CSC 730 Assignment 08
Active Learning

Mangesh Sakordekar

Overview

The aim of this assignment is to:

- Get the MNIST dataset.
- Write your own version of an active learning classifier.
- Run your code on the dataset and determine accuracy and a confusion matrix.
- Do this sequentially over a number of iterations sufficiently large to see performance flatten out, and plot accuracy vs. number of iterations.

Importing Dataset

MNIST data was imported and decimated down to 6000 entries. The data was then scaled and split into training and test datasets. The training set contains 5000 entries whereas the test set has 1000 entries.

Query Strategy

Apart from the base strategy of selecting points at random, I tried 4 different strategies.

Random / Base Strategy

This strategy randomly shuffles the indices of the labels array and pops the element at front for each query.

Strategy 1 - Using Isolation Forest

Isolation forest was used to sort the points by their anomaly scores and then the list was sampled as a binary search tree to ensure points from different regions from the center of the distribution would be queried.

Strategy 2 - OPTICS Clustering

This strategy compresses the data using tSNE and run OPTICS clustering algorithm on it. Points from each cluster are given sequentially. I expected this strategy to not perform that well as tSNE is mainly used for plotting and not for analysis but I wanted to see if the separation the classes what we can see in tSNE plot could be translated into computer knowledge.

Strategy 3 - Lowest Confidence Point

This strategy queries the point which has the lowest probability for the class that it has been classified. This helps the classifier to learn about the region which it is the least confident with.

Strategy 4 - GMM Clustering

This strategy runs GMM clustering algorithm on the data. Points from each cluster are given sequentially. By doing so I wanted to provide the model with points from different regions of the data-space.

Active Learner Implementation

Active Learner was implemented as a class which takes in the training data, testing data, testing labels, a Strategy instance, a Classifier instance and an Oracle instance. Strategy, Classifier and Oracle are classes implemented separately.

The Oracle class takes in the labels of the training data set and returns the label of the index queried. I made this into a class implementation as it would be easier to extend it for human interaction. The Classifier class contains a MLPClassifier and functions to train, predict and get scores from the classifier.

The Active Learner's implementation follows the diagram provided in the slides. One change from the diagram is that the first point also involves the utility strategy instead of being random. First it checks the stopping criteria, then gets the index of the data-point with the highest utility value from the strategy class. Next, it calls the oracle for the label of that data-point and adds the data-point and the label to the learning data set. Then it trains the classifier using the learning data set and gets the predicted labels for the testing data. Next it generates the confusion matrix comparing the predicted labels with the actual labels. It calculates the accuracy of the model and stores the accuracy and the confusion matrix of that iteration into an array.

```
def learn(self, threshold = 0.6):

    X_L = np.empty((0,self._X.shape[1]))
    y_L = np.empty((0,))
    accuracies = []
    cms =  []
    accuracy = 0.0
    query_count = 0

    while accuracy < threshold and query_count < self._max_queries:

        query_count += 1

        indx = self._strategy.get_query()
        X_L = np.concatenate((X_L, self._X[indx,:].reshape((1, self._X.shape[1]))))
        y_L = np.concatenate((y_L, self._oracle.get_label(indx).reshape((1,))))

        self._classifier.train(X_L, y_L)
        y_pred = self._classifier.predict(self._X_test)

        accuracy, cm = self._metrics.get_accuracy(self._y_test, y_pred)
        accuracies.append(accuracy)
        cms.append(cm)


    return accuracies, cms
```

Figure 1: Active Learner Implementation

Confusion Matrices

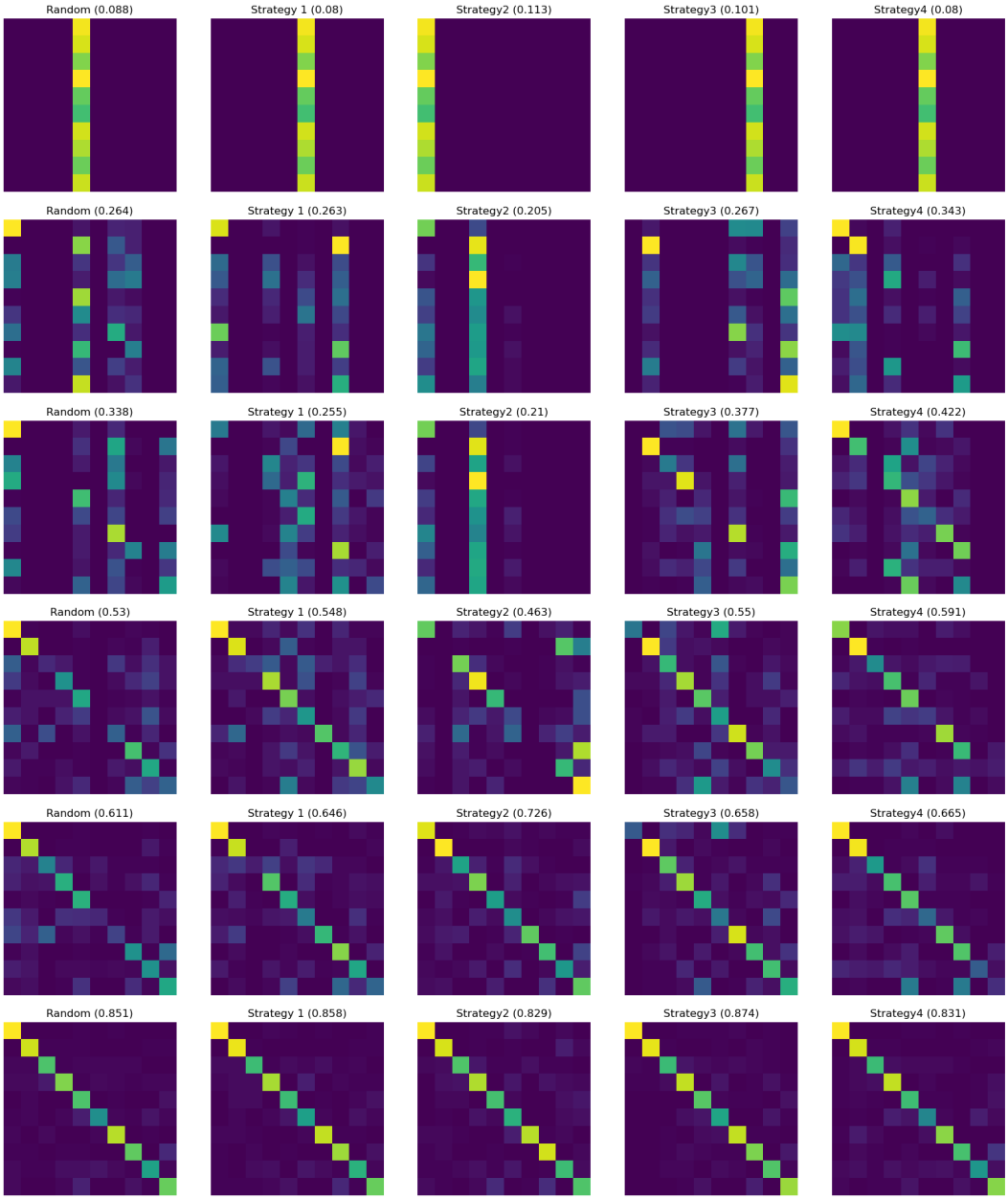

Figure 2: Confusion Matrices for Query (Top to Bottom) 1, 5, 10, 40, 75, 400)

Here we can see the confusion matrices at queries 1, 5, 10, 40, 75 and 400 from top to bottom. We can see, some strategies end up guessing points from the same class in the first

few queries. Also we notice that at different iterations, different strategies predict better. For example at iteration 40 the model with strategy 2 is struggling whereas strategy 4 has the highest accuracy, but at iteration 75 Strategy 2 model outperforms everyone. Overall the Strategy 3, which is also the simplest, has the highest accuracy. Thus, we can refer to Occam's razor and choose the simplest model here.

Accuracy vs Number of Queries Plot

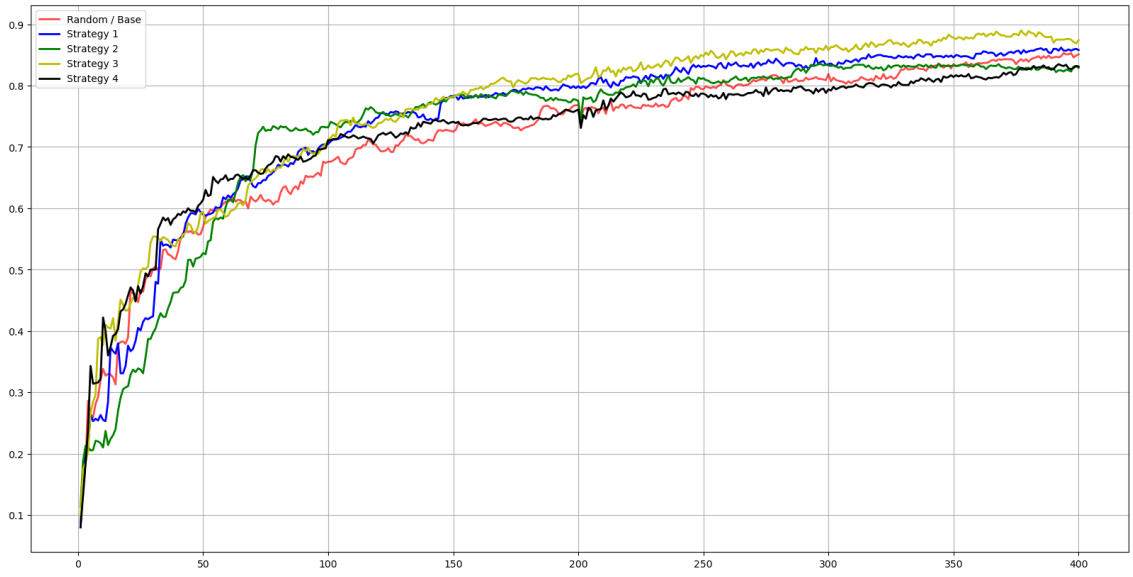

Figure 3: Accuracy vs Number of Queries

Here we can confirm our observations from the confusion matrices that at different queries, different strategies work better. A surprising this is how good the random model works. We can also see strategy 2 struggling at the beginning as it ends up querying from the same class. Overall, all the models end up with predicting with an accuracy of more than 80%.

Conclusion

An active learning model was successfully implemented and tested with different strategies. The model uses a neural net based classifier thus the training times are high. All the models end up predicting with an accuracy above 80% with 400 queries. Apart from the random model, all the models reach an accuracy of 70% by the 100th iteration. The increase in accuracy starts plateauing as more and more points are queried.