Module 4 Assignment Answers

July 27, 2021

1 Module 4 HW

1.1 Filipp Krasovsky, July 25th - 2021

[2]:
```
import os
import re
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
```

[3]:
```
current_folder = os.getcwd()
print("FOLDER:")
print(current_folder);

# data = sklearn.datasets.load_svmlight_files(['labeledBow.feat'])

def load_aclImdb_file(filename):
    ''' filename is id_rating

        returns a dict object, {id, rating, text}
    '''
    id,rating,txt = re.split('[_.]',filename)
    with open(filename,'r',encoding="utf8") as f:
        text_of_file = f.read()

    return {'id':id,'rating':rating,'text':text_of_file}


data_dicts=[]
for cat in ['neg','pos']:

    os.chdir(os.path.join(current_folder,'aclImdb\\train\\',cat))
    print (os.getcwd())
    file_list = os.listdir('.')

    for file,f_num in zip(file_list,range(0,len(file_list))):
        dd = load_aclImdb_file(file)
        dd.update({'cat':cat})
        data_dicts.append(dd)
```

```
        if (f_num%1000==0):
            print('%d file number %s file name' % (f_num,file))
```

```
FOLDER:
C:\Users\13234\Documents\usd_data_sci\504 machine learning\Module 4
C:\Users\13234\Documents\usd_data_sci\504 machine learning\Module
4\aclImdb\train\neg
0 file number 0_3.txt file name
1000 file number 10900_3.txt file name
2000 file number 11800_4.txt file name
3000 file number 1450_3.txt file name
4000 file number 2350_1.txt file name
5000 file number 3250_3.txt file name
6000 file number 4150_2.txt file name
7000 file number 5050_1.txt file name
8000 file number 5951_4.txt file name
9000 file number 6851_4.txt file name
10000 file number 7751_4.txt file name
11000 file number 8651_1.txt file name
12000 file number 9551_1.txt file name
C:\Users\13234\Documents\usd_data_sci\504 machine learning\Module
4\aclImdb\train\pos
0 file number 0_9.txt file name
1000 file number 10900_8.txt file name
2000 file number 11800_8.txt file name
3000 file number 1450_8.txt file name
4000 file number 2350_9.txt file name
5000 file number 3250_9.txt file name
6000 file number 4150_10.txt file name
7000 file number 5050_7.txt file name
8000 file number 5951_10.txt file name
9000 file number 6851_8.txt file name
10000 file number 7751_7.txt file name
11000 file number 8651_9.txt file name
12000 file number 9551_10.txt file name
```

[4]:
```
acl_imdb_data = pd.DataFrame(data_dicts)

# first take on this...
corpus = acl_imdb_data['text']
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

# MemoryError: Unable to allocate 12.5 GiB for an array with shape (76065,␣
 ↪22110) and data type int64

features = vectorizer.get_feature_names()
```

```
word_count_df=pd.DataFrame.sparse.from_spmatrix(X,columns=features,index=corpus.
 ↪index)

from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
lb.fit(acl_imdb_data['cat'])
labels = pd.Series(lb.transform(acl_imdb_data['cat']).reshape(-1))

labels.name = 'LABELS_FOR_CLASSIFICATION'
```

[5]:
```
word_count_df.head() #this is the dataframe we'll be using for the exercise.␣
 ↪this is a very sparse matrix.
```

[5]:
```
   00  000  0000000000001  00001  00015  000s  001  003830  006  007  …  \
0   0    0              0      0      0     0    0       0    0    0  …
1   0    0              0      0      0     0    0       0    0    0  …
2   0    0              0      0      0     0    0       0    0    0  …
3   0    0              0      0      0     0    0       0    0    0  …
4   0    0              0      0      0     0    0       0    0    0  …

   était  état  étc  évery  êxtase  ís  ísnt  østbye  über  üvegtigris
0      0     0    0      0       0   0     0       0     0           0
1      0     0    0      0       0   0     0       0     0           0
2      0     0    0      0       0   0     0       0     0           0
3      0     0    0      0       0   0     0       0     0           0
4      0     0    0      0       0   0     0       0     0           0

[5 rows x 74849 columns]
```

2 Linear Classification Section

Install the package mlextend: http://rasbt.github.io/mlxtend/installation/

Next, you will do a few exercises to visualize the difference between the different linear classifiers.

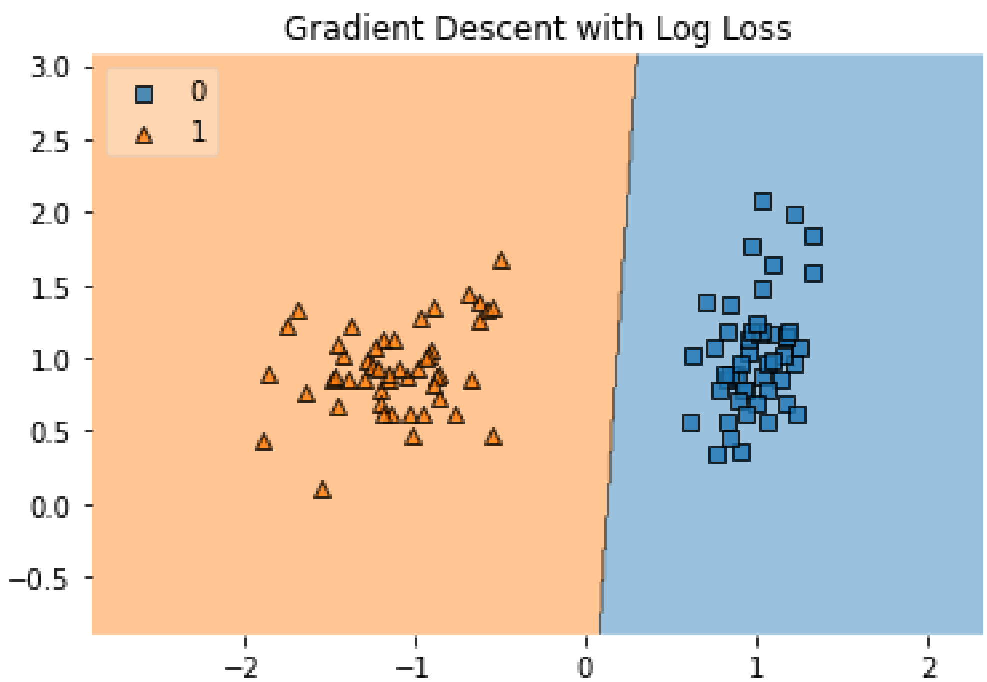
Generate classification data using make_classification from sklearn.datasets:

[6]:
```
import mlxtend as mlxtend
from sklearn.datasets import make_classification
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from mlxtend.plotting import plot_decision_regions
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from mlxtend.evaluate import bootstrap_point632_score
```

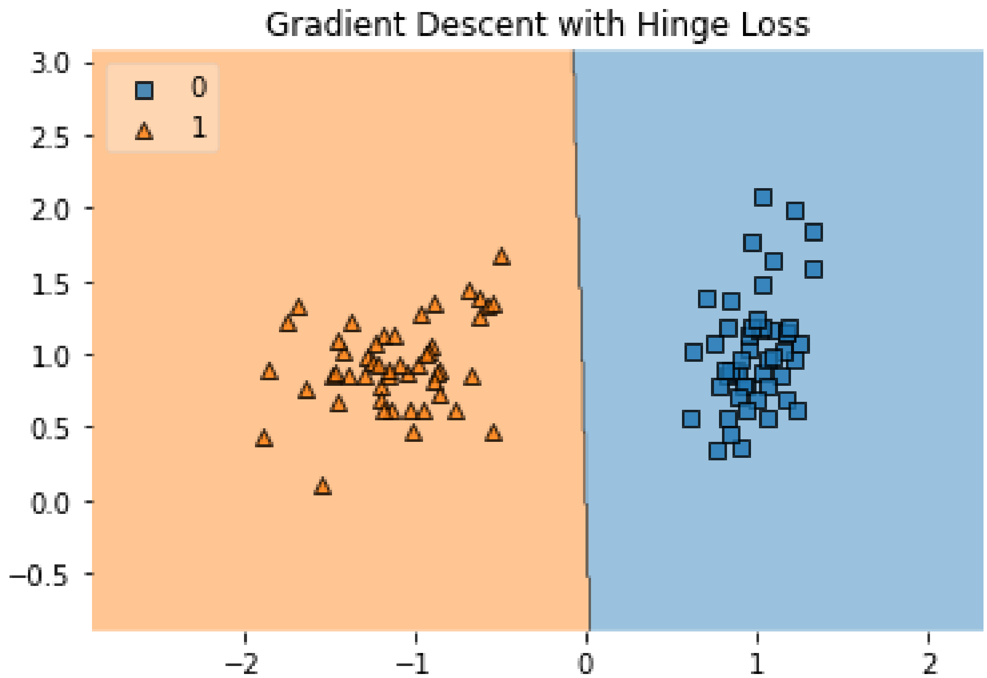
```
import numpy as np
from seaborn import distplot
```

[7]:
```
X  , y = make_classification(n_features=2, n_redundant=0,␣
↪n_informative=2,random_state=1, n_clusters_per_class=1)
```

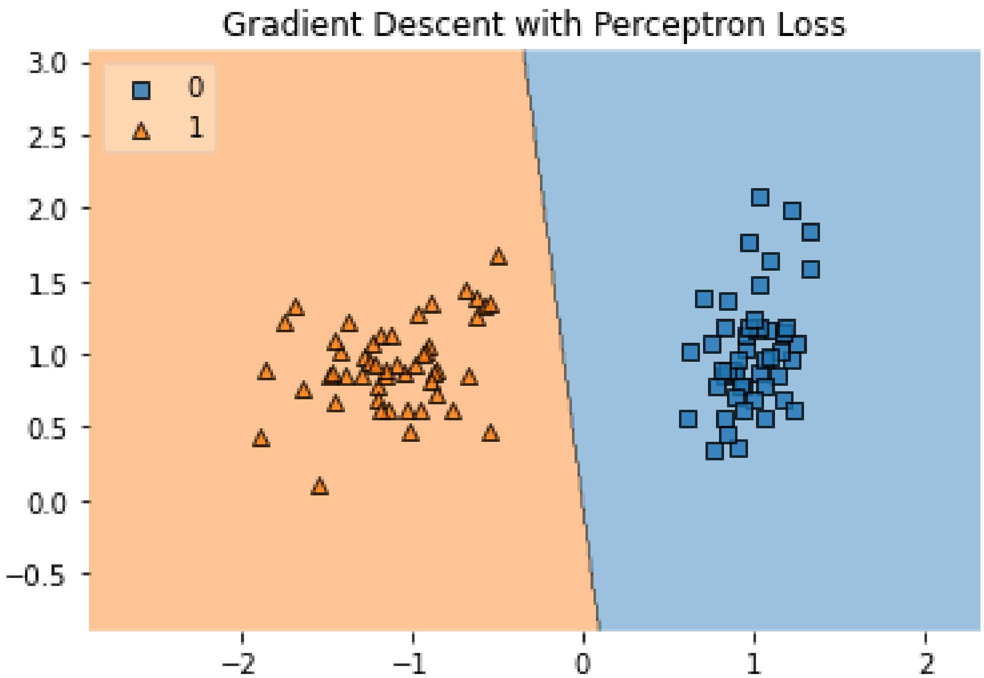
[8]:
```
#Use SGDClassifier to train classifiers using different loss functions: log,␣
↪hinge, and perceptron
log_loss = SGDClassifier(max_iter=1000,loss='log')
log_loss.fit(X,y)
plot_decision_regions(X, y, clf=log_loss, legend=2,)
plt.title("Gradient Descent with Log Loss")
plt.show()
```



[9]:
```
hinge_loss = SGDClassifier(max_iter=1000,loss='hinge')
hinge_loss.fit(X,y)
plot_decision_regions(X, y, clf=hinge_loss, legend=2,)
plt.title("Gradient Descent with Hinge Loss")
plt.show()
```



[10]:
```
perc_loss = SGDClassifier(max_iter=1000,loss='perceptron')
perc_loss.fit(X,y)
plot_decision_regions(X, y, clf=perc_loss, legend=2,)
plt.title("Gradient Descent with Perceptron Loss")
plt.show()
```



Now, create a larger classification dataset. You will use cross_val_score from scikit-learn and compare this to bootstrap_scores from mlextend. Set up the simulated data as follows:

[11]:
```
X, y = make_classification(
    n_samples=10000,
    n_features=20,
    n_redundant=0,
    n_informative=20,
    random_state=1,
    n_clusters_per_class=1
)
```

[12]:
```
#train again

#log-loss
log_loss = SGDClassifier(max_iter=1000,loss='log')
log_loss.fit(X,y)
#hinge loss
hinge_loss = SGDClassifier(max_iter=1000,loss='hinge')
hinge_loss.fit(X,y)
#perceptron
perc_loss = SGDClassifier(max_iter=1000,loss='perceptron')
perc_loss.fit(X,y)
```

[12]:
```
SGDClassifier(loss='perceptron')
```

[13]:
```python
#we can get the accuracies and the bootstrap accuracies as follows:
def getScores(model):
    #get cross validation and bootstrap scores.
    scores = cross_val_score(model, X, y, cv=5)
    bootstrap_scores = bootstrap_point632_score(model, X, y, method='oob')
    return({'cv':scores,'bootstrap':bootstrap_scores})
```

[14]:
```python
#get cv and boot scores for each classifier.
log_loss_scores        = getScores(log_loss)
hinge_loss_scores      = getScores(hinge_loss)
perceptron_loss_scores = getScores(perc_loss)
```

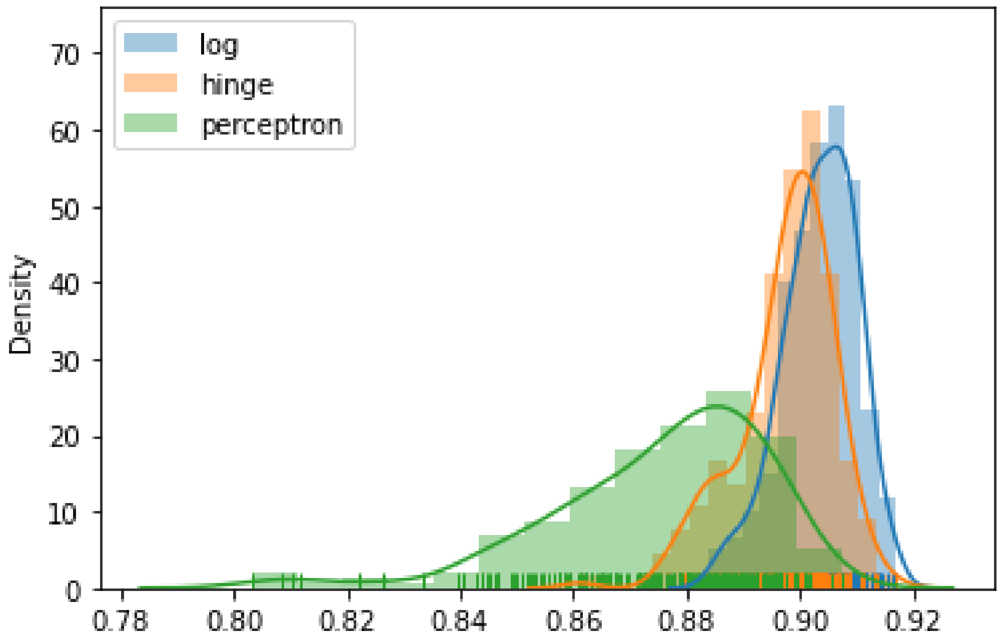
[15]:
```python
#table of average scores
scores_list = []
for i in [log_loss_scores,hinge_loss_scores,perceptron_loss_scores]:
    avg_cv = round(np.mean(i['cv']),3)
    avg_bs = round(np.mean(i['bootstrap']),3)
    scores_list.append([avg_cv,avg_bs])

pd.
 ↪DataFrame(scores_list,columns=["CV","Bootstrap"],index=["Log","Hinge","Perceptron"])
```

[15]:

	CV	Bootstrap
Log	0.902	0.904
Hinge	0.896	0.898
Perceptron	0.873	0.876

[16]:
```python
#plot bootstraps
import warnings
warnings.filterwarnings('ignore')
distplot(log_loss_scores['bootstrap'], rug=True,label="log")
distplot(hinge_loss_scores['bootstrap'], rug=True,label="hinge")
distplot(perceptron_loss_scores['bootstrap'], rug=True,label="perceptron")
plt.legend()
plt.show()
```



Finally, you will look at the importance of setting the regularization parameter. Create a database with only two informative features:

[17]:
```
X, y = make_classification(
    n_samples=1000,
    n_features=2000,
    n_redundant=0,
    n_informative=2,
    random_state=1,
    n_clusters_per_class=1)

alpha_range = [0.0001,0.001,0.01,0.1,1,10,100,1000]
```

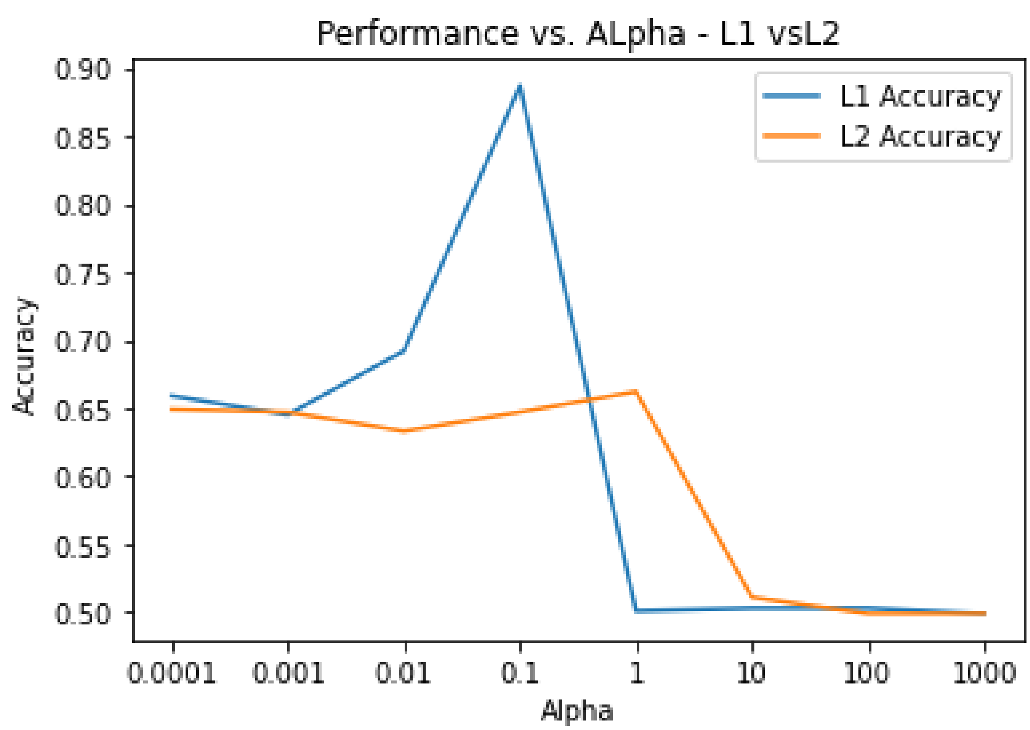
[18]:
```
#experiment with different types of regularization
l1_penalty = []
l2_penalty = []

for alpha in alpha_range:
    model_l1  = SGDClassifier(max_iter=1000,loss='log',alpha=alpha,penalty="l1")
    scores_l1 = np.mean(cross_val_score(model_l1, X, y, cv=5))
    model_l2  = SGDClassifier(max_iter=1000,loss='log',alpha=alpha,penalty="l2")
    scores_l2 = np.mean(cross_val_score(model_l2, X, y, cv=5))
    l1_penalty.append(scores_l1)
    l2_penalty.append(scores_l2)
    print(scores_l1,scores_l2)
```

```
0.659 0.649
```

```
0.6450000000000001 0.647
0.692 0.633
0.8870000000000001 0.647
0.501 0.662
0.503 0.511
0.503 0.499
0.499 0.499
```

[19]:
```python
#plot performance
alpha_range = ["0.0001","0.001","0.01","0.1","1","10","100","1000"]
plt.plot(alpha_range,l1_penalty,label = "L1 Accuracy")
plt.plot(alpha_range,l2_penalty,label = "L2 Accuracy")
plt.legend()
plt.xlabel("Alpha")
plt.ylabel("Accuracy")
plt.title("Performance vs. ALpha - L1 vsL2")
plt.show()
```



3 Large Scale Linear Classification

we will be working with the movie review data for this exercise.

[20]:
```python
word_count_df.head()
```

[20]:
```
   00  000  0000000000001  00001  00015  000s  001  003830  006  007  …  \
0   0    0              0      0      0     0    0       0    0    0  …
1   0    0              0      0      0     0    0       0    0    0  …
2   0    0              0      0      0     0    0       0    0    0  …
3   0    0              0      0      0     0    0       0    0    0  …
4   0    0              0      0      0     0    0       0    0    0  …

   était  état  etc  évery  êxtase  ís  ísnt  østbye  über  üvegtigris
0      0     0    0      0       0   0     0       0     0           0
1      0     0    0      0       0   0     0       0     0           0
2      0     0    0      0       0   0     0       0     0           0
3      0     0    0      0       0   0     0       0     0           0
4      0     0    0      0       0   0     0       0     0           0

[5 rows x 74849 columns]
```

[22]:
```python
#experiment with different types of regularization for movie review data.
#max iterations reduced to speed up computation,
#commented out to prevent pdf conversion from taking too long.

# l1_penalty = []
# l2_penalty = []
# alpha_range = [0.0001,0.001,0.01,0.1,1,10,100,1000]
# cores = 2

# for alpha in alpha_range:

#     print ("running job for alpha:",alpha," num cores:",cores)
#     model_l1  =␣
↪SGDClassifier(max_iter=500,loss='log',alpha=alpha,penalty="l1",n_jobs =␣
↪cores)
#     scores_l1 = np.mean(cross_val_score(model_l1, word_count_df, labels,␣
↪cv=5))

#     model_l2  =␣
↪SGDClassifier(max_iter=500,loss='log',alpha=alpha,penalty="l2",n_jobs =␣
↪cores)
#     scores_l2 = np.mean(cross_val_score(model_l2, word_count_df, labels,␣
↪cv=5))

#     l1_penalty.append(scores_l1)
#     l2_penalty.append(scores_l2)

#     print(scores_l1,scores_l2)
```

```
running job for alpha: 0.0001  num cores: 2
0.8402000000000001 0.8343599999999999
running job for alpha: 0.001  num cores: 2
```

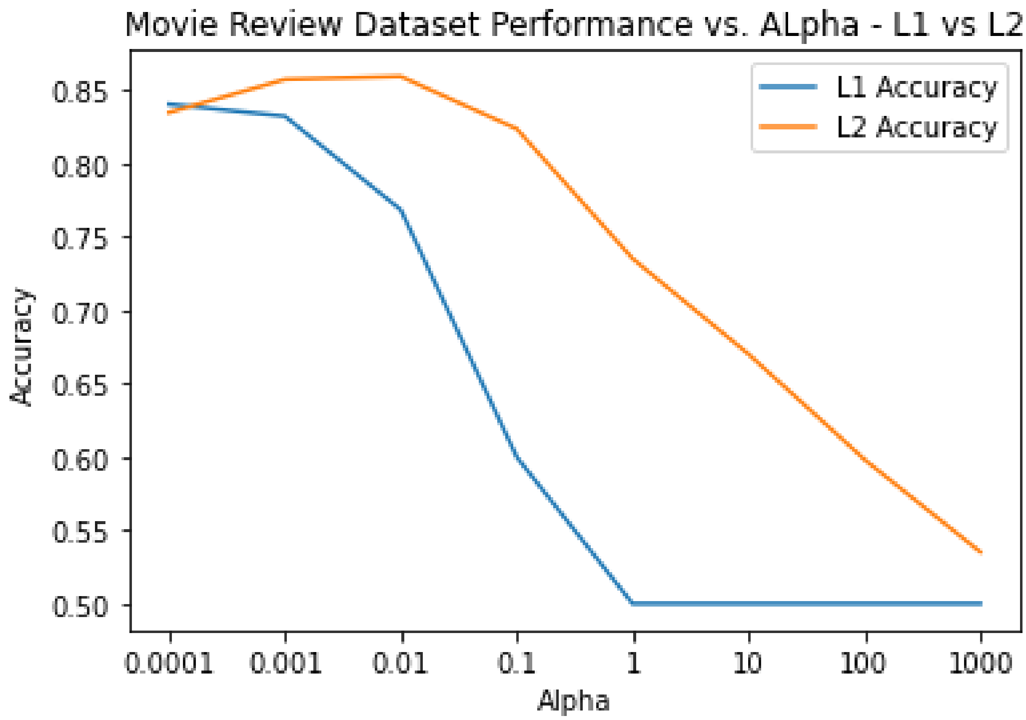
```
0.8320000000000001 0.8570399999999999
running job for alpha: 0.01  num cores: 2
0.7682 0.8590800000000002
running job for alpha: 0.1  num cores: 2
0.6 0.8234400000000001
running job for alpha: 1  num cores: 2
0.5 0.73528
running job for alpha: 10  num cores: 2
0.5 0.67096
running job for alpha: 100  num cores: 2
0.5 0.5982000000000001
running job for alpha: 1000  num cores: 2
0.5 0.5352399999999999
```

[23]:

```
#scores hard-coded to avoid processing time during pdf output
l1_penalty = [0.8402,0.832,0.7682, 0.6,0.5,0.5,0.5,0.5]
l2_penalty = [0.8345,0.857,0.859, 0.82344,0.735,0.67,0.5982,0.5352]
alpha_range = [0.0001,0.001,0.01,0.1,1,10,100,1000]
```

[24]:

```
#plot performance for the text data.
alpha_range = ["0.0001","0.001","0.01","0.1","1","10","100","1000"]
plt.plot(alpha_range,l1_penalty,label = "L1 Accuracy")
plt.plot(alpha_range,l2_penalty,label = "L2 Accuracy")
plt.legend()
plt.xlabel("Alpha")
plt.ylabel("Accuracy")
plt.title("Movie Review Dataset Performance vs. ALpha - L1 vs L2")
plt.show()
```


Movie Review Dataset Performance vs. ALpha - L1 vs L2
L1 Accuracy
L2 Accuracy
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.0001
0.001
0.01
0.1
1
10
100
1000
Alpha