Active Learning

- Manav Kaushik

Introduction

The major motivation behind active learning is that if a learning algorithm can choose the training data it wants to learn from, it can perform better than traditional methods (or passive learning) with substantially less data for training. Formally, Active learning is a special kind of learning where the algorithm can interactively query an oracle to label data points based on the algorithm's preferences.

One of the highly time-consuming tasks in passive learning/ traditional learning is collecting labelled data for training. Several times, there are numerous factors that hamper gathering large amounts of labelled data such high cost and time investment. Some examples where labeled data is hard to come by:

- Speech Recognition
- Document Classification
- Image & Video annotation

In active learning, there are typically three scenarios or settings in which the learner will query the labels of instances. The three main scenarios that have been considered in the literature are:

- **Membership Query Synthesis**: this is a big term which simply means that the learner generated/constructs an instance (from some underlying natural distribution). For example, if the data is pictures of digits, the learner would create an image that is similar to a digit (it might be rotated or some piece of the digit excluded) and this created image is sent to the oracle to label.

Fig. Membership Query Synthesis

- **Stream-Based Selective Sampling**: in this setting, you make the assumption that getting an unlabelled instance is free. Based on this assumption, you then select each unlabelled instance one at a time and allow the learner to determine whether it wants to query the label of the instance or reject it based on its informativeness. To determine informativeness of the the instance, you use a query strategy (see next section).

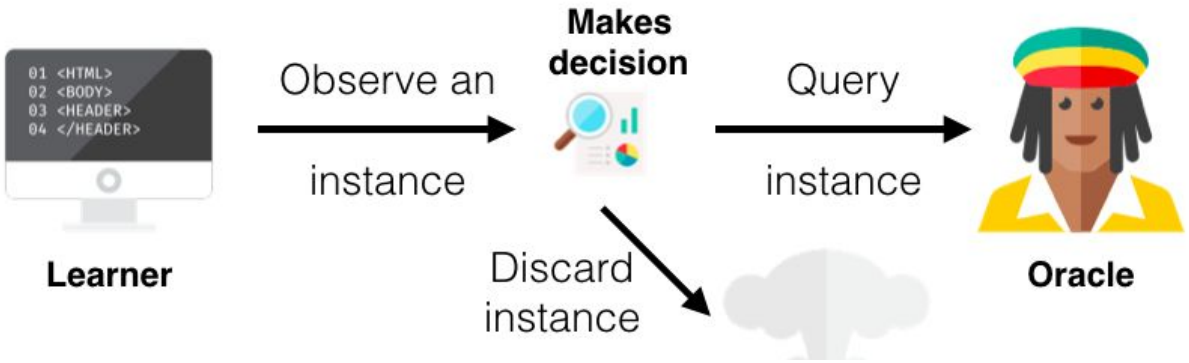


Fig. Stream Based Scenario

- **Pool-Based sampling**: this setting assumes that there is a large pool of unlabelled data, as with the stream-based selective sampling. Instances are then drawn from the pool according to some informativeness measure. This measure is applied to all instances in the pool (or some subset if the pool is very large) and then the most informative instance(s) are selected. This is the most common scenario in the active learning community.

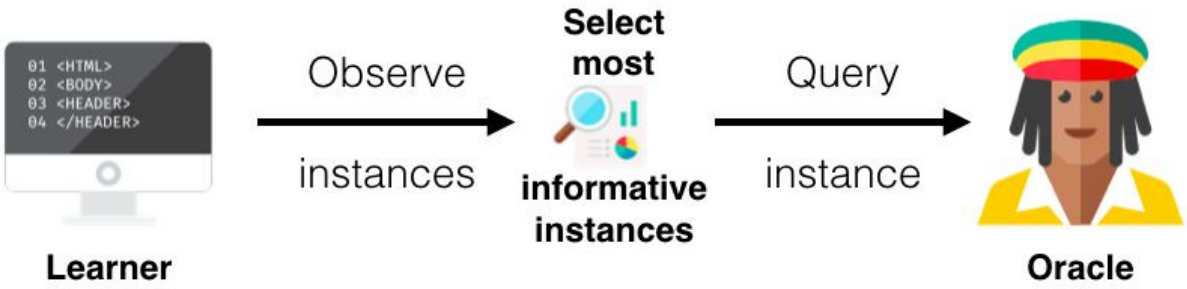


Fig. Pool Based Scenario

Now, with this background of Active Learning in mind, we shall move to solving the proposed problem of the assignment and looking into the other concepts involved

Problem Setting & Dataset

Q: Convert a supervised learning problem (multiclass classification – having more than 3 classes) into an Active Learning problem by randomly removing class labels of data points (retain only 10% of labelled points). The removed labelled points will work as a human oracle!!

For Q1, we shall be using an inbuilt dataset of sklearn library: **Digits Dataset**

- **Problem:** Classification of digits (0 to 9).

- **Dataset Details:**
  - Total number of Datapoints = **1797**
  - Dimensionality/ No. of features = **64**
  - Shape of the Dataset = **(1797, 65)**
  - Shape of the Feature array (X) = **(1797, 64)**
  - Shape of the Label array (Y) = **(1797,1)**

Machine treats the dataset as a multidimensional array, but if we plot this array, following digits can be visualized:

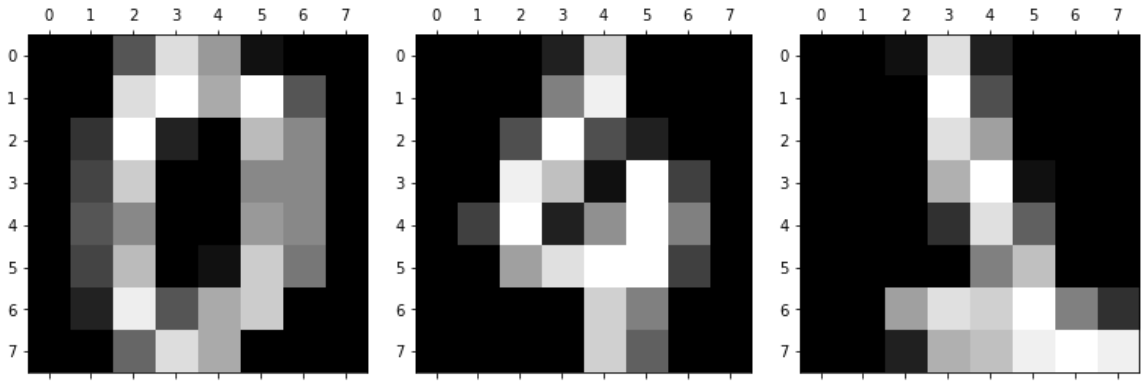


Fig. Sample Data Points Visualized

- **Data Split for Active Learning:**

For our purpose, we have split the data into 3 pools: **Train, Test & Unlabelled** in the ratio of **1:2:7** respectively.

Pool	Percentage of Split	Number of Data Points
Total Dataset	100%	1797
Train Pool	10%	179
Test Pool	20%	324
Unlabel Pool	70%	1294

- **Code Snippet for the split:**

```
[7]  # splitting function for splitting dataset into test set, train set and unlabel pool

     def split(dataset, train_size, test_size):
         x = dataset[:, :-1]
         y = dataset[:, -1]
         x_train, x_pool, y_train, y_pool = train_test_split(
             x, y, train_size = train_size)
         unlabel, x_test, label, y_test = train_test_split(
             x_pool, y_pool, test_size = test_size)
         return x_train, y_train, x_test, y_test, unlabel, label
```

Fig. Code Snippet for Split

- **Classifier Used:** Support Vector Machine Classifier

This brings us to the end of Q1(a). We have formulated an Active Learning problem with a split dataset and are now going to process it in the further sections.

Uncertainty Sampling

Q: Use uncertainty sampling (Least confident, Margin Sampling, & Entropy) to label points. Compare the three measures of informativeness.

In Active Learning, the process by which the learning algorithm selects the data points to be labelled from the pool of unlabelled data points is known as **Sampling.**
Broadly, sampling can be performed in two ways:

- **Query by Committee (QBC) or Disagreement Sampling**: discussed in the next section

- **Uncertainty Sampling**: It is a learning strategy for identifying unlabeled data points that are near the decision boundary in our current Machine Learning model. We present unlabelled examples to an active learner, which in turn finds the most *informative* examples and asks the oracle to label. This is done by calculating the *informativeness* of prediction for each unlabelled data point. There three major ways to measure this informativeness:

    - **Least Confidence (LC)**: in this strategy, the learner selects the instance for which it has the least confidence in its most likely label. This confidence is inversely proportional to the probability of the class of the predicted label. The following figure shows the probability of different classes for the 5 most informative data points according to our active learner (as you may notice, in all top 5 samples shown below, most probable class has its probability close to 0.1, which suggests that the classifier is not able to confidently classify them. Thus, they are most informative)

$$x^*_{LC} = \underset{x}{\operatorname{argmax}}\ 1 - P_\theta(\hat{y}|x),$$

```
[ ]  # function defining all types of uncertainity sampling techniques (least confidence, margin, entropy)

     def uncertainity_sampling(name, unlabel, classifier, percent_of_samples, dataset):

         # For LEAST CONFIDENCE SAMPLING
         if name == 'least_confidence':
             proba = classifier.predict_proba(unlabel)
             prob_max = proba.max(axis=1)
             sorted_idx = np.argsort(prob_max)
             #print('Probability distribution of Top 5 Selection through LEAST CONFIDENCE: \n')
             #for i in range(1,6):
                 #print("Selection #" + str(i) + ': ' + str(proba[sorted_idx[i-1]]) + '\n')

             uncrt_pt_ind = []
             for i in range(math.floor(percent_of_samples * 0.01 * dataset.shape[0])):
                  uncrt_pt_ind.append(sorted_idx[i])
             return uncrt_pt_ind
```

```
[ ]  print('The probablity distributions of top 5 samples chosen by LEAST CONFIDENCE SAMPLING are: \n')
     print('Selection# 1: ' + str(proba_chosen_samples[0]) + '\n')
     print('Selection# 2: ' + str(proba_chosen_samples[1]) + '\n')
     print('Selection# 3: ' + str(proba_chosen_samples[2]) + '\n')
     print('Selection# 4: ' + str(proba_chosen_samples[3]) + '\n')
     print('Selection# 5: ' + str(proba_chosen_samples[4]) + '\n')
```

```
The probablity distributions of top 5 samples chosen by LEAST CONFIDENCE SAMPLING are:

Selection# 1: [0.03377535 0.13482272 0.13504004 0.08105982 0.10158878 0.12600726
 0.13278614 0.07105668 0.09960072 0.08426248]

Selection# 2: [0.03217007 0.12288759 0.15259691 0.08537868 0.10531505 0.15464832
 0.08110345 0.08902137 0.091273   0.08560556]

Selection# 3: [0.03139759 0.12704485 0.10618448 0.08718111 0.15376551 0.15538316
 0.03700761 0.1421501  0.07338214 0.08650345]

Selection# 4: [0.03207068 0.12534214 0.15832235 0.0870851  0.10644321 0.14958801
 0.07687967 0.09132436 0.09251313 0.08043133]

Selection# 5: [0.12164034 0.06479183 0.11005472 0.16149169 0.08266774 0.14254209
 0.08348967 0.05748808 0.08821526 0.08761857]
```

Fig. Code Snippet for Best 5 KL Least Confidence Instances

- **Margin Sampling:** The margin sampling strategy selects the instance that has the smallest difference between the first and second most probable labels (as evident from the top 5 most informative samples shown below: difference between the most and second most probable class is very less in all of them).

$$x_M^* = \underset{x}{\operatorname{argmin}}\, P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)$$

```
# For MARGIN SAMPLING
if name == 'margin_sampling':
    proba = classifier.predict_proba(unlabel)
    part = np.partition(-proba, 1, axis=1)
    margin = - part[:, 0] + part[:, 1]
    sorted_idx = np.argsort(margin)
    # print('Probability distribution of Top 5 Selection through MARGIN SAMPLING: \n')
    # for i in range(1,6):
    #     print("Selection #" + str(i) + ': ' + str(proba[sorted_idx[i-1]]) + '\n')
    uncrt_pt_ind = []
    for i in range(math.floor(percent_of_samples * 0.01 * dataset.shape[0])):
        uncrt_pt_ind.append(sorted_idx[i])
    return uncrt_pt_ind
```

```
print('The probablity distributions of top 5 samples chosen by MARGIN SAMPLING are: \n')
print('Selection# 1: ' + str(proba_chosen_samples[0]) + '\n')
print('Selection# 2: ' + str(proba_chosen_samples[1]) + '\n')
print('Selection# 3: ' + str(proba_chosen_samples[2]) + '\n')
print('Selection# 4: ' + str(proba_chosen_samples[3]) + '\n')
print('Selection# 5: ' + str(proba_chosen_samples[4]) + '\n')
```

```
The probablity distributions of top 5 samples chosen by MARGIN SAMPLING are:

Selection# 1: [0.01926678 0.37285199 0.07381158 0.015624   0.02509358 0.02569371
 0.0117287  0.05324851 0.37365496 0.02902621]

Selection# 2: [0.02979727 0.1323283  0.16063684 0.15769669 0.11763358 0.04533803
 0.08252795 0.16204096 0.05684729 0.05515311]

Selection# 3: [0.03029646 0.13613896 0.16604292 0.16389448 0.1166174  0.04637432
 0.08414068 0.14132759 0.05840498 0.05676221]

Selection# 4: [0.01702301 0.42329887 0.04808375 0.02106894 0.01863101 0.00782095
 0.01121724 0.01480359 0.42555753 0.01249511]

Selection# 5: [0.03882487 0.066998   0.13341152 0.14293491 0.06960307 0.18110725
 0.04385095 0.17878818 0.09615192 0.04832933]
```

Fig. Code Snippet for Best 5 Margin Instances

- **Entropy:** The entropy formula is applied to the probabilities of each instance and the instance with the largest value of entropy is queried.

$$x_H^* = \underset{x}{\operatorname{argmax}} - \sum_i P_\theta(y_i|x) \log P_\theta(y_i|x)$$

```
# For ENTROPY SAMPLING
if name == 'entropy':
    proba = classifier.predict_proba(unlabel)

    entr = entropy(proba.T)
    sorted_idx = np.argsort(entr)
    # print('Probability distribution of Top 5 Selection through ENTROPY: \n')
    # for i in range(1,6):
    #     print("Selection #" + str(i) + ': ' + str(proba[sorted_idx[len(entr) - i]]) + '\n')
    uncrt_pt_ind = []
    for i in range(math.floor(percent_of_samples * 0.01 * dataset.shape[0])):
        uncrt_pt_ind.append(sorted_idx[-i -1])
    return uncrt_pt_ind
```

```
print('The probablity distributions of top 5 samples chosen by ENTROPY SAMPLING are: \n')
print('Selection# 1: ' + str(proba_chosen_samples[0]) + '\n')
print('Selection# 2: ' + str(proba_chosen_samples[1]) + '\n')
print('Selection# 3: ' + str(proba_chosen_samples[2]) + '\n')
print('Selection# 4: ' + str(proba_chosen_samples[3]) + '\n')
print('Selection# 5: ' + str(proba_chosen_samples[4]) + '\n')
```

```
The probablity distributions of top 5 samples chosen by ENTROPY SAMPLING are:

Selection# 1: [0.01843127 0.10059916 0.10384636 0.05831105 0.18564742 0.11280405
 0.04999152 0.09330931 0.17123753 0.10582233]

Selection# 2: [0.02041818 0.11913466 0.11391838 0.09161612 0.09971817 0.03921163
 0.12300953 0.04846477 0.17520992 0.16929864]

Selection# 3: [0.02282844 0.18752423 0.07538208 0.06432988 0.100367   0.03432487
 0.08780538 0.09791623 0.13527377 0.19424812]

Selection# 4: [0.0223244  0.08344535 0.09678973 0.06897379 0.09165202 0.03690082
 0.14552922 0.06006769 0.23700035 0.15731664]

Selection# 5: [0.02417314 0.04793493 0.15768859 0.06870072 0.1286465  0.10324625
 0.0855158  0.02568961 0.22345608 0.1349484 ]
```

Fig. Code Snippet for Best 5 Entropy Instances

Comparison of these Uncertainty Sampling techniques:

In the code file, we have shown the performance of each of these techniques with 3 metrics: **Accuracy, F1 Score & Precision Score.**

Given below is a table comparing each of their accuracies after each iteration for 10% unlabelled data:

Method	Accuracy @ 10% sampling	Accuracy @ 20% sampling	Accuracy @ 30% sampling	Accuracy @ 40% sampling
Least Cost	96.914%	98.457%	98.765%	98.457%
Margin	97.840%	98.457%	98.765%	98.455%
Entropy	95.988%	98.457%	98.457%	98.458%

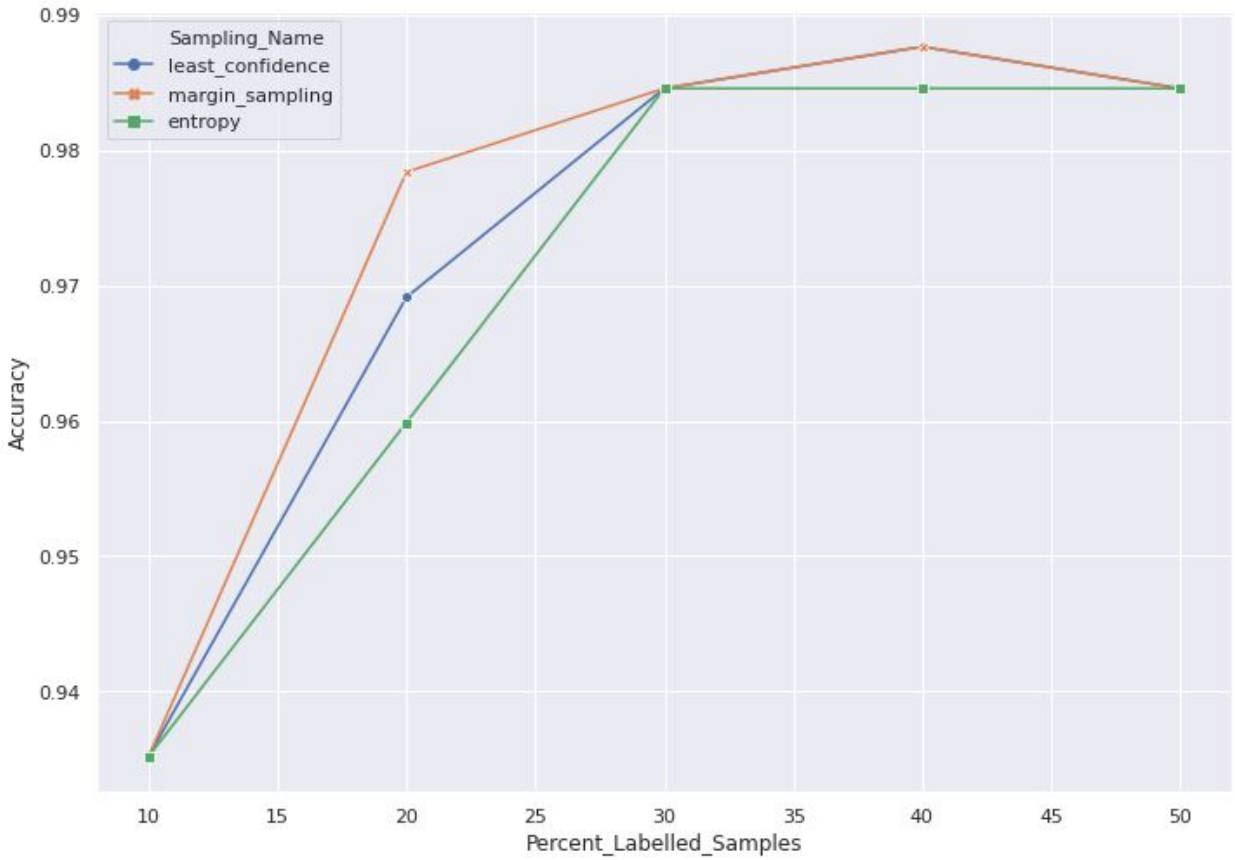


Fig. Least Confidence vs Margin vs Entropy Sampling

QBC/ Disagreement Sampling

Q: Use QBC (Vote Entropy & KL divergence) with at least 5 committee members, to label points. Compare the two measures of disagreement.

In Active Learning, the process by which the learning algorithm selects the data points to be labelled from the pool of unlabelled data points is known as **Sampling.**
Broadly, sampling can be performed in two ways:

- **Uncertainty Sampling:** discussed in the previous section.

- **Query by Committee (QBC)/ Disagreement Sampling**: A committee of different classification models is trained on the same labeled training set and used for inference on the rest of the (yet unlabeled) data. The instances for which the largest disagreement is observed are deemed to be the most informative ones. The core idea behind this framework is minimizing the version space, which is the set of hypotheses that are consistent with the current labeled training data. For our purpose, we used a committee of **5 Bagging Classifiers** consisting of SVMs. The disagreement can be measured in two ways:
    - **Vote Entropy**: It is a measure of difference between two probability distributions: *P* & *Q*.

$$x^*_{VE} = \underset{x}{\operatorname{argmax}} - \sum_i \frac{V(y_i)}{C} \log \frac{V(y_i)}{C},$$

V(yi): number of votes received by a label (yi) from among the committee members' predictions

C: Committee size

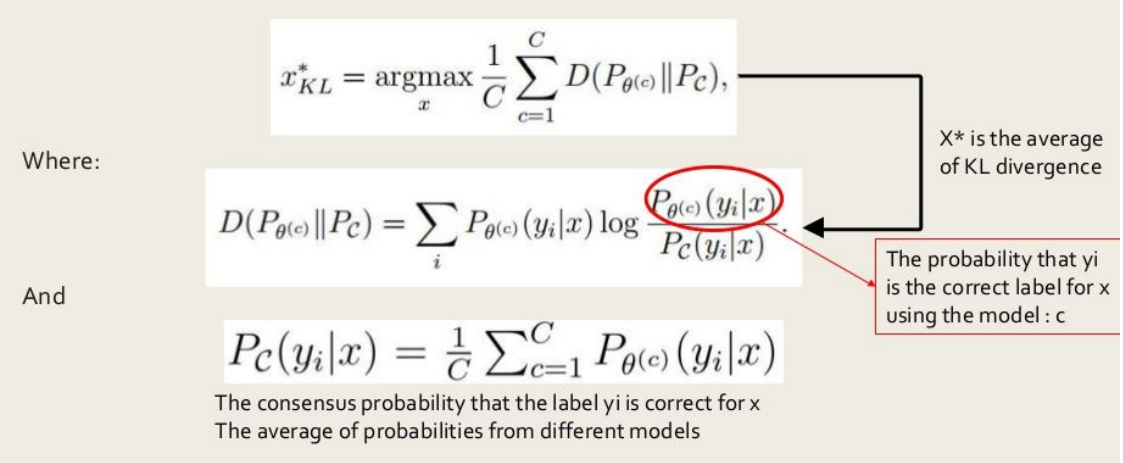
```
Accuracy by ACTIVE MODEL [with Vote Entropy Sampling] : 95.5246913580247
Precision Score by ACTIVE MODEL [with Vote Entropy Sampling] : 0.9541830937935609
F1 Score by ACTIVE MODEL [with Vote Entropy Sampling] : 0.9540381407999916
```

```
print('The probablity distributions of top 5 samples chosen by KL DIVEREGENCE are: \n')
print('Selection# 1: ' + str(proba_chosen_samples[0]) + '\n')
print('Selection# 2: ' + str(proba_chosen_samples[1]) + '\n')
print('Selection# 3: ' + str(proba_chosen_samples[2]) + '\n')
print('Selection# 4: ' + str(proba_chosen_samples[3]) + '\n')
print('Selection# 5: ' + str(proba_chosen_samples[4]) + '\n')
```

```
The probablity distributions of top 5 samples chosen by KL DIVEREGENCE are:

Selection# 1: [0.03841254 0.20098409 0.04271278 0.025526   0.21385441 0.08048011
 0.0226037  0.06058329 0.1923056  0.12253748]

Selection# 2: [0.02345869 0.04022912 0.01668927 0.01216987 0.11282953 0.07912745
 0.01876442 0.19067821 0.17024412 0.33580932]

Selection# 3: [0.01973604 0.02983349 0.02085076 0.01134197 0.03137651 0.06439616
 0.00978771 0.39884728 0.35316273 0.06066735]

Selection# 4: [0.09381487 0.01895843 0.02245913 0.02059961 0.02558426 0.0367214
 0.01566272 0.03074603 0.30786667 0.42758689]

Selection# 5: [0.02351073 0.06345047 0.2298963  0.03214776 0.17063151 0.05847947
 0.02022989 0.26243369 0.08992639 0.04929379]
```

Fig. Code Snippet for Best 5 Vote Entropy Instances

- **KL Divergence:** It is the expected value of the difference between two probability distributions: *P* & *Q*.



```
[ ]  Accuracy by ACTIVE MODEL [with KL Divergence Sampling] : 96.2037037037037
     Precision Score by ACTIVE MODEL [with KL Divergence Sampling] : 0.960996900768426
     F1 Score by ACTIVE MODEL [with KL Divergence Sampling] : 0.960774778365789
```

```
[ ]  print('The probablity distributions of top 5 samples chosen by KL DIVEREGENCE are: \n')
     print('Selection# 1: ' + str(proba_chosen_samples[0]) + '\n')
     print('Selection# 2: ' + str(proba_chosen_samples[1]) + '\n')
     print('Selection# 3: ' + str(proba_chosen_samples[2]) + '\n')
     print('Selection# 4: ' + str(proba_chosen_samples[3]) + '\n')
     print('Selection# 5: ' + str(proba_chosen_samples[4]) + '\n')
```

```
     The probablity distributions of top 5 samples chosen by KL DIVEREGENCE are:

     Selection# 1: [0.02755374 0.09405863 0.10053627 0.046855   0.15759883 0.10292011
      0.07200532 0.15020006 0.07349175 0.17478031]

     Selection# 2: [0.02299101 0.10335342 0.11296559 0.0420826  0.17784307 0.08944557
      0.04537351 0.18351258 0.06011787 0.16231477]

     Selection# 3: [0.02264676 0.10487624 0.11515956 0.04260633 0.16848698 0.08951956
      0.04473493 0.18750838 0.06010379 0.16435747]

     Selection# 4: [0.02264674 0.10487603 0.11515933 0.04260628 0.16848871 0.08951939
      0.04473487 0.18750776 0.06010369 0.1643572 ]

     Selection# 5: [0.02264556 0.10486617 0.1151984  0.04261283 0.16845455 0.08951681
      0.04473179 0.18750502 0.06011037 0.1643585 ]
```

Fig. Code Snippet for Best 5 KL Divergence Instances

Comparison of these QBC Sampling techniques:

In the code file, we have shown the performance of each of these techniques with 3 metrics: **Accuracy, F1 Score & Precision Score.**

Given below is a table comparing each of their accuracies after each iteration for 10% unlabelled data:

Method	Accuracy @ 10% sampling	Accuracy @ 20% sampling	Accuracy @ 30% sampling	Accuracy @ 40% sampling
Vote Entropy	94.444%	98.148%	97.839%	98.446%
KL Divergence	97.222%	98.148%	97.531%	98.148%

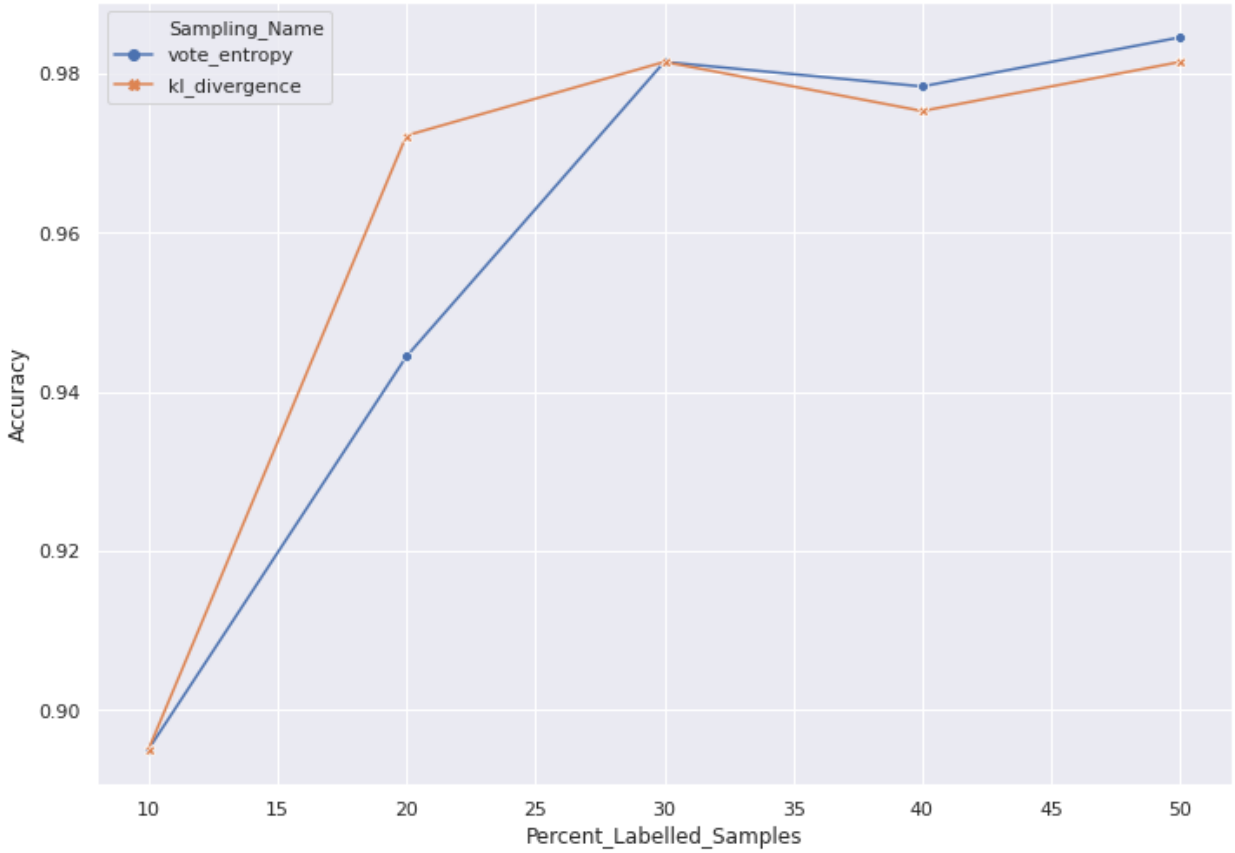
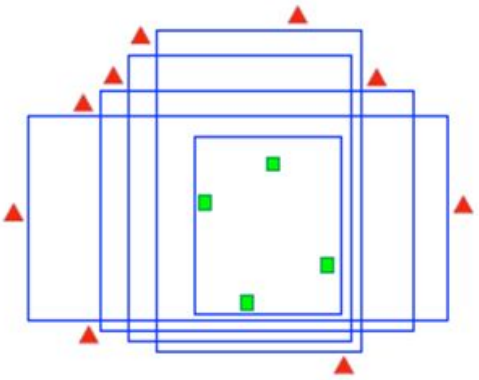


Fig. Vote Entropy vs KL Divergence

Version Space

Q: What is the size (number of points) of the version space? Order points to label in such a way that the version space gets reduced by maximum with each point chosen to be labelled.

Version space is the region that is still unknown to the overall model class, i.e., version space is the set of hypotheses that are consistent with the current labeled training data.
In other words, if any two models of the same model class (but different parameter settings) agree on all the labeled data, but disagree on some unlabeled instance, then that instance lies within the region of uncertainty.



- To reduce the version space, we label the points with the highest disagreement according to (vote entropy approach, as it serves better results for our dataset)
- We store the ordered points to label, in such a way that the version space gets reduced by maximum with each point chosen to be labelled, in the array named: *uncrt_pt_ind*
- We obtain 10% of the unlabelled points for having the highest disagreement and add them to our active learner's training data. Number of points in the version in the version space is measured before and after the addition of these 10% of unlabelled data points and the difference was noted.

The results are as follows:

Size of Version Space (before adding these 10% points)	33 points
Size of Version Space (after adding these 10% points)	13 points
Percentage reduction in Version Space	**60.61%**

Comparisons

Q: Incorporate the additional labelled points (separately, the best from i & ii) into your model and compare with corresponding models trained with randomly chosen labelled points. Also, compare with a stream-based scenario.

- **Least Cost** vs **Margin** vs **Entropy** vs **Vote Entropy** vs **KL Divergence**

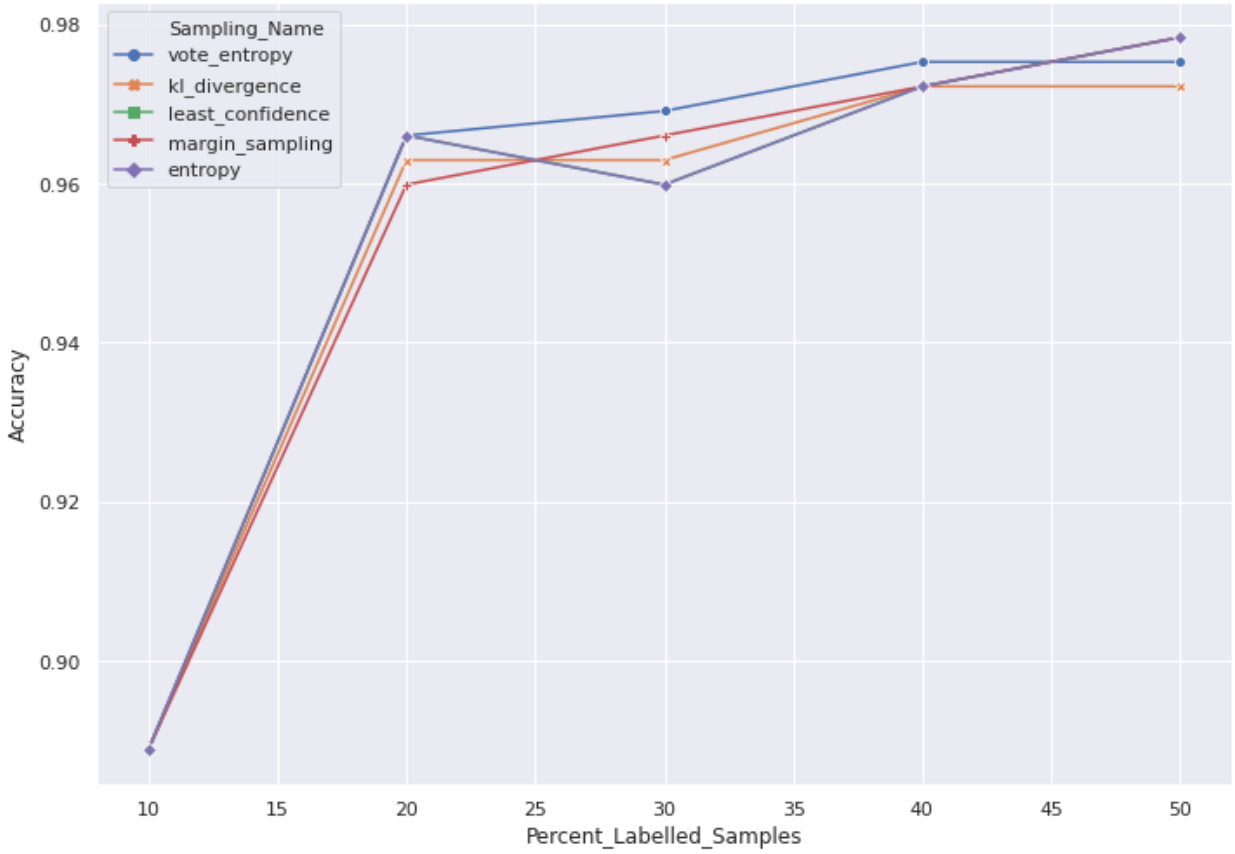


Fig. All Sampling Techniques Comparison

- **Stream Based Scenario** vs **Pool Based Scenario:**

  Sampling measure used: **ENTROPY**

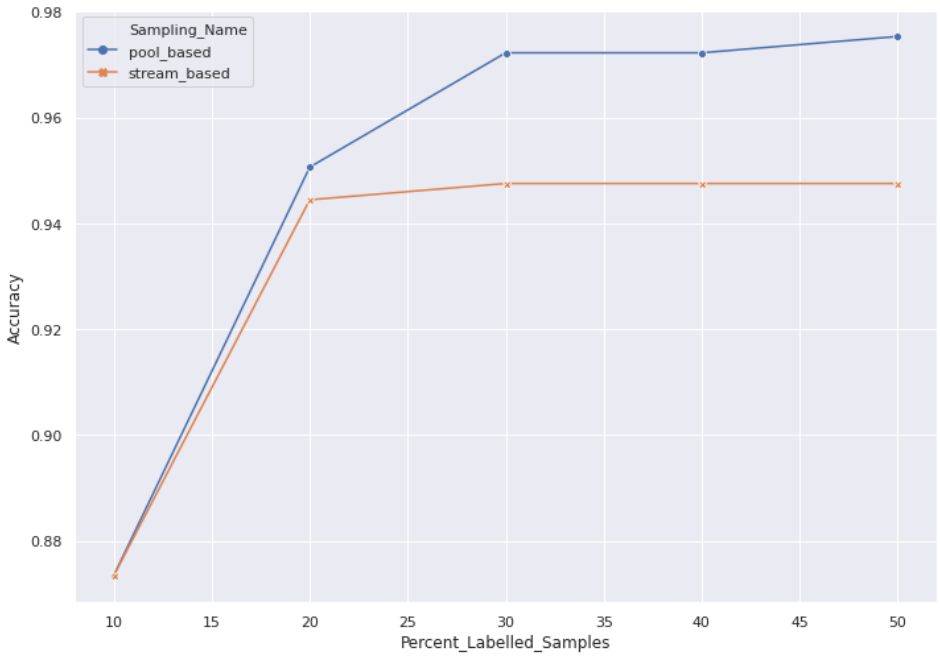


Fig. Pool Based vs Stream Based Scenario

- **Active Learning** vs **Random Learning:**

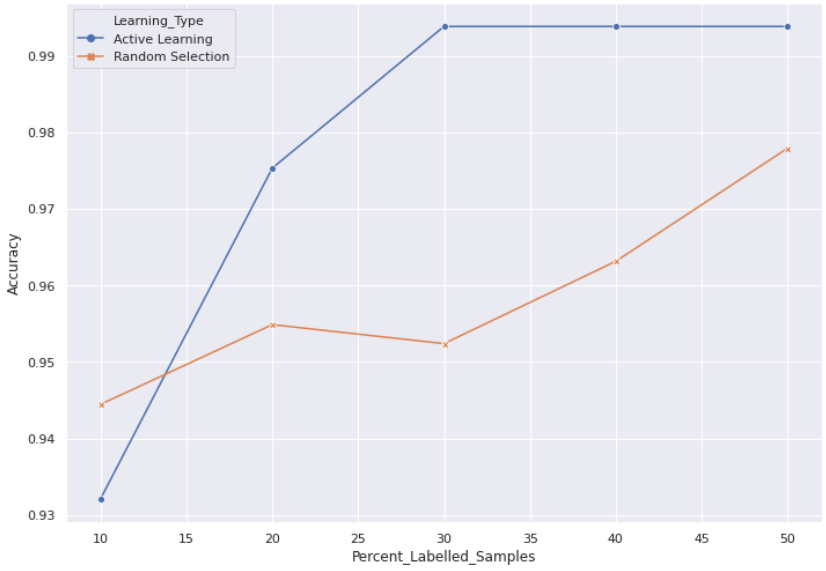
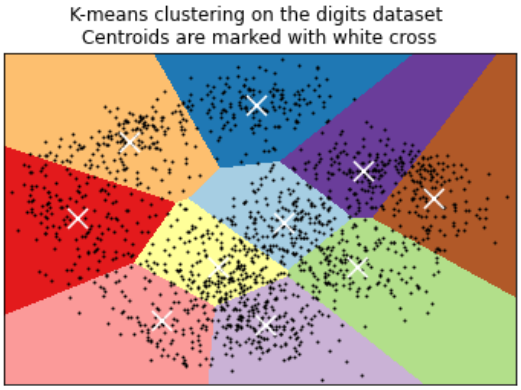


Fig. Active Learning vs Random Learning

KMeans Clustering

Q: From 90% of unlabelled points, randomly pick 40% of the points and use clustering (using K-means with K=number of class labels). In each cluster, randomly label 20% of the points to label remaining points in the cluster. How accurate is the cluster-based labelling? How much saving it results in if each label costs you Rs. 100 and each labelling takes one hour.



- **Clustering 40% of 90% left out points:**
  - Clustering Algorithm = KMeans
  - Number of Data Points = 647
  - Number of Features = 64
  - Number of Clusters = 10

  These points are clustered as follows:

- **Predicting Cluster labels using 20% labelled points in each cluster:**

  These 647 data points are along with their respective indices and correct labels are given to a function which uses the 20% correctly labelled points to label the complete clusters and returns an array of the assigned labels corresponding to each data point's index. This function is shown in the following code snippet:

```python
# Function to label clusters using 20% (correctly labelled) points of these 40% clustered points
def label_clusters_with_20_percent(data_40_percent, indices):

    # Segregating the features and labels of the 40% data points
    kmeans_x_train = data_40_percent[:, :-1]
    kmeans_y_train = data_40_percent[:, -1]

    dict_20_percent = {}
    clusters_20_percent_label = {}
    cluster_label = {}
    kmeans_cluster_labels = np.zeros(kmeans_y_train.shape)
    for i in indices:

        # Randomly picking 20% of the 40%
        dict_20_percent[i] = np.array(random.sample(list(indices[i]), math.ceil(indices[i].shape[0] * 0.2)))

        # Labelling these 20% points
        clusters_20_percent_label[i] = kmeans_y_train[dict_20_percent[i]]

        # Most frequently element in this 20% labels in each cluster
        cluster_label[i] = max(set(list(clusters_20_percent_label[i])), key = list(clusters_20_percent_label[i]).count)

        for j in indices[i]:
            # Assigning the values to respective most frequently value to each cluster
            kmeans_cluster_labels[j] = cluster_label[i]

    return kmeans_cluster_labels
```

Fig. Code Snippet for 20% Labelling

- **Accuracy of Clustering:**

  After testing these labelled clustered, the accuracy achieved is **80%** (approx.)

  While the accuracy of Active Learning after 10 percent sampling is **96%** (approx.)