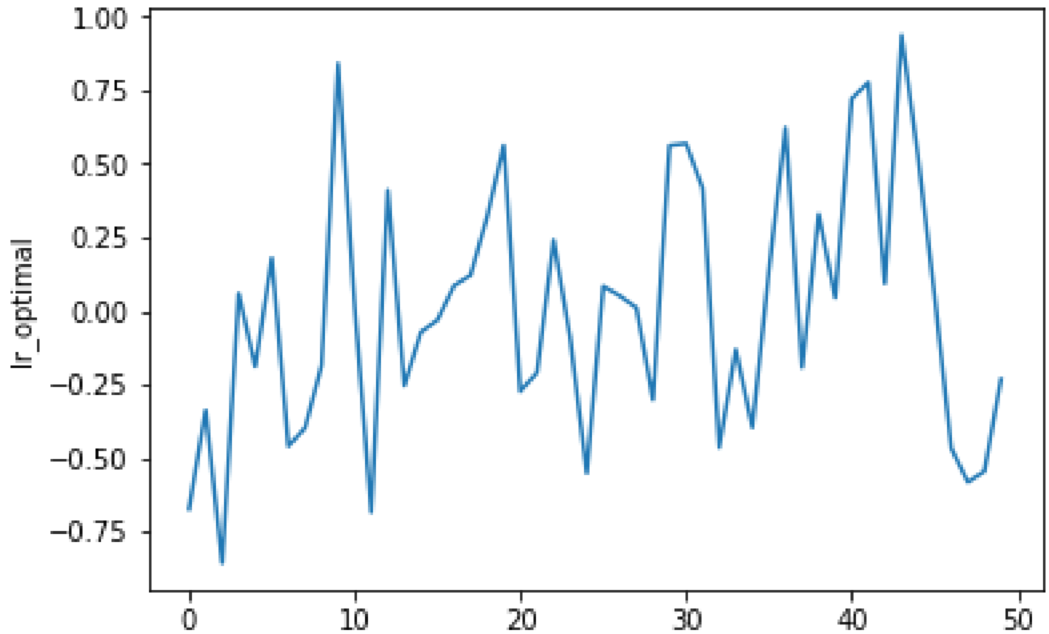
```
In [186]: X_test1, y_test1 = standardized_data3, label1

          # split the train data set into cross validation train and cross validation test
          X_tr1, X_cv1, y_tr1, y_cv1 = cross_validation.train_test_split(standardized_data2, la
```

```
In [230]: lr_optimal1 = LogisticRegression(penalty='l1',C=a)

          # fitting the model
          lr_optimal1.fit(X_tr1, y_tr1)

          # predict the response
          pred = lr_optimal1.predict(X_test1)

          # evaluate accuracy
          acc = accuracy_score(y_test1, pred) * 100
          print('\nThe test accuracy of the knn classifier for  = 240 with l1 regularizer is %
```

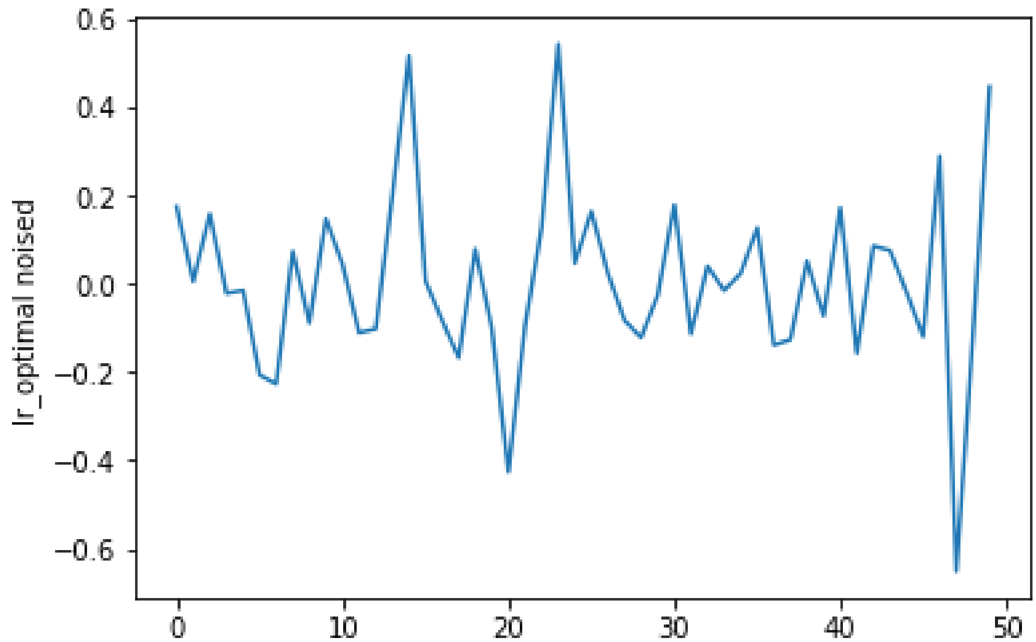
```
The test accuracy of the knn classifier for  = 240 with l1 regularizer is 81.818182%
```

```
In [188]: diff=((lr_optimal.coef_-lr_optimal1.coef_))
```

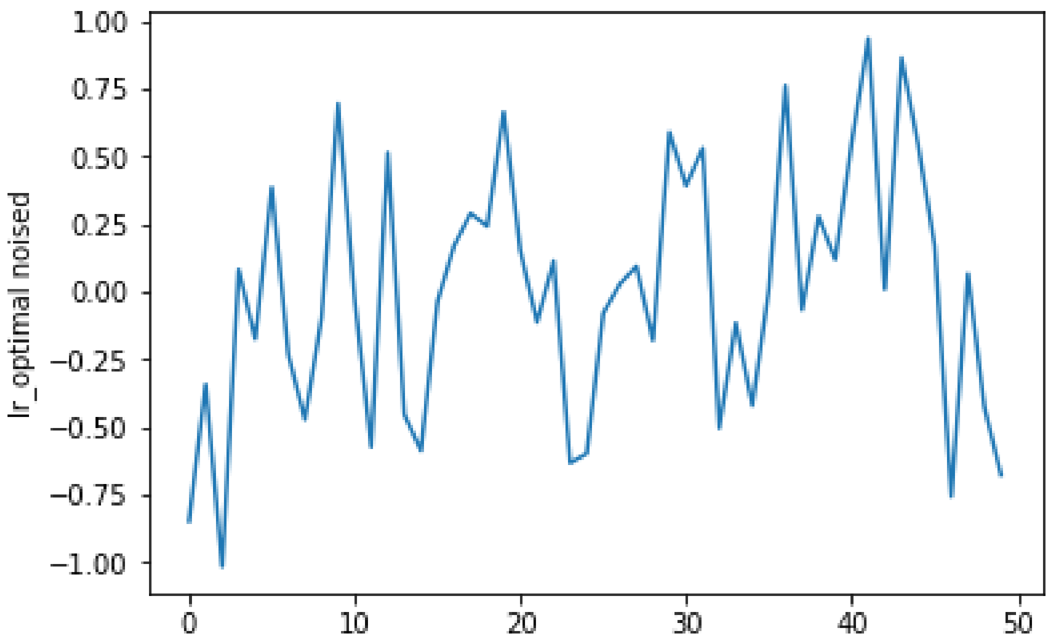
```
In [189]: import matplotlib.pyplot as plt
          plt.plot(lr_optimal.coef_[0])
          plt.ylabel('lr_optimal')
          plt.show()
```



In [190]:
```
plt.plot(lr_optimal1.coef_[0])
plt.ylabel('lr_optimal noised')
plt.show()
```



```
In [191]: plt.plot(diff[0])
          plt.ylabel('lr_optimal noised')
          plt.show()
```



4.5.2 Top 10 features

```
In [192]: dict ={k: v for v, k in enumerate(lr_optimal.coef_[0])}
          imp = [dict[k] for k in sorted(dict.keys())[:10]]

In [193]: imp

Out[193]: [2, 11, 0, 47, 24, 48, 46, 32, 6, 7]
```

5 Conclusion

```
In [231]: import pandas as pd
          list=['model','Grid/RandomSearch','best hyperparam','Train metric','Test metric','spa
          conclu = [('BoW','Grid',341,'87.98', '85.80%', '26.31%'),
                    ('BoW','Random',37,'88.84', '85.34%', '12.31%'),
                          ('tf-idf','Grid',3,'88.07', '84.97%', '2.06%'),('tf-idf','Random',64
                          ('W2Vec','Grid', 1,'81.60','81.76%',' 94%'),('W2Vec','Random', 240,'8

                         ]
          df = pd.DataFrame(conclu, columns=list)
          df
```

Out[231]:

	model	Grid/RandomSearch	best hyperparam	Train metric	Test metric	sparsity
0	BoW	Grid	341	87.98	85.80%	26.31%
1	BoW	Random	37	88.84	85.34%	12.31%
2	tf-idf	Grid	3	88.07	84.97%	2.06%
3	tf-idf	Random	64	83	82.90%	7.44%
4	W2Vec	Grid	1	81.60	81.76%	94%
5	W2Vec	Random	240	81.83	81.86%	100%