Active Learning Cycle

November 3, 2018

This Notebook demonstrates the process of Active Learning using features from the Stanford movies review database. It is divided into two short parts: 1) background about active learning and the purpose of the active learning service 2) technical demonstration of the iterative Active Learning Service using Scikit-learn.

1 Background

Active Learning is a process that relies on an expert's feedback to help label samples. The benefit of an Active Learning Service is to learn a model of the concept (in this case sentiment) in a manner that minimizes the need for expert resources.

The Active Learning Service accomplishes this through an iterative process that trains on the labeled features and then predicts on the unlabeled samples' features. These predictions can then be evaluated in a manner to suggest the next desirable review for the expert to label.

For example, ordering based on distance from the classification boundary allows prioritization of the sample that may be questionable. Once labeled, the iterative process repeats, the labeled features are used to train the model, and predictions are evaluated to provide the next sample(s) for labeling.

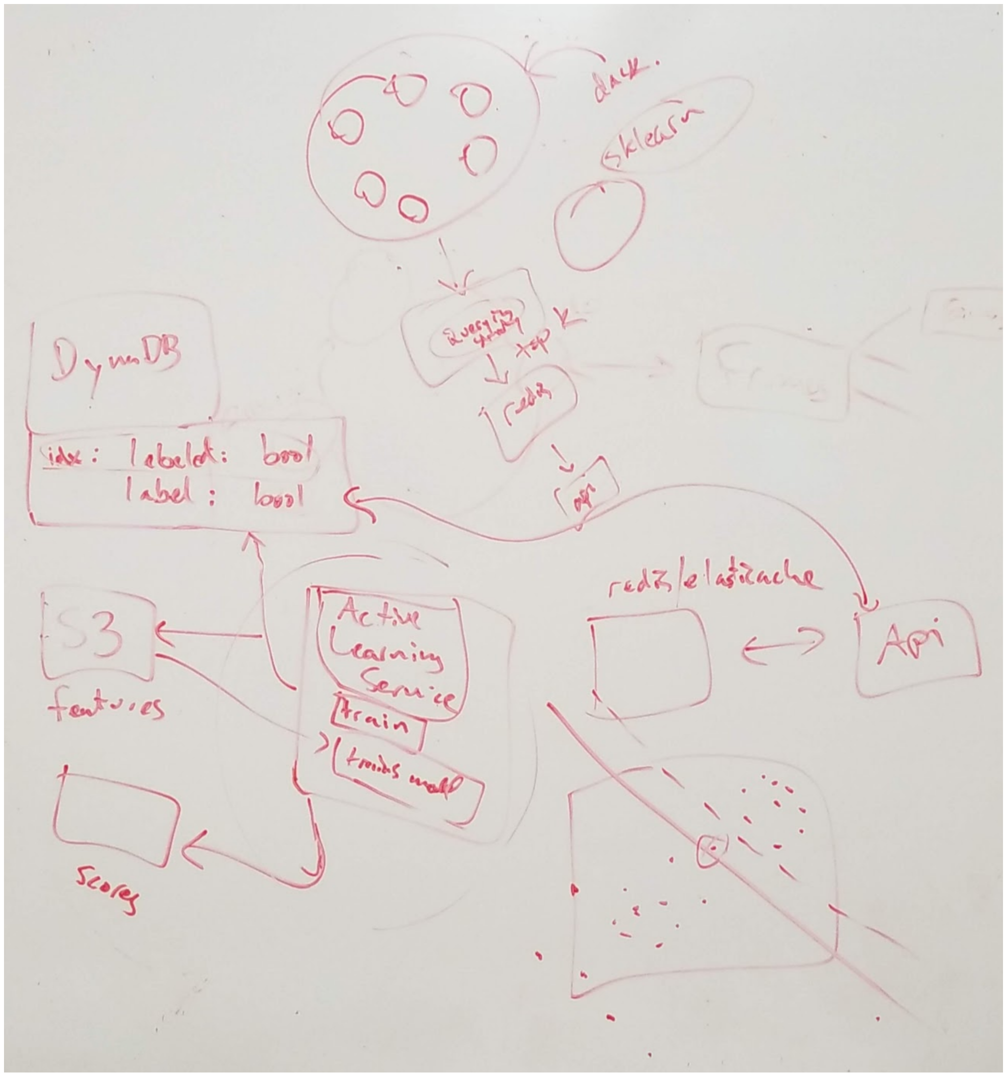
In the case of concept sentiment, the labels positive and negative could be captured as a boolean value. For those not familiar with sentiment prediction, here is a sample from the movie review dataset that has a positive sentiment label:

[train/pos/1999_9.txt]

Oliver! the musical is a favorite of mine. The music, the characters, the story. It all just seems perfect. In this rendition of the timeless classic novel turned stage musical, director Carol Reed brings the Broadway hit to life on the movie screen.The transition from musical to movie musical is not an easy one. You have to have the right voices, the right set, the right script, and the right play. All signs point to yes for this play. It almost appears that it was written for the screen!Our story takes place in jolly old England where a boy named Oliver manages to work his way out of the orphanage. He winds his way through the country to London where he meets up with a group of juvenile delinquents, headed by Dodger, the smart talking, quick handed pick-pocket. The leader of this gang is named Fagin, an older fellow who sells all the stolen goods.But all is not well in London town when Bill Sykes played by Oliver Reed and his loving girlfriend Nancy get tangled up with Oliver, Fagin and his young troops, and the law. What ensues is a marvelous tale of love, affection, and great musical numbers.Whether or not you like musicals or not, one listen to these tunes and you will be humming them all day long. Oliver! is a triumph on and off the stage and is a timeless work of art.

2 Technical Demonstration

```
In [3]: # get information about the features, these are precomputed and cached locally
        !ls /Users/username/Downloads/features/ | wc -l
```


dask.
sklearn
Querying strategy
top k
redis
api
DynmDB
idx: labeled: bool
label: bool
redis/elasticache
Api
S3
features
Active Learning Service
train
trains model
scores

caption

```
  100000
```

```
In [10]: # get information about a review's features
         import numpy as np
         dat = np.fromfile('/Users/username/Downloads/features/train_pos_1999_9.npy',dtype=np.
         print(len(dat))
         print(str(dat))
```

```
300
[ 0.02389269  0.03208643  0.00130268  0.04201266 -0.03200313 -0.00951154
  0.01946418 -0.02488746  0.01856465  0.04633249 -0.01337551 -0.02968472
 -0.00741054  0.00332867 -0.02204074  0.02735291  0.02232926  0.04266064
 -0.01007411 -0.00491554  0.00753427  0.01638046  0.0167506   0.00097866
  0.01007058 -0.00586906 -0.04155467  0.0172828  -0.01259738  0.01598104
 -0.00583364 -0.00481078 -0.02290771 -0.00967428  0.00477918  0.01339398
  0.00203974 -0.00345512  0.01239723  0.01354235  0.0033645  -0.00636118
  0.00804293  0.00258544  0.01370893 -0.00540722 -0.02329065 -0.0031036
  0.03814979 -0.00680215 -0.02033133  0.01307743  0.01066388 -0.01918704
  0.01334588  0.02784437 -0.00047107 -0.02694349 -0.00936304 -0.02427477
 -0.01462668  0.03339586 -0.01372353 -0.0217119  -0.01895672 -0.00555888
 -0.01645579  0.03707255 -0.00773336  0.02747312  0.0178586  -0.0108341
  0.03610139  0.00416935 -0.04431386 -0.02259126  0.04469617  0.03034797
  0.01368262  0.04607592  0.0065044  -0.05266588  0.01527258 -0.00072507
 -0.00498095 -0.0508548  -0.03305129  0.02531077 -0.00628972  0.01796868
 -0.01334467  0.00348812 -0.00674038 -0.03906911 -0.01915325 -0.00301781
  0.00945968  0.00803841 -0.01605559 -0.00620032 -0.01822795 -0.02726782
  0.00907856  0.0013698  -0.02069484  0.00273439 -0.01732865 -0.01682537
  0.00333772 -0.02669027 -0.00283327  0.01146817 -0.01189662 -0.00231617
  0.0221677   0.02624358 -0.00512576 -0.00462128  0.02872706  0.0272086
 -0.02384911 -0.00996297 -0.0192763   0.022771   -0.0210983  -0.02340583
 -0.03324932 -0.02609345 -0.00423159  0.00436864 -0.01576943 -0.03324709
 -0.03147617 -0.00551402 -0.01196537 -0.02144691 -0.00548807 -0.01150319
 -0.00607275  0.01513718  0.01554144 -0.00842663  0.0140314   0.00560301
  0.00572821 -0.00605641 -0.00472237 -0.01955026 -0.0108583   0.0037239
  0.01982815 -0.00590357 -0.01470358  0.01876133 -0.02340551 -0.00393199
 -0.00188099 -0.03407123 -0.01271148 -0.03685787 -0.00121099  0.01571736
  0.00812683  0.03134582  0.014875   -0.02215409  0.01225888 -0.01547595
  0.00113172  0.0013605  -0.04972023 -0.03289973 -0.02160489 -0.04909409
 -0.01563334 -0.00349573  0.00863243 -0.02221062  0.00297942 -0.01910936
 -0.0438404  -0.00916625  0.01039164 -0.01442079 -0.01801936 -0.01493038
 -0.02030629  0.00807121  0.03536198  0.01017396  0.01314361  0.00576683
  0.01219863  0.00717768 -0.02816735  0.01117668 -0.01144364  0.00186007
 -0.02404465 -0.03957703  0.00386397 -0.00992133  0.00460757  0.02132179
 -0.01001743  0.01432652 -0.034856    0.00194338 -0.00228568  0.00340654
 -0.00553177  0.00873545 -0.00705064 -0.0009634  -0.03094816  0.00124642
  0.02176346  0.00481479 -0.02645016  0.00217478  0.01795107  0.00690572
 -0.013875    -0.00997209  0.01239885 -0.0023845   0.00367149  0.01037739
  0.00573493 -0.01269314  0.02223626 -0.03066641  0.0115904   0.00362307
  0.01917978 -0.00709836  0.00679137 -0.01963491  0.00842112  0.00476251
  0.0091506   0.0099989   0.001147   -0.04496251 -0.00119214  0.00722916
  0.01717613  0.02357702  0.01986211 -0.01729477  0.01567401  0.01723851
  0.02311447  0.02820389  0.04410781 -0.00246761  0.02893915  0.009391
 -0.03390107 -0.01812316 -0.00081339  0.01522141 -0.02584924  0.01803888
  0.0188424   0.04982048 -0.01466913 -0.00820022 -0.02228144 -0.00505447
  0.00269071  0.0381523   0.02761522  0.0235537   0.02016649  0.00288224
 -0.01669673 -0.03879717 -0.00420486 -0.02278801 -0.00508184 -0.01301287
 -0.0010654   0.04225359  0.03079057 -0.00169411 -0.04902793 -0.01308202
  0.01655679 -0.00260251 -0.03669704  0.02413035 -0.03091129 -0.00201081
 -0.0122902  -0.01193171  0.00155638 -0.00645472  0.011218   -0.01024845]
```

We see that the 300 features listed above are float values. These particular features represent the positive movie review for Oliver!.

Lets now cover how these 300 values were generated. Each word in the document is used to find its features, and then these features are averaged to produce the above document-level feature. Services like Pymagnitude are used to create the 300 floats for each word using the query method. Then, for each document, these are averaged.

```
In [ ]: #!pip install pymagnitude
        from pymagnitude import *
        import glob

        # set up local path
        local_path = r'/Users/username/Downloads/aclImdb'
        pretrained_magnitude = r'/Users/username/Downloads/GoogleNews-vectors-negative300.magn

        vectors = Magnitude(pretrained_magnitude)

        def get_cleaned_string(in_string):
            safechars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 -.'
            cleaned_list = []
            for s in in_string:
                if s in safechars:
                    cleaned_list.append(s)
                else:
                    cleaned_list.append(' ')
            return ''.join(cleaned_list)

        for filename in glob.iglob(local_path+'/**/*.txt', recursive=True):
            with open(filename) as f:
                data = f.read().replace('\n', '')
                cleaned_data = get_cleaned_string(data)
                vecs = vectors.query(cleaned_data.split(' '))r
                avg_vec = np.mean(vecs, axis=(0))
                f_name = filename.replace(local_path + '/','')
                avg_vec.tofile(r'/Users/username/Downloads/features/'+f_name.replace(r'/','_')
```

With a technical understanding of how the document's feature vector is made, let's move forward to see how the iterative Active Learning Service could be implemented using Scikit-learn.

```
In [2]: # lets get a list of each positive and negative labeled features
        import glob

        pos_list = list()
        neg_list = list()
        for filename in glob.iglob(r'/Users/username/Downloads/features'+'/*.npy', recursive=Fa
            if '_pos_' in filename:
                pos_list.append(filename.replace('/Users/username/Downloads/features/',''))
            if '_neg_' in filename:
                neg_list.append(filename.replace('/Users/username/Downloads/features/',''))
```

```
In [3]: # now, we assume a uniform selection and labeling by randomly picking 500 labeled pos
        pos_list[500]
```

```
Out[3]: 'train_pos_5318_7.npy'
```

```
In [4]: import random
        labeled_pos = random.sample(pos_list, 500)
        labeled_neg = random.sample(neg_list, 500)
```

```
In [5]: # lets remove these from the pool of simulated unlabeled values
        pos_list = list(set(pos_list) - set(labeled_pos))
        neg_list = list(set(neg_list) - set(labeled_neg))
```

```
In [17]: # now generate the training and target data
         import numpy as np

         np_arr_list = list()
         for file in (labeled_pos + labeled_neg):
             dat = np.fromfile('/Users/username/Downloads/features/'+file,dtype=np.float32,cou
             #print(len(dat))
             np_arr_list.append(dat)

         X_train = np.vstack(np_arr_list)
```

```
In [19]: print(type(X_train))
         print(len(X_train))
```

```
<class 'numpy.ndarray'>
1000
```

```
In [39]: # now lets build the target, pos = 1 and neg = 0
         target = np.concatenate([np.zeros(500),np.ones(500)])
         print(target.shape)
         print(target)
```

```
(1000,)
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
In [40]: # lets see some of these labels
         print(target[0])
         print(target[498])
```

```
        print(target[499])
        print(target[500])
        print(target[501])
        print(target[999])
```

```
0.0
0.0
0.0
1.0
1.0
1.0
```

```
In [59]: from sklearn.linear_model import LogisticRegression
         clf = LogisticRegression()
         model = clf.fit(X_train, target)
```

```
In [43]: # lets predict the output
         import numpy as np

         np_train_list = list()
         for file in (pos_list + neg_list):
             dat = np.fromfile('/Users/username/Downloads/features/'+file,dtype=np.float32,cou
             np_train_list.append(dat)
         X_test = np.vstack(np_train_list)
```

Let's take a look at how many 'unlabeled' samples do we have.

```
In [50]: print(len(pos_list))
         print(len(neg_list))
         print(len(X_test))
```

```
24500
24500
49000
```

```
In [68]: # now predict the 'unknown values'
         predictions = model.predict_proba(X_test)
```

```
In [69]: print(type(predictions))
```

```
<class 'numpy.ndarray'>
```

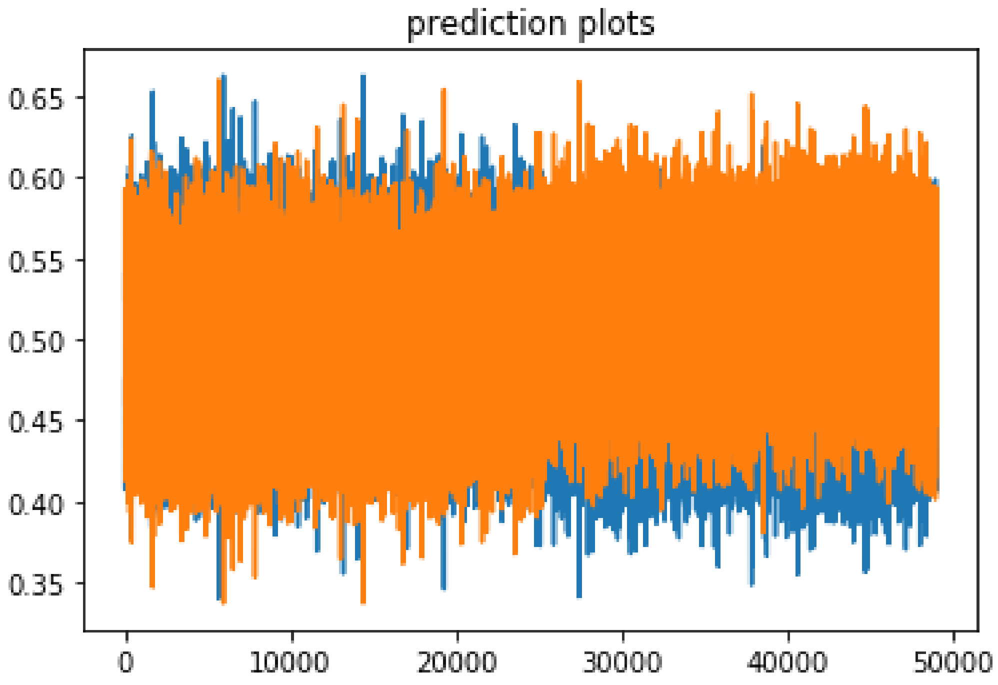
```
In [70]: np.unique(predictions)
```

```
Out[70]: array([0.33727295, 0.33994489, 0.34103801, ..., 0.65896199, 0.66005511,
                0.66272705])
```

```
In [71]: %matplotlib inline

         import matplotlib
         import numpy as np
         import matplotlib.pyplot as plt

         plt.plot(predictions)
         plt.title('prediction plots')
         plt.show()
```



Now we evaluate the prediction results and suggest the next label. The simplest implementation is to get the index of the value closest to the boundary.

```
In [76]: def find_nearest(array, value):
             array = np.asarray(array)
             idx = (np.abs(array - value)).argmin()
             return idx

In [77]: next_to_label_idx = find_nearest(predictions,0.5)

In [78]: print(next_to_label_idx)

33546
```

Lets look at this next-to-label file.

```
In [79]: (pos_list + neg_list)[next_to_label_idx]
```

```
Out[79]: 'train_neg_1016_4.npy'
```

The movie itself is not too bad; many comments have pointed out the obvious flaws of the script, but it is watchable. What really gives me the creeps though is that people like Justin Timberlake even get cast for movies, and on top of that for movies like this one. I have to admit I had never heard the man's name before watching this, but the very instant he appeared I was just plain annoyed. The voice is crap, the face is a bad rip-off of Legolas, the posture is horrible, and he cannot even properly coordinate all three of them. Said to say, I was delighted when he got jumped after leaving the disco, because I was hoping from then on it would be Morgan Freeman and Kevin Spacey only. Too bad I was wrong. These two and also LL Cool J give a very decent performance, and they are the main reason I give this a 4. I see many upcoming movies with the little Timberlake cast... and cannot believe it.

So what do you think? Positive or negative? Is it apparent why the model may have a harder time classifying this review?

The Active Learning Service repeats by (re)training on a slightly larger labeled set of training data. The retrained model predicts on the somewhat smaller collection of unlabeled data. Then the service returns the next best-unlabeled review.

```
In [ ]:
```