ASSIGNMENT #2

ACTIVE LEARNING

&

SELF ORGANISING MAPS (SOM)

BITS F464 - MACHINE LEARNING

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI, PILANI CAMPUS

SUBMITTED BY –

REVENTH SHARMA – 2017A1PS0832P

PRAKHAR AGNIHOTRI – 2017A8PS0280P

BHAGYAM GUPTA – 2017A8PS0525P

SUBMITTED TO -

NAVNEET GOYAL SIR

Abstract

The following assignment deals with active machine learning approach to classify wheat grain based on geometric parameters. It also deals with clustering different animal species w.r.t their physiological characteristics using Self Organizing Maps.

ACTIVE LEARNING

1. Introduction:

Active learning is a subset of machine learning in which a learning algorithm can interactively query a user (or some other information source) to label new data points with the desired outputs [semi-supervised learning]. Source of information is referred as teacher or oracle. Active learning is a way to achieve better accuracy with less training. In other words "The process of guiding the sampling process by querying for certain types of instances based upon the data that we have seen so far is called active learning". Whereas collecting and labelling the data randomly from an underlying population distribution is Passive learning. Hence, Active learning systems attempt to overcome the labeling bottleneck by asking queries in the form of unlabeled instances to be labeled by an oracle.

Active Learning majorly falls into types as: membership query synthesis (examples to be queried generated de novo), stream based(considers an unlabeled example and decides to query it or not) and pool based learning(unlabeled examples ranked based on pool of informativeness and most informative n-examples are queried).

Any Active Learning algorithm requires a query selection strategy. These strategies aim to find out the most informative instance or sample among the given unlabeled samples to label it so that we can apply them repeatedly on an unlabeled data sets to achieve the required set of most informative labeled data points. Some examples of such strategies are Uncertainty Sampling, Query by Committee

(QBC), Expected Model Change, Expected Error Reduction, Variance Reduction, Density Weighted Methods, etc. In this assignment we will explore the domain of Uncertainty sampling and Query by Committee.

Uncertainty Sampling is one of the commonly used selection strategy used in classification problems and work on the principle of selecting the event to query that the current classifier is most uncertain about. Active learner queries instances about which it is least certain how to label. There are three ways to implement uncertainty sampling selection strategy –

- Least confident – In this scheme we figure out the most probable class for each sample to which it can belong and then we lookout for an index which can tell us how certain we are about a sample belonging to its most probable class. Sample with least amount of certainty in belonging to its most probable class is selected for query.
- Margin sampling – In least confident scheme we took the most probable class for each sample and didn't look for the possibility of that sample belonging to other classes , so in this scheme we take two most probable classes for each sample and then figure out the certainty in deciding among both classes as difference in probability of these classes. Sample with most uncertainty in belonging to any one of the two most probable classes is selected for query.
- Entropy – This scheme uses entropy as measure to uncertainty. This scheme takes entire distribution of probability of each sample belonging to any of the classes and then uses entropy to look for the most uncertain sample.

Query by committee (QBC) is another commonly used selection strategy which involves maintaining a committee of models which are all trained on the current labeled data L, but represent competing hypotheses. Each committee member

(model) is allowed to vote on the labeling of query candidates and most informative query is the one about which the committee most disagree. This strategy tries to minimize the version space (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 L but inconsistent with some unlabeled data U. 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 and is called version space. Any point in the version space is guaranteed to have some disagreements and any point outside the version space is guaranteed to have no disagreements. In Active Learning, we try to constrain the size of the version space as much as possible, so that the search can be more precise with as few labeled instances as possible contrary to Machine Learning where we search for best model in version space.

The informativeness in Query Based Committee method is measured in 2 ways:

- Vote Entropy: The vote probability distribution of each class is measured as the ratio of (votes received by the Class)/(total number of Votes). The data point for which the entropy of this probability distribution is highest is chosen for query.
- KL Divergence: Probabilities of each class for each Committee model for every datapoint are calculated. For a data point consensus probability of a class is average of probabilities for class from each committee model. For each committee learner-i we measure the entropy of a data point's class probability (from committee learner-i) w.r.t consensus probability. For a datapoint, the maximum entropy of all the committee learners is max

disagreement of that point. The datapoints are selected from high to low max disagreement.

2. Dataset

- The seed classification dataset (https://archive.ics.uci.edu/ml/datasets/seeds) is used for this assignment. The seeds belong to 3 different varieties of wheat: Kama, Rosa and Canadian with 70 instances for each type. High quality visualization of internal kernel structure using X-ray technique was done to measure 7-real valued continuous geometric parameters (area A, perimeter P, compactness C = 4*pi*A/P^2, length of kernel, width of kernel, asymmetry coefficient, length of kernel groove) for wheat-kernel. The seven parameters will be used to classify wheat amongst 3 classes {0: Kama, 1: Rosa, 2: Canadian}. The attributes for seed classification are as follows:

3. Data Pre-processing

The dataset was searched for any null values and found nil. All the parameters are standard normalized. Random shuffling of whole dataset was done to distribute classes evenly amongst whole data.

3.) Active Learning

- Pool-based uncertainty sampling:

The data is initially trained with random forest classifier for randomly selected 10% labelled data. The remaining 90% data unlabelled data is ranked based on informativeness. The top 10%, 20%, 30% and 40% informative data points (least confident) are selected for querying by human oracle. Fig-1 shows the graph of informativeness vs. number of data points queried.

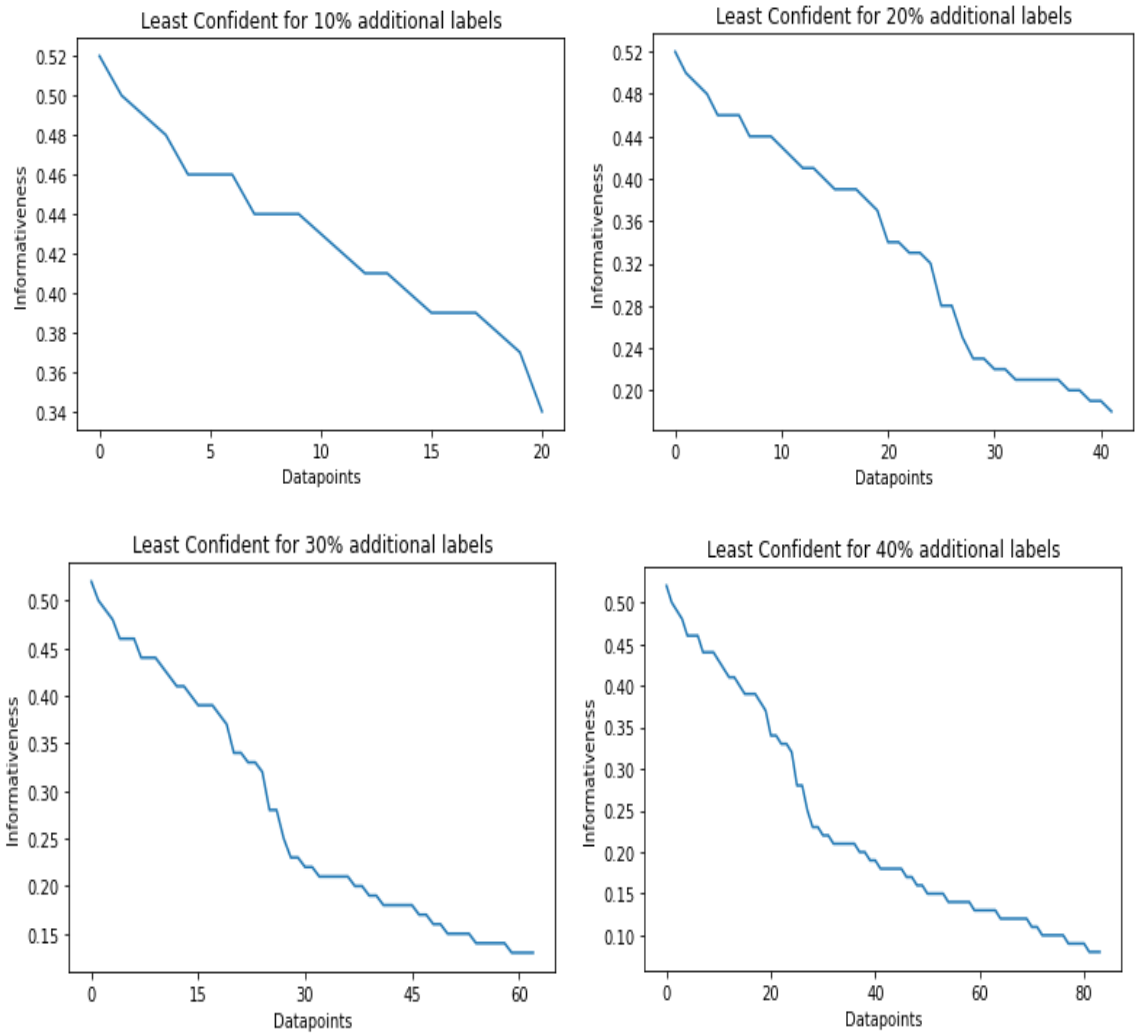


Fig-1

It can be seen as more data points are queried the informativeness of each successive data point decrease as model becomes more and more accurate.

- Pool based margin sampling:

The data is initially trained for randomly selected with random forest classifier 10% labelled data. The remaining 90% data unlabelled data is ranked based on informativeness. The top 10%, 20%, 30% and 40% informative data points (smallest margin) are selected for querying by human oracle. Fig-2 shows the graph of informativeness vs. number of data points queried.

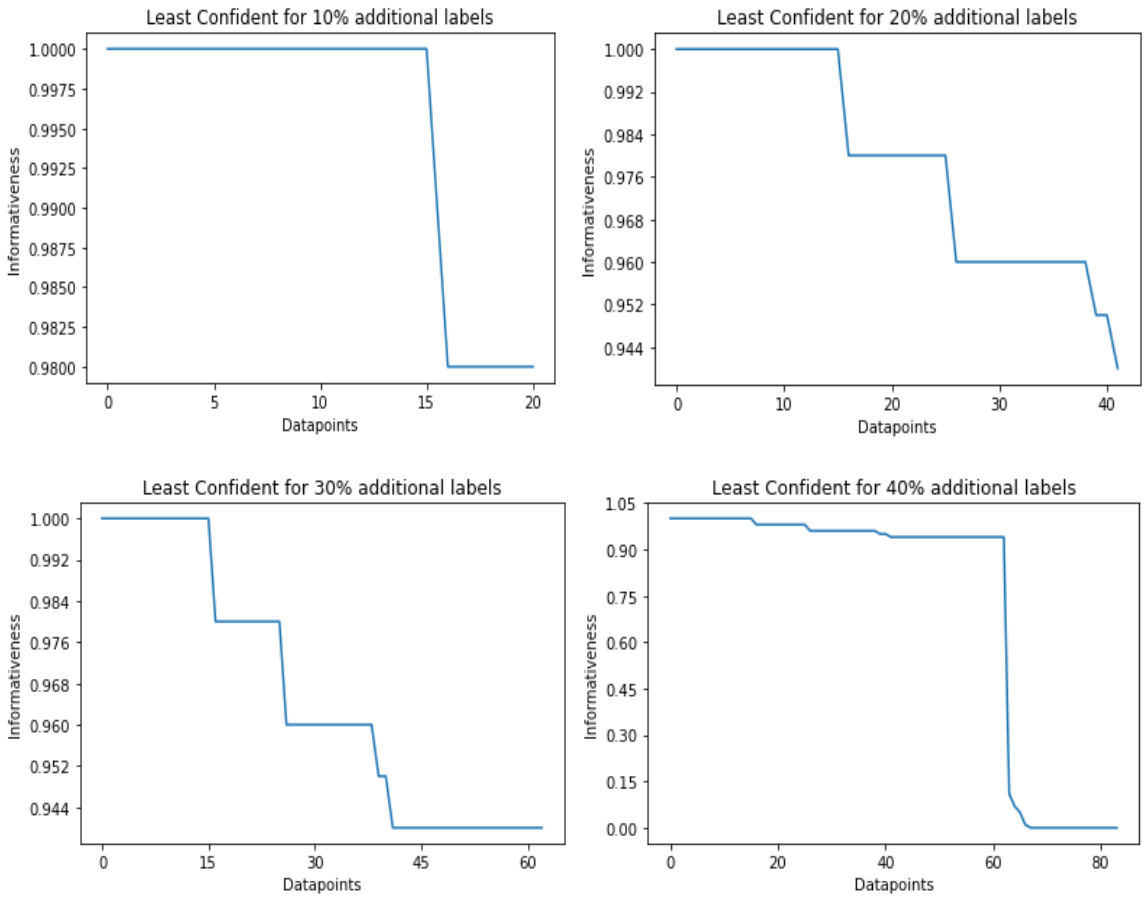


Fig-2

The trend is similar to fig-1 but the smoothness has decreased.

- Pool based entropy sampling:

The data is initially trained with random forest classifier for randomly selected 10% labelled data. The remaining 90% data unlabelled data is ranked based on informativeness. The top 10%, 20%, 30% and 40% informative data points (highest entropy) are selected for querying by human oracle. Fig-3 shows the graph of informativeness vs. number of data points queried.

The trend is similar to fig-1 and fig-2 but with highest smoothness amongst all signifying gradual increase in model accuracy and avoidance of local minima.

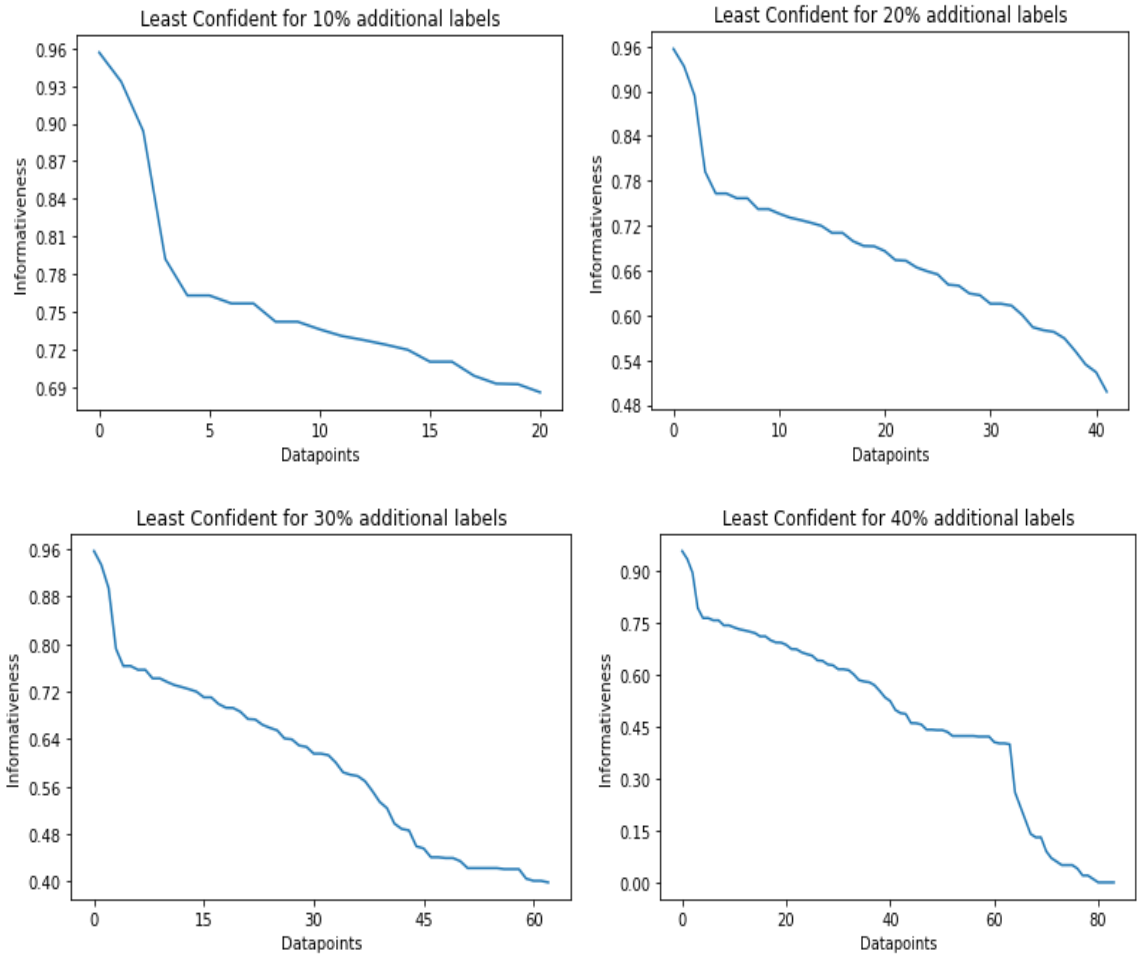
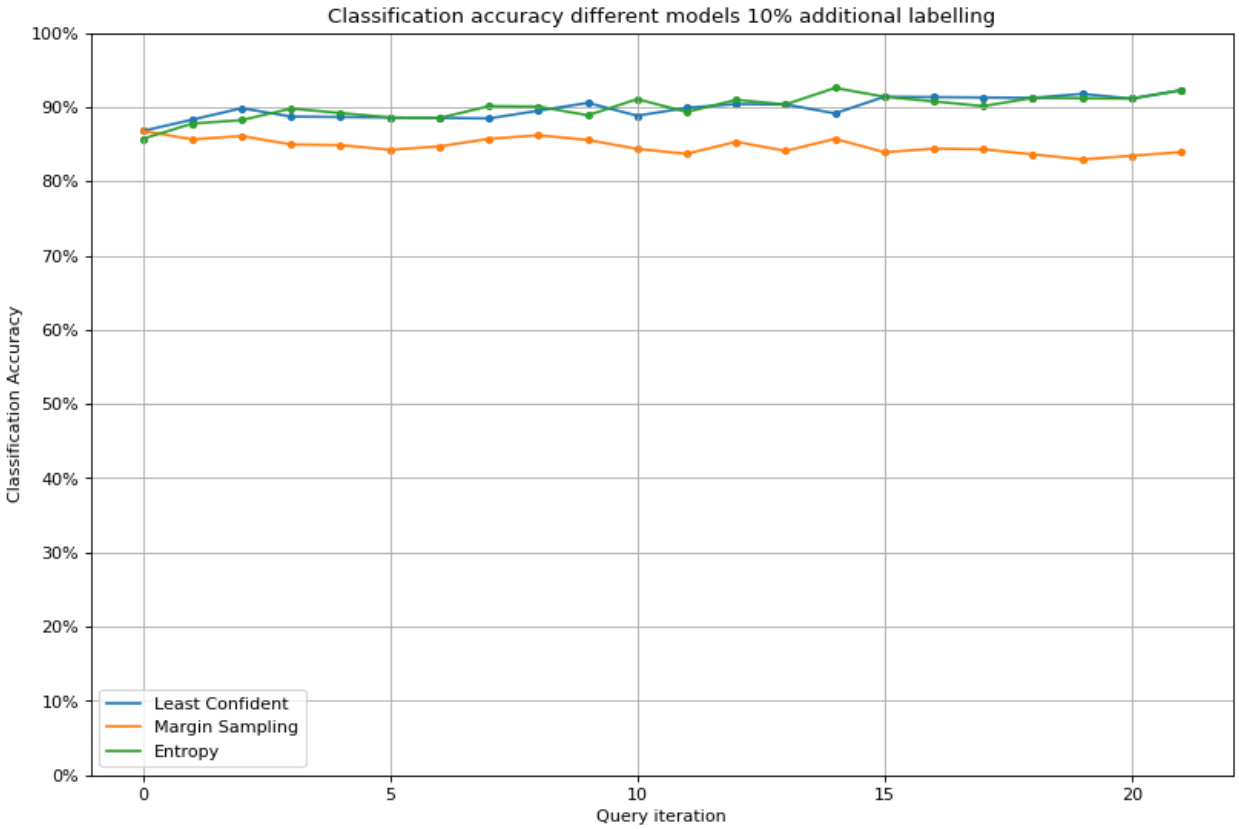
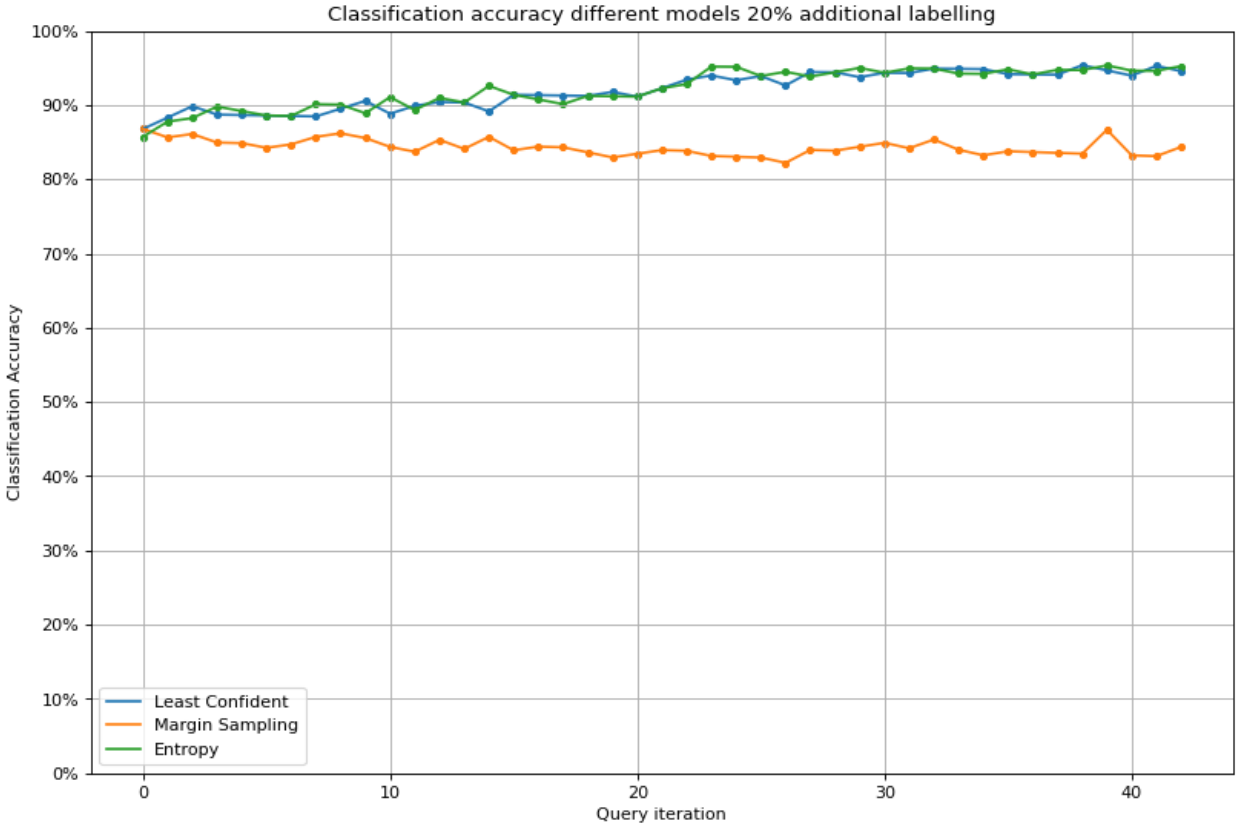
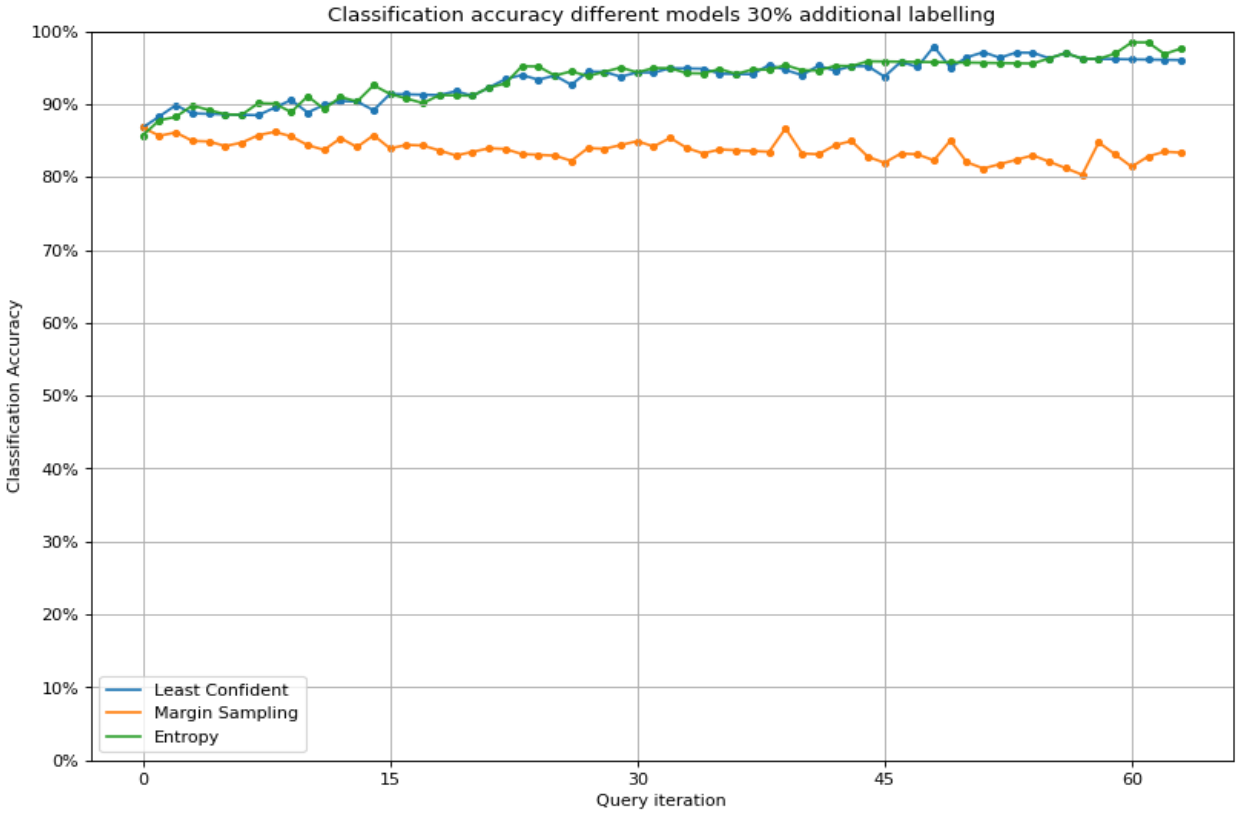


Fig-3

The classification accuracy of 3 orders of comparativeness w.r.t number of queries labelled by human oracle is given in fig-4.


Classification accuracy different models 10% additional labelling
Classification Accuracy
Query iteration
Least Confident
Margin Sampling
Entropy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
5
10
15
20


Classification accuracy different models 20% additional labelling
Classification Accuracy
Query iteration
Least Confident
Margin Sampling
Entropy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
10
20
30
40


Classification accuracy different models 30% additional labelling
Classification Accuracy
Query iteration
Least Confident
Margin Sampling
Entropy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
15
30
45
60

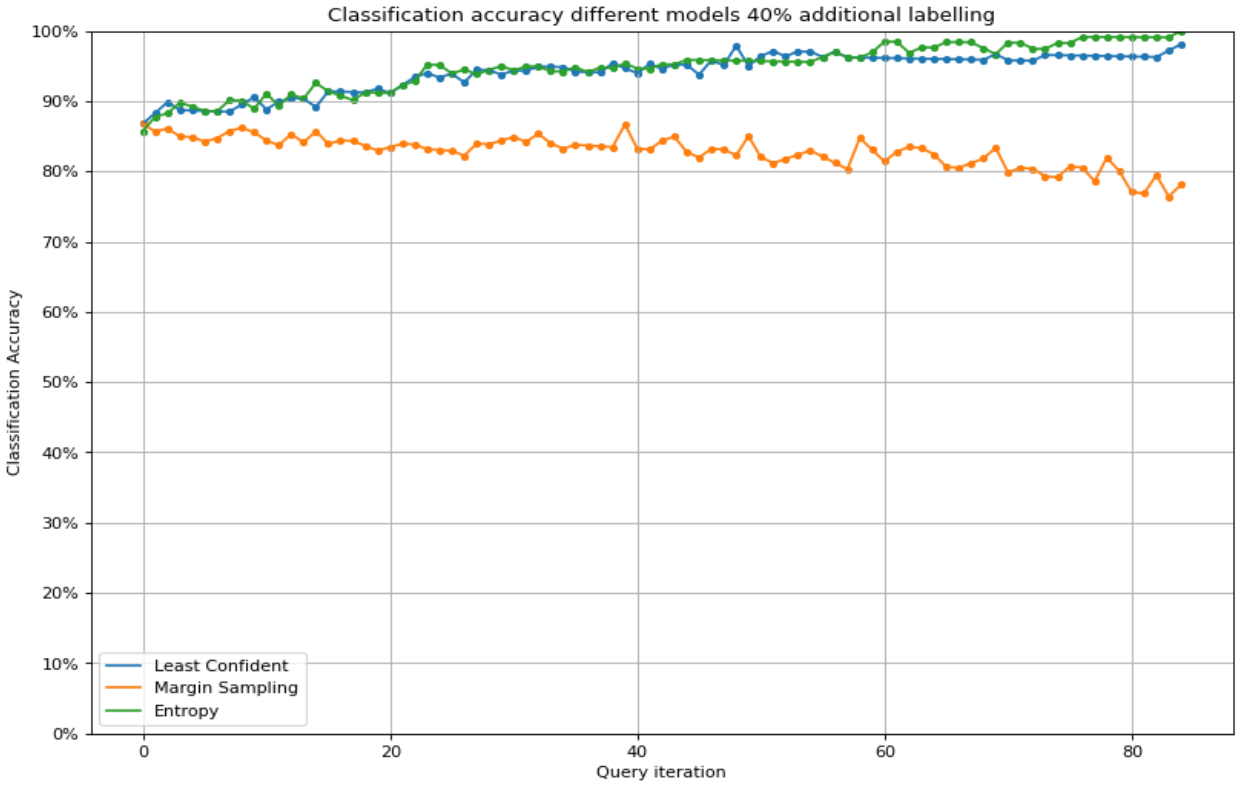

Classification accuracy different models 40% additional labelling
Classification Accuracy
Query iteration
Least Confident
Margin Sampling
Entropy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
20
40
60
80

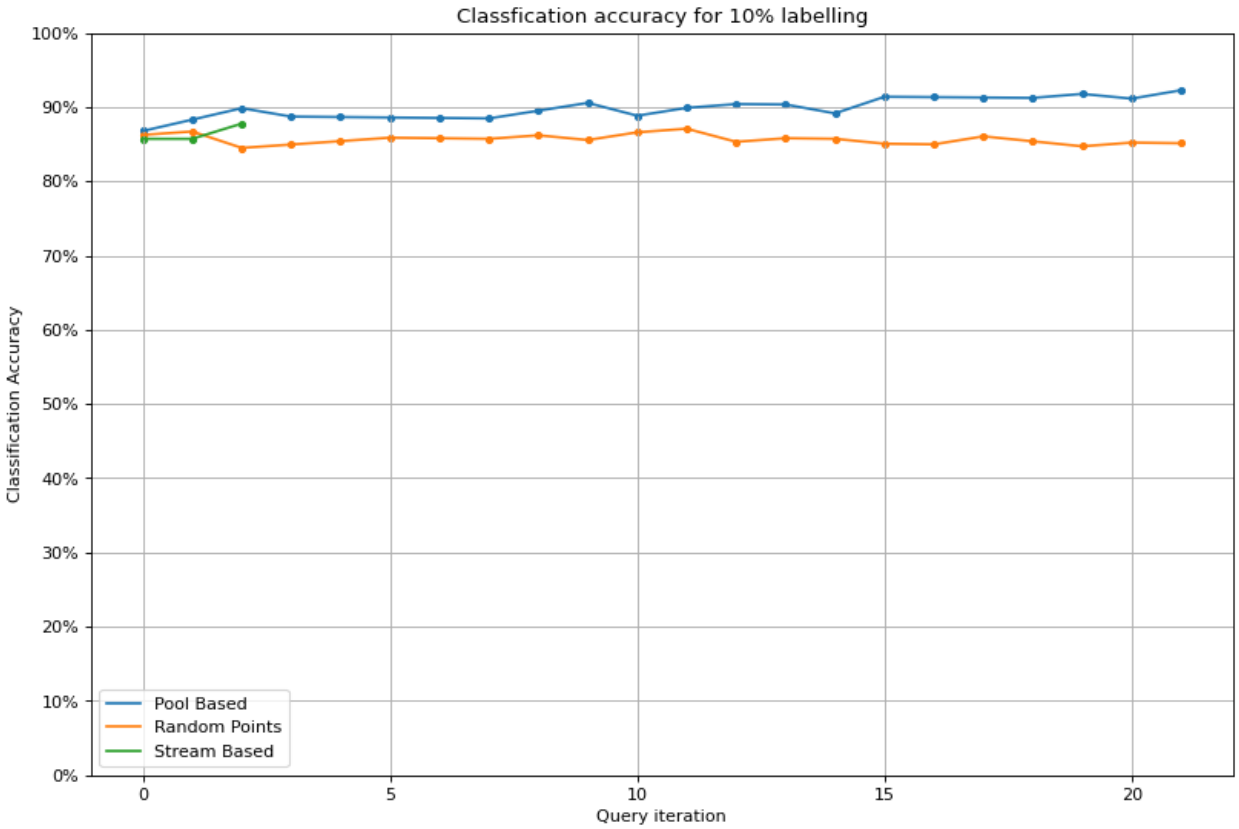
Fig-4

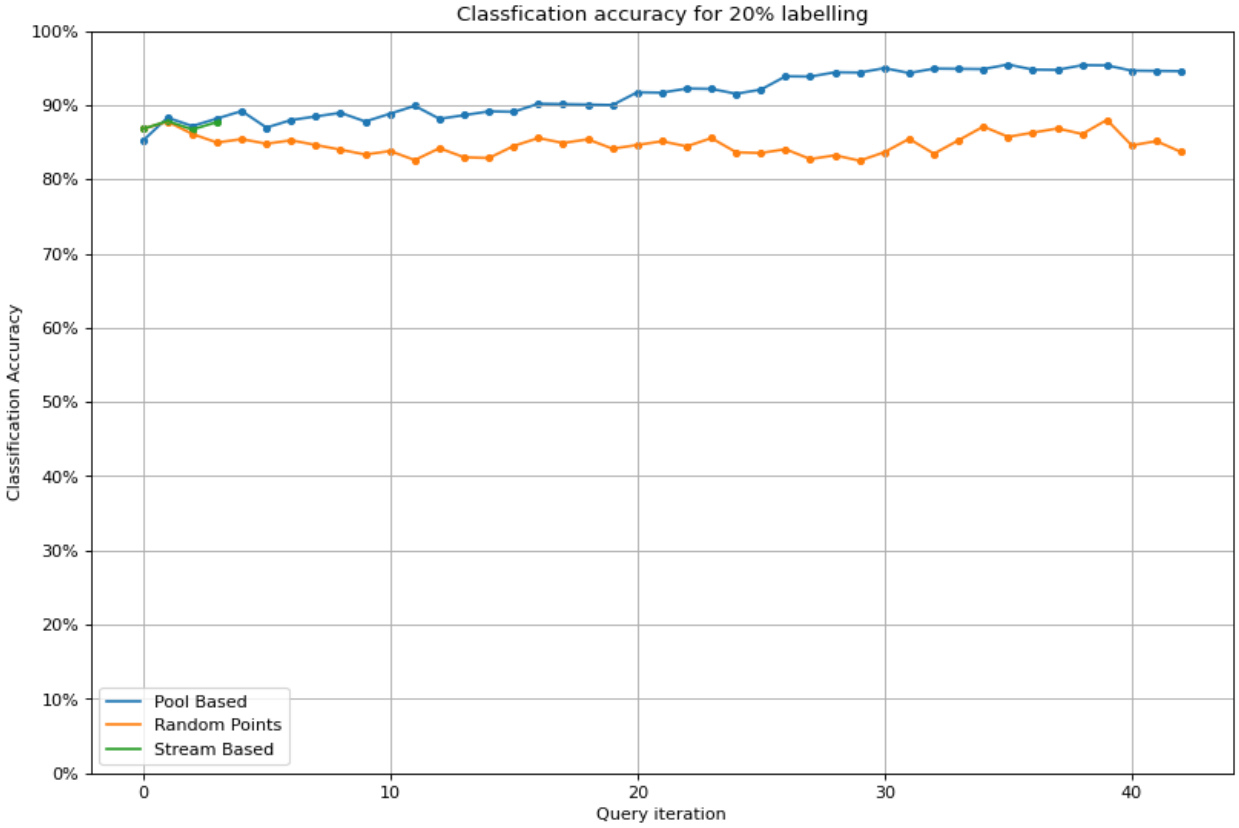
It can be clearly seen that both least confident and entropy-based sampling provided higher accuracy to the model than margin sampling. This is due to the non-smooth nature of margin-sampling which forces the model into local minima.

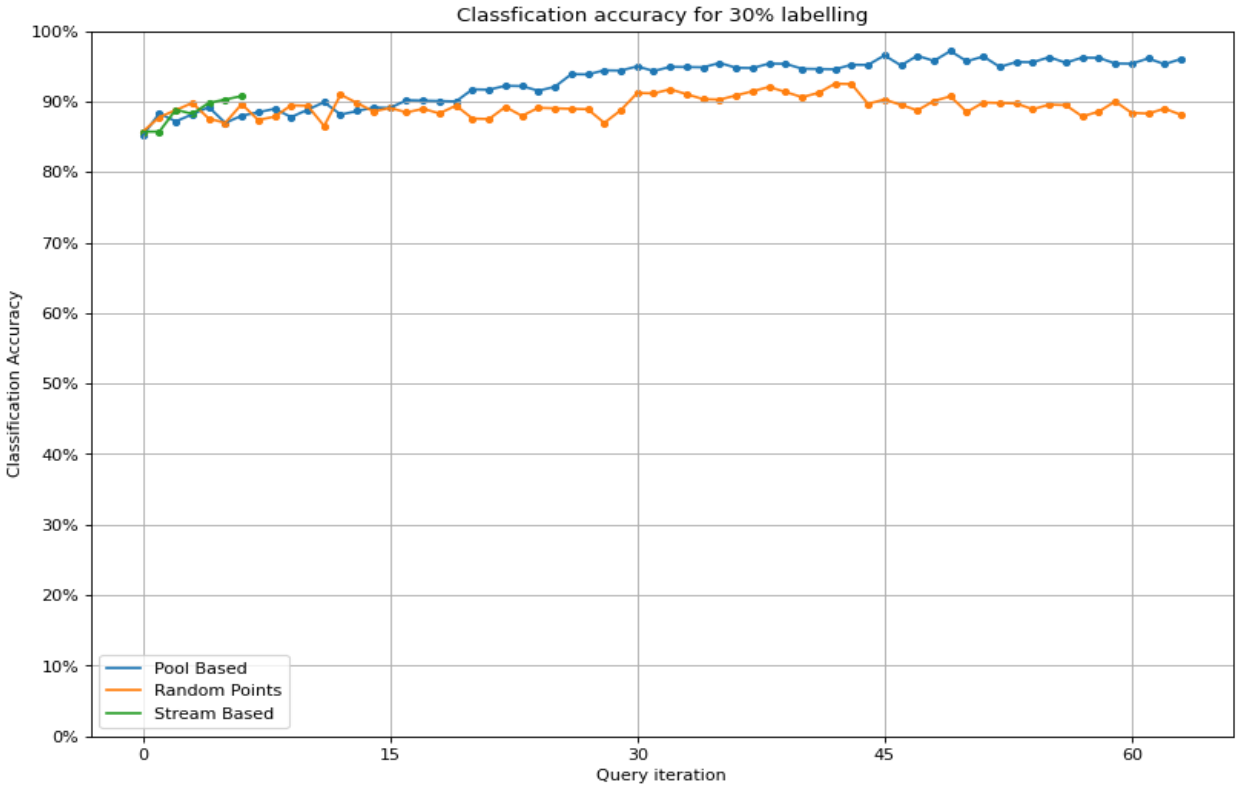
- **Random vs Stream based vs Pool based active learning:**

The data set is trained with random forest classifier for 10% randomly selected labelled models. From 90% unlabelled data 10%, 20%, 30% and 40% randomly selected data is queried to human oracle. Similarly, same amount of data is queried by stream-based method selecting queries with uncertainty>=0.33. Same amount of is queried for pool-based sampling with data point's margin as informative measure. The fig-5 represents classification for each type of active learning model with different volumes of unlabelled data queried.

As can be seen from the figure pool based and stream-based sampling have higher accuracies compared to random sampling because random sampling selects any data point for querying however stream and pool based sampling selects the most informative data point for querying.


Classfication accuracy for 10% labelling
Classification Accuracy
Query iteration
Pool Based
Random Points
Stream Based
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
5
10
15
20


Classfication accuracy for 20% labelling
Classification Accuracy
Query iteration
Pool Based
Random Points
Stream Based
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
10
20
30
40


Classfication accuracy for 30% labelling
Classification Accuracy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
15
30
45
60
Query iteration
Pool Based
Random Points
Stream Based

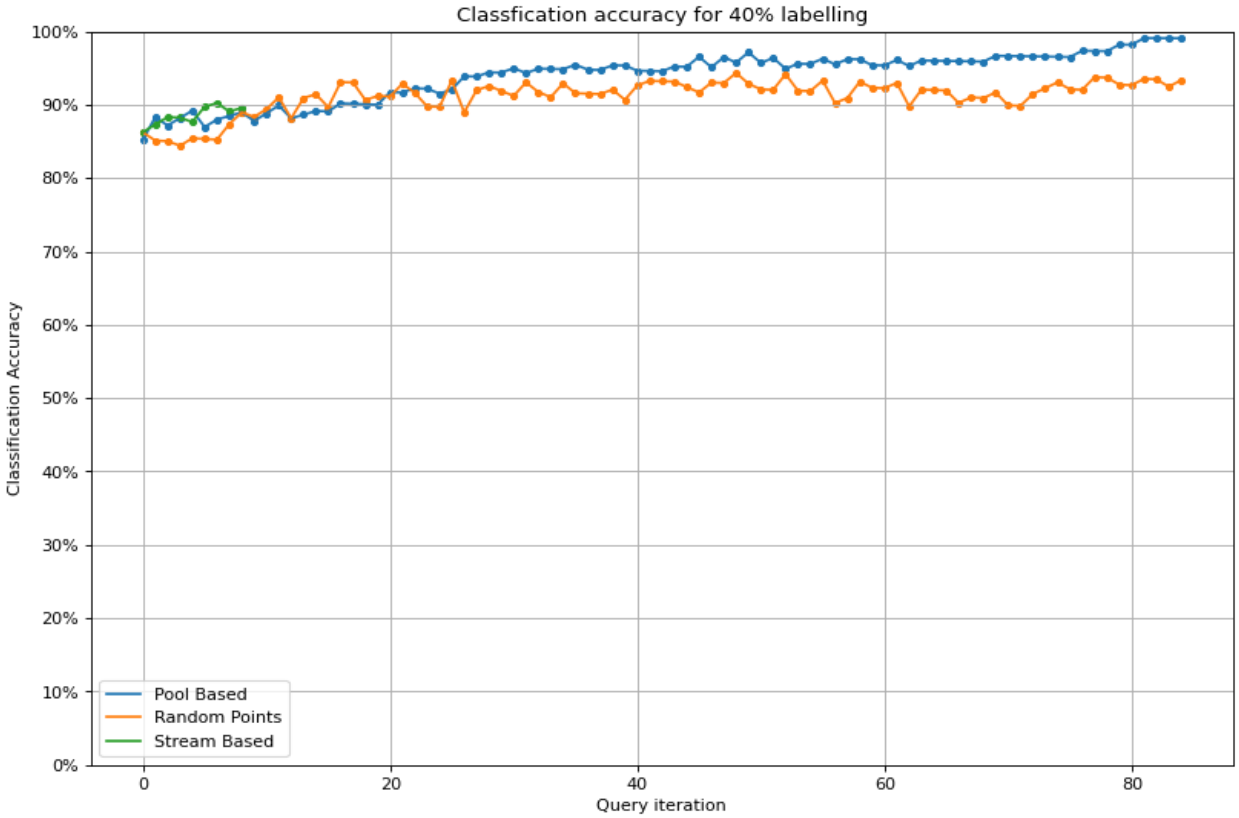

Classfication accuracy for 40% labelling
Classification Accuracy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
20
40
60
80
Query iteration
Pool Based
Random Points
Stream Based

Fig-5

- Query Based Committee:

  The data is initially trained with five random forest classifiers (forming classifier committee) for randomly selected 10% labelled data. The remaining 90% data unlabelled data is ranked based on their informativeness (from classifier committee) using vote entropy and KL divergence. The top 10%, 20%, 30% and 40% informative data points are selected for querying by human oracle. Fig-6 shows the graph of informativeness vs. number of data points queried for vote entropy method and fig-7 shows similar graph but for KL Divergence method.

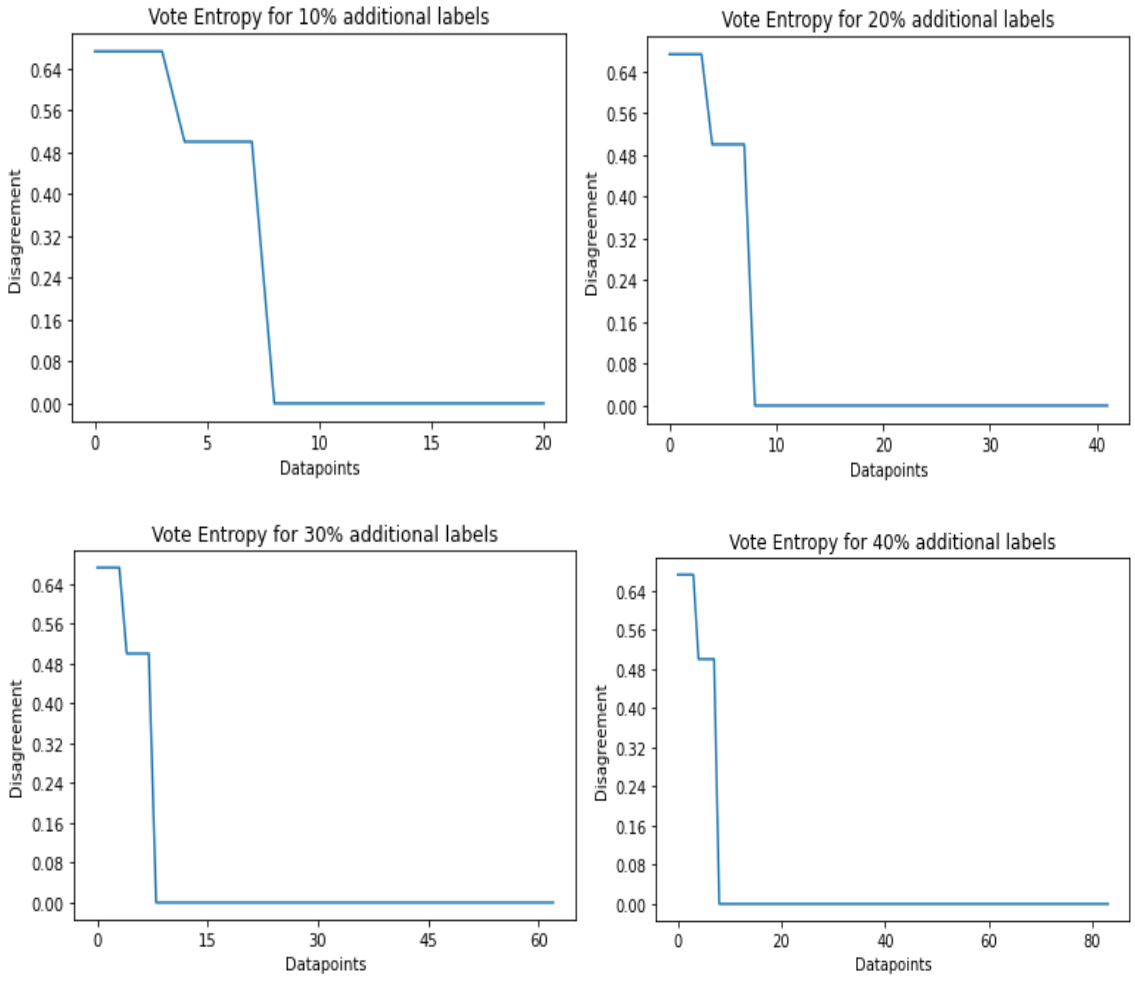


Fig-6

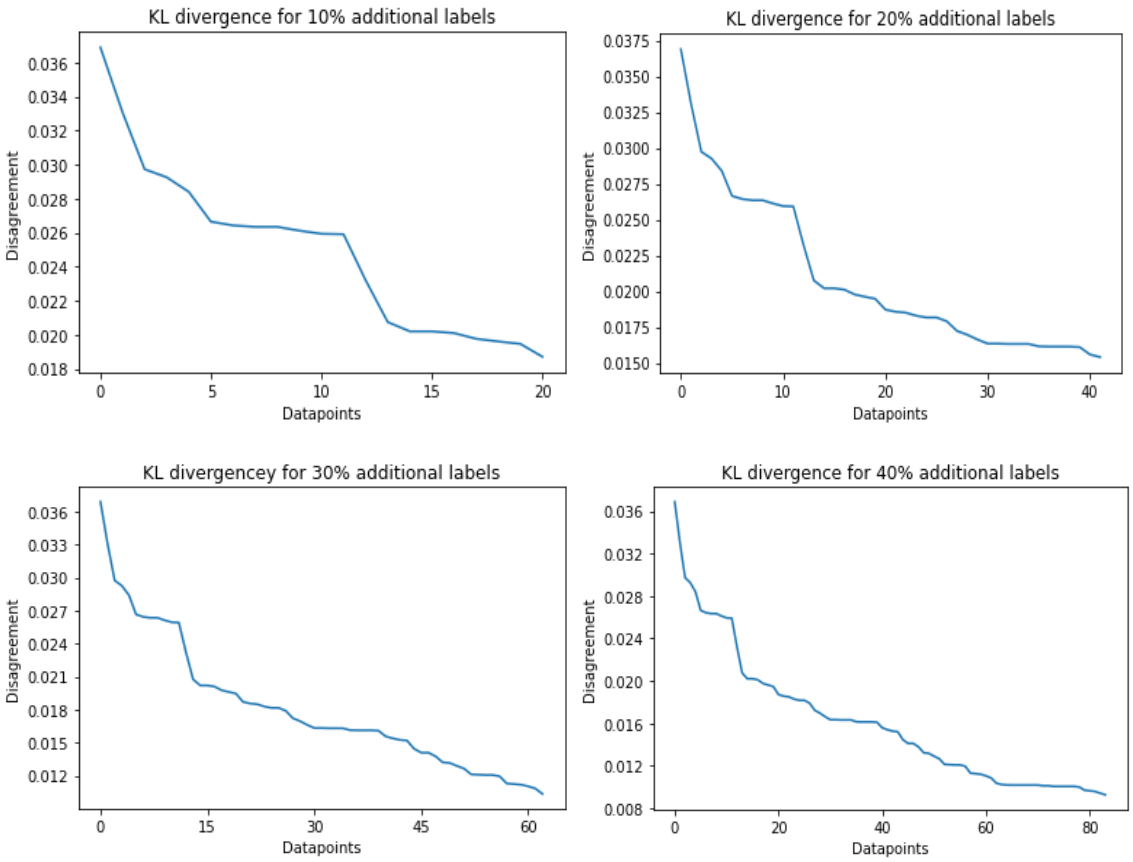
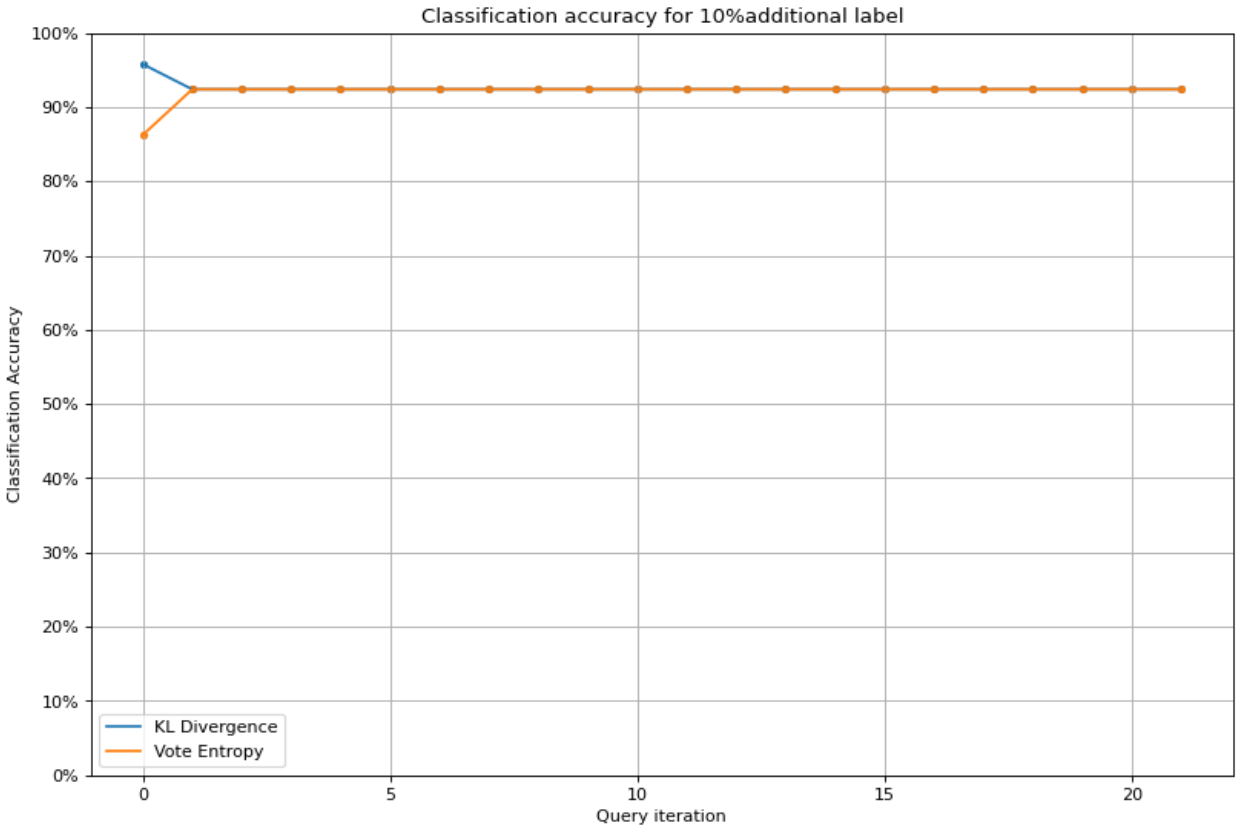
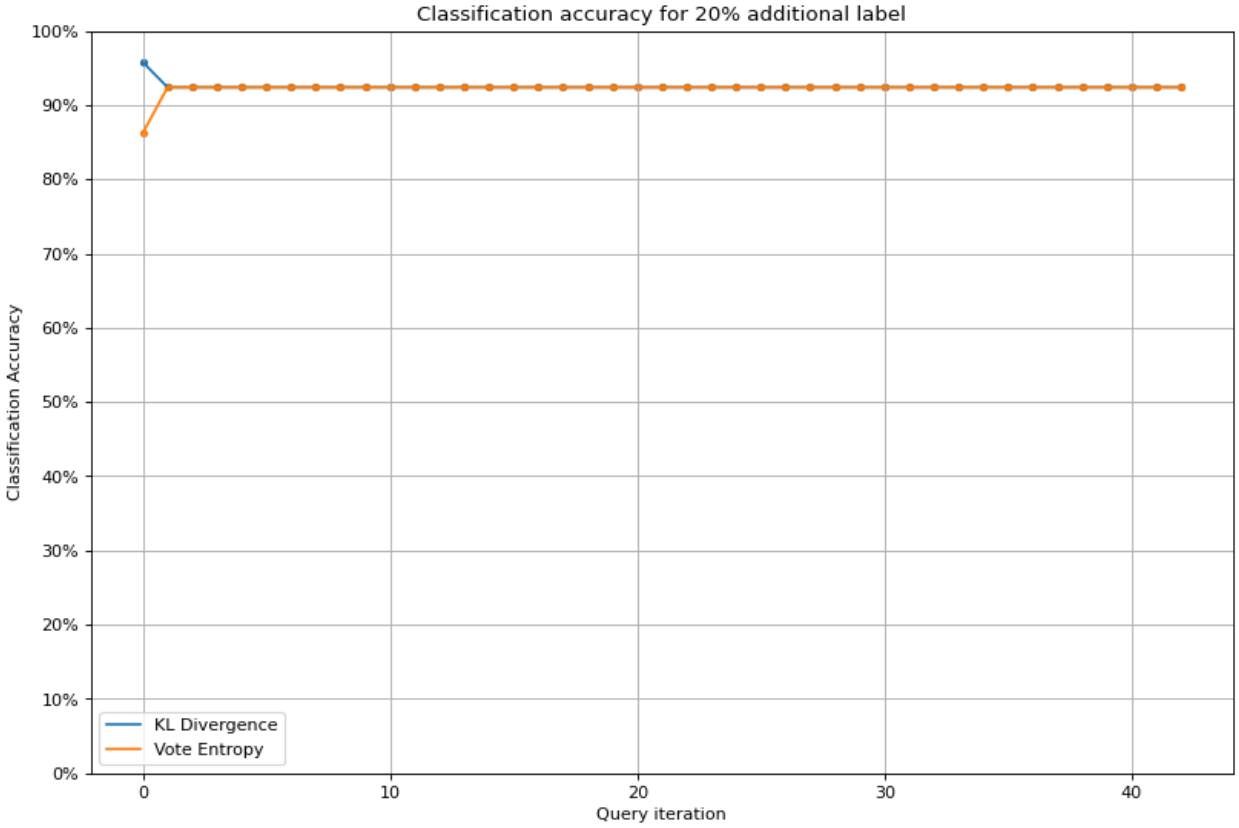
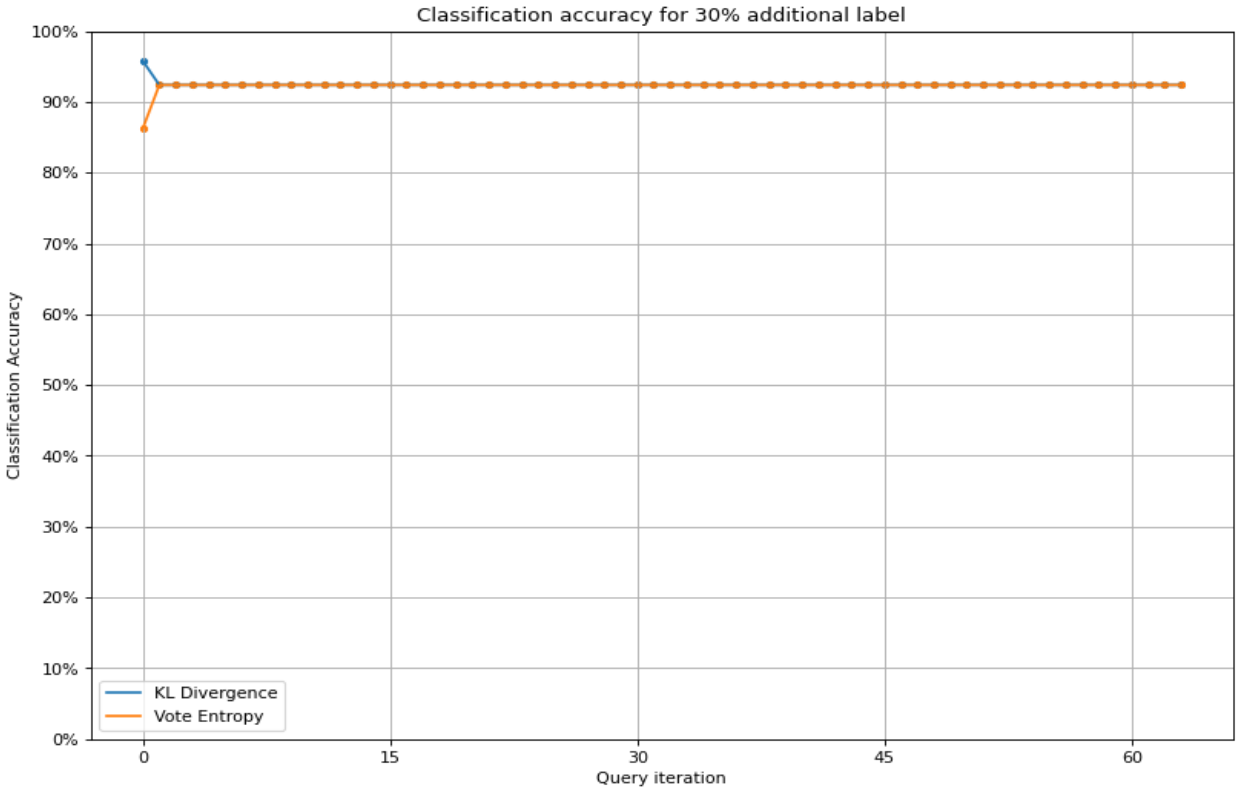


Fig-7

It can be seen that vote entropy produces non-smooth changes in informativeness as opposed from KL-Divergence. This is due to the fact that vote entropy takes into account actual disagreement in a weak way. Fig-8 represents accuracy w.r.t number of points quarried.


Classification accuracy for 10%additional label
Classification Accuracy
Query iteration
KL Divergence
Vote Entropy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
5
10
15
20


Classification accuracy for 20% additional label
Classification Accuracy
Query iteration
KL Divergence
Vote Entropy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0
10
20
30
40


Classification accuracy for 30% additional label
Classification Accuracy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
KL Divergence
Vote Entropy
0
15
30
45
60
Query iteration

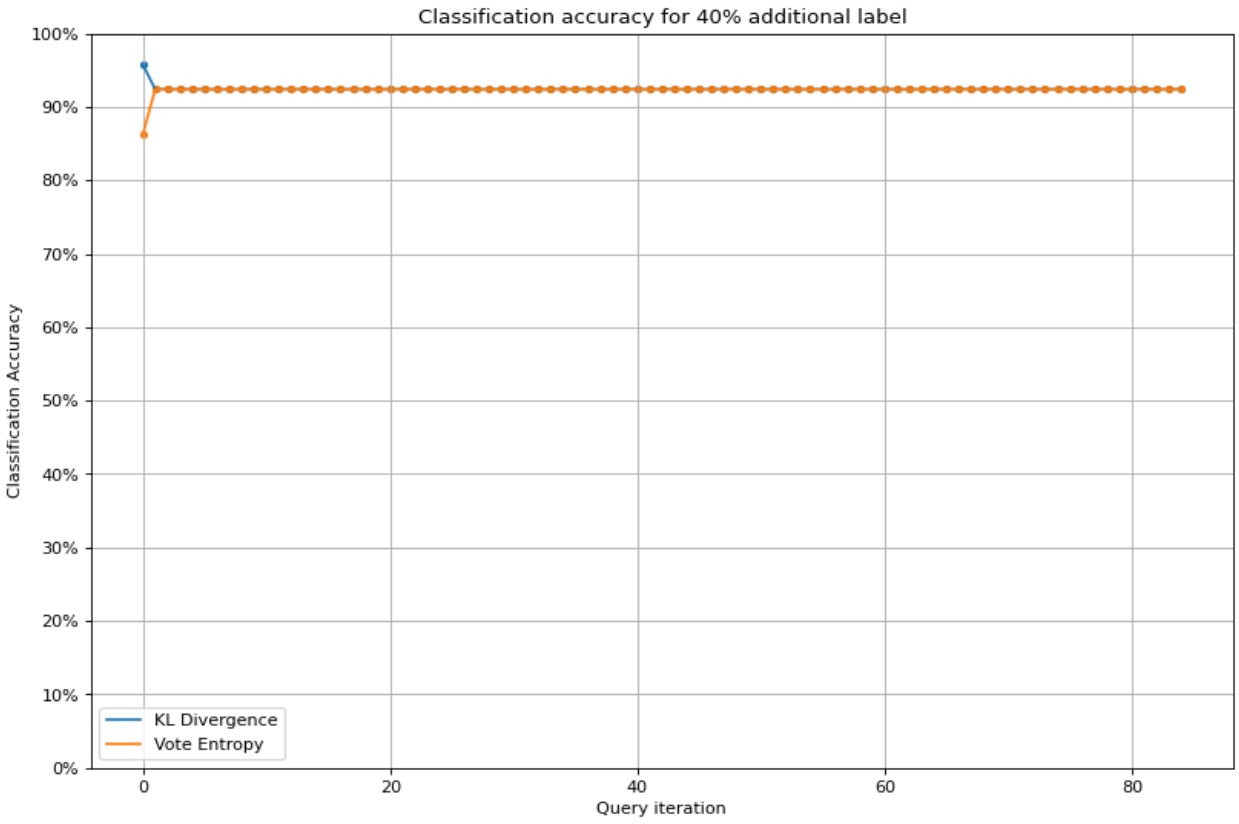

Classification accuracy for 40% additional label
Classification Accuracy
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
KL Divergence
Vote Entropy
0
20
40
60
80
Query iteration

Fig-8

Both KL Divergence and Vote Entropy method converge to same classification accuracy. But generally, KL Divergence should be more preferred as it provides smooth decay in informativeness and increase in accuracy and is closer to actual disagreement amongst committee learners.

- Version Space:

  Version Space represents the vector space where the unlabeled points disagree with the classifier committee predicted class. Fig-9 shows the decrease in version space as we increase the number of instances queried.

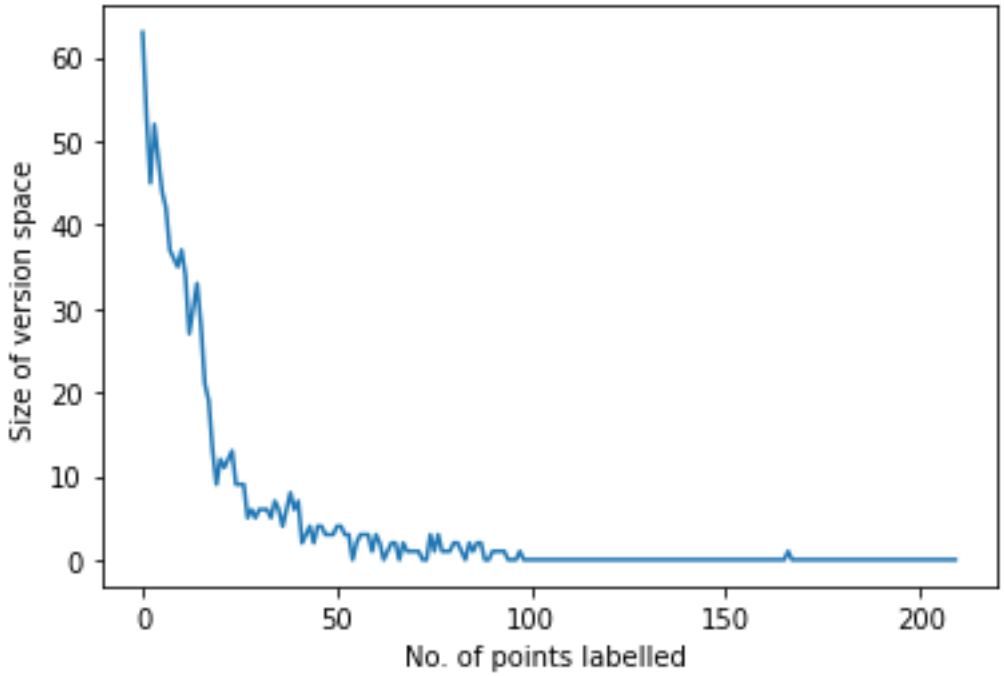


Fig-9

Version space converges to zero with 20% additionally labeled points.

- Active Learning through K-means:

  From 90% unlabeled points 40% (114 data points) points are randomly selected for forming 3 clusters. In each cluster 20% randomly selected points are labelled through oracle, assigning label to each cluster. Hence, total of 22 data points are passed for labelling. The remaining unlabeled

point are passed as test data to identify their class. The accuracy for test data is obtained 94.012%.

If labelling of each point would cost Rs. 100 and 1 hour of time then we have saved Rs.9200 and 92 hours. Hence, percentage decrease in cost and time requirement for labelling is 80.7%.

Code Snippets for Active Learning:

Data Pre-processing:

```
def preprocessing():
data = pd.read_csv("/content/drive/My Drive/Assignments/ML/datacsv/seed_dataset.csv", header=None)
print(data[data.isnull().any(axis=1)])
data = data.sample(frac=1, random_state=11)
#Random sampling to distribute data
data_labels = data.iloc[:,7].astype('category', copy=True)  #data_labels is 7th column are extracted
data = data.iloc[:,:-1]
data = (data-data.mean(axis=0))/data.std(axis=0)
return data_labels, data
if __name__=="main":
   data_labels, data = preprocessing()
   data, data_labels = np.array(data.iloc[:,:]), np.array(data_labels.iloc[:])
   train_len = int(data.shape[0]*0.1)
   X_train, y_train = data[0:train_len,:], data_labels[0:train_len]
   X_unlabld, y_oracles = data[train_len:,:], data_labels[train_len:]
   oracles_len = np.array([data.shape[0]*0.1, data.shape[0]*0.2, data.shape[0]*0.3, data.shape[0]*0.4], dtype=int)
   print("Data for human oracle additional labelling has lengths:", oracles_len[0],',', oracles_len[1],',',
       oracles_len[2],'and', oracles_len[3], 'for 10%, 20%, 30% and 40% labelling respectively')
```

Pool Based Active Learning: (Uncertainty, Margin, Entropy Based):

```
def LCP(ls, oracles_len, X_unlabld, y_unlabld, _learner, values, dicti):
    performance= deepcopy(ls)
    for i in range(oracles_len[0]):
        idx = np.argmax(values)
        X, y = X_unlabld[idx,:].reshape(1, -1), y_oracles[idx].reshape(1,)
        info = values[idx]
        _learner.teach(X=X, y=y)
        X_unlabld, y_oracles, values = np.delete(X_unlabld, idx, axis=0), np.delete(y_oracles, idx, axis=0), np.delete(values, idx, axis=0)
        model_accuracy = _learner.score(X_unlabld, y_oracles)
        print('Accuracy after query:',idx, 'is', model_accuracy, 'for iter:', i)
        performance.append(model_accuracy)
        dict1 = {i:info}
        dicti.update(dict1)
    sort_dict = sorted(dicti.items(), key=lambda x: x[1], reverse=True)
    import collections
    dict_1 = collections.OrderedDict(sort_dict)
    from matplotlib.pyplot import figure
    fig, ax = plt.subplots()
    ax.plot(list(dict_1.values()))
    ax.set_xlabel("Datapoints")
    ax.set_ylabel("Informativeness")
    ax.set_title("Least Confident for 10% additional labels")
    ax.xaxis.set_major_locator(mpl.ticker.MaxNLocator(nbins=5, integer=True))
    ax.yaxis.set_major_locator(mpl.ticker.MaxNLocator(nbins=10))
    plt.show()
    return performance


def poolbased_Learner(X_train, y_train, X_unlabld, y_oracles, oracles_len, strategy):
    classifier = RandomForestClassifier()
    _learner = ActiveLearner(estimator=classifier,
                             query_strategy=strategy,
                             X_training=X_train, y_training=y_train)
    initial_acc = _learner.score(X_unlabld, y_oracles)
    print('Initial prediction accuracy using least confident:', initial_acc)
    values = _learner.query(X_unlabld)
    dicti = {}
    performance10 = LCP([initial_acc], oracles_len, X_unlabld, y_unlabld, _learner, values, dicti)
    performance20 = LCP(performance10, oracles_len, X_unlabld, y_unlabld, _learner, values, dicti)
    performance30 = LCP(performance20, oracles_len, X_unlabld, y_unlabld, _learner, values, dicti)
```

Random vs Stream-based vs Pool-based:

```
def percent_labelling(i, X_unlabld, y_oracles, X_train, y_train, oracles_len, margin_performance10):
    X_unlabld, y_oracles = data[train_len:,:], data_labels[train_len:]
    X_train, y_train = data[0:train_len,:], data_labels[0:train_len]
    Random_learner = ActiveLearner(estimator=classifier,
                                   X_training=X_train, y_training=y_train,
                                   query_strategy = classifier_margin)
    init_acc = Random_learner.score(X_unlabld, y_oracles)
    print('Initial prediction accuracy using margin:', init_acc)
    values = Random_learner.query(X_unlabld)
    # values.sort()
    Random_performance = [init_acc]
    dicti = {}
    for i in range(oracles_len[int((i-10)/10)]):
        random = np.random.randint(low=0, high=X_unlabld.shape[0], size=1)
        X, y = X_unlabld[random[0]].reshape(1,-1), y_oracles[random[0]].reshape(1,)
        Random_learner.teach(X=X, y=y)
        X_unlabld = np.delete(X_unlabld, random[0], axis = 0)
        y_oracles = np.delete(y_oracles, random[0])
        model_accuracy = Random_learner.score(X_unlabld, y_oracles)
        print('Accuracy after query', random[0], model_accuracy)
        Random_performance.append(model_accuracy)

    X_unlabld, y_oracles = data[train_len:,:], data_labels[train_len:]
    X_train, y_train = data[0:train_len,:], data_labels[0:train_len]
    Stream_learner = ActiveLearner(estimator=classifier,
                                   X_training=X_train, y_training=y_train,
                                   query_strategy = classifier_margin)
    init_acc = Stream_learner.score(X_unlabld, y_oracles)
    Stream_performance = [init_acc]

    for index in range(oracles_len[int((i-10)/10)]):
        stream_idx = np.random.choice(range(len(X_unlabld)))
        if classifier_uncertainty(Stream_learner, X_unlabld[stream_idx].reshape(1, -1)) >= 0.33:
            Stream_learner.teach(X_unlabld[stream_idx].reshape(1, -1), y_oracles[stream_idx].reshape(1, ))
            stream_acc = Stream_learner.score(X_unlabld, y_oracles)
            X_unlabld = np.delete(X_unlabld, stream_idx, axis = 0)
            y_oracles = np.delete(y_oracles, stream_idx, axis = 0)
            Stream_performance.append(stream_acc)
            print('Accuracy after stream based query on:', stream_idx, 'is', stream_acc)
```

Query Based Committee:

```
def LCP(ls, oracles_len, X_unlabld, y_unlabld, _learner, dicti):
    performance= deepcopy(ls)
    for i in range(oracles_len[0]):
        idx = np.argmax(values)
        X, y = X_unlabld[idx,:].reshape(1, -1), y_oracles[idx].reshape(1,)
        info = values[idx]
        _learner.teach(X=X, y=y)
        X_unlabld, y_oracles, values = np.delete(X_unlabld, idx, axis=0), np.delete(y_oracles, idx, axis=0), np.delete(values, idx, axis=0)
        model_accuracy = _learner.score(X_unlabld, y_oracles)
        print('Accuracy after query:',idx, 'is', model_accuracy, 'for iter:', i)
        performance.append(model_accuracy)
        dict1 = {i:info}
        dicti.update(dict1)
    sort_dict = sorted(dicti.items(), key=lambda x: x[1], reverse=True)
    import collections
    dict_1 = collections.OrderedDict(sort_dict)
    from matplotlib.pyplot import figure
    fig, ax = plt.subplots()
    ax.plot(list(dict_1.values()))
    ax.set_xlabel("Datapoints")
    ax.set_ylabel("Informativeness")
    ax.set_title("Least Confident for 10% additional labels")
    ax.xaxis.set_major_locator(mpl.ticker.MaxNLocator(nbins=5, integer=True))
    ax.yaxis.set_major_locator(mpl.ticker.MaxNLocator(nbins=10))
    plt.show()
    return performance


def poolbased_Learner(X_train, y_train, X_unlabld, y_oracles, oracles_len, strategy):
    n_committe = 5;
    committee = [];
    for i in range(n_committe):
        learner = ActiveLearner(estimator=RandomForestClassifier(), X_training=X_train, y_training=y_train)
        committee.append(learner)
    committee_ = Committee(learner_list=committee,
                           query_strategy=strategy)


    unlabld_entropyScore = committee_entropy.score(X_unlabld, y_oracles)
    print('Initial prediction accuracy using vote entropy', unlabld_entropyScore)
    Performance10 = [unlabld_entropyScore]
```

Version Space:

```
def version_space(data, data_labels, train_len):
    # initializing Committee members
    n_committee = 5
    committee = []
    for i in range(n_committee):
        # initial training data
        n_init = round(len(data)*0.1)
        X_train = data[i*n_init:(i+1)*n_init, :]
        y_train = data_labels[i*n_init:(i+1)*n_init]
        # creating a reduced copy of the data with the known instances removed
        X_unlabld = data[(i+1)*n_init:, :]
        y_oracles = data_labels[(i+1)*n_init:]
        # initializing learner
        learner = ActiveLearner(
            estimator=RandomForestClassifier(),
            X_training=X_train, y_training = y_train
        )
        committee.append(learner)
    # assembling the committee
    committee_KL = Committee(learner_list=committee,
                    query_strategy=KL_max_disagreement)
    committee_entropy = Committee(learner_list=committee,
                    query_strategy=vote_entropy_sampling)
    comittee_version_space_KL = Committee(learner_list=committee,
                    query_strategy=KL_max_disagreement)
    version_space_size = []
    version_space_points = vote_entropy(comittee_version_space_KL, data)
    version_space_points = version_space_points[version_space_points!=0]
    print('Intial Size of version space is ', len(version_space_points))
    performance_history_KL_10 = [committee_KL.score(data, data_labels)]
    X_unlabld = data[:, :]
    y_oracles = data_labels[:]
    # query by committee
    for i in range(len(data)):
```

```
# print(i)
    query_idx,_ = max_disagreement_sampling(comittee_version_space_KL, X_unlabld)
  #   query_instance = committee_KL.query(X_pool)
  #   query_idx = query_instance.argmax()
    query_idx = np.array((query_idx))
    comittee_version_space_KL.teach(
       X = X_unlabld[query_idx].reshape(1, -1),
       y = y_oracles[query_idx].reshape(1, )
    )
    performance_history_KL_10.append(comittee_version_space_KL.score(data, data_labels))
    # remove queried instance from pool
    X_unlabld = np.delete(X_unlabld, query_idx, axis = 0)
    y_oracles = np.delete(y_oracles, query_idx)
    version_space_points = vote_entropy(comittee_version_space_KL, data)
    version_space_points = version_space_points[version_space_points!=0]
    version_space_size.append(version_space_points.shape)
    plt.plot(version_space_size)
    plt.xlabel('No. of points labelled ')
    plt.ylabel('Size of version space')
    plt.show()
if __name__="main":
  version_space(data, data_labels, train_len)
```

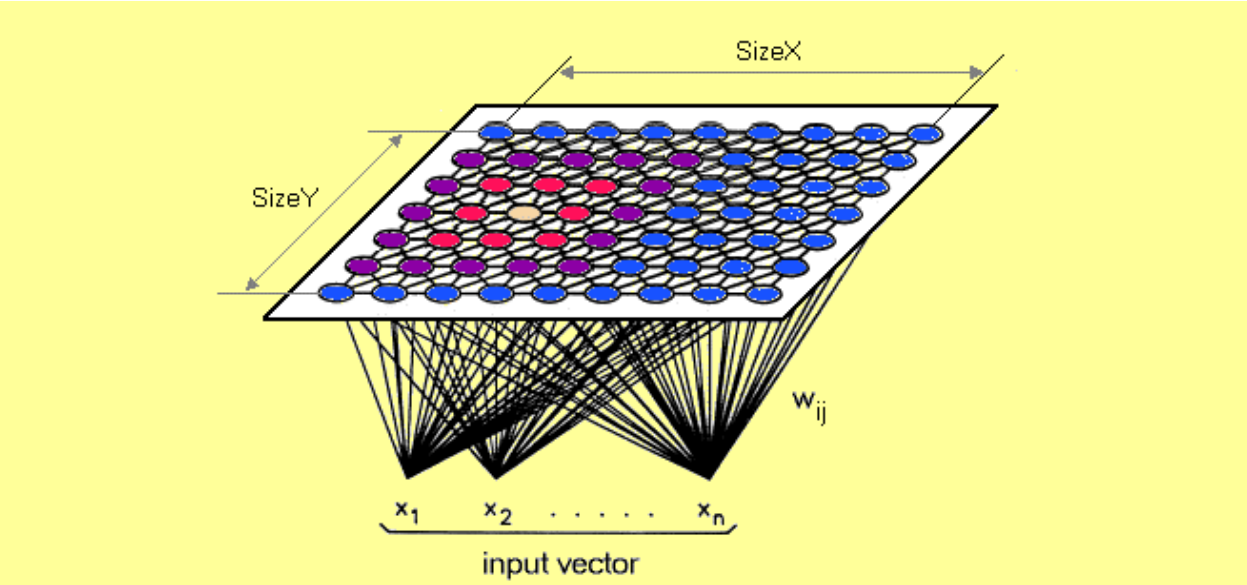
K-means:

```
def k_means(data, data_labels, train_len):
    X_unlabld = data[train_len:,:]
    y_oracles = data_labels[train_len:]
    for_clusters = int(len(X_unlabld)*0.4)
    X_forclusters = X_unlabld[for_clusters:,:]
    y_forclusters = y_oracles[for_clusters:]
    K = 3
    kmeans = KMeans(n_clusters=K, random_state=1).fit(X_forclusters)
    labels = kmeans.labels_
    index = []
    for i in range(K):
        index.append(np.argwhere(labels==i))
    index = np.array(index)
    count = 0
    X_test = deepcopy(data[train_len:,:])
    y_test = deepcopy(data_labels[train_len:])
    cluster_class = {}
    for i in range(K):
        label_pts = round(index[i].shape[0]*0.2)
        label_idx = index[i][np.random.choice(range(index[i].shape[0]), size = label_pts, replace=False)]
        count = count + label_idx.shape[0]
        cluster_class[i] = stats.mode(y_forclusters[label_idx])[0]
        X_test = np.delete(X_test, label_idx.reshape(1,-1)[0], axis=0)
        y_test = np.delete(y_test, label_idx.reshape(1,-1)[0], axis=0)
    y_test_labels = kmeans.predict(X_test)
    error = 0
    for i in range(X_test.shape[0]):
        if cluster_class[y_test_labels[i]]!=y_test[i]:
            error = error + 1
    print('KMeans accuracy = ', 1 - error/X_test.shape[0], 'for', count, 'number of points labelled as 20% instead of all',
X_forclusters.shape[0])


if __name__=="main":
    k_means(data, data_labels, train_len)
```

Self -Organizing Maps

A self-organizing map (or Kohenin's map) is a type of artificial neural network that is trained using unsupervised learning to produce a low-dimensional, discretized representation of the input space of the training samples. SOMs map multidimensional data onto lower dimensional subspaces where geometric relationships between points indicate their similarity.



SOM Architecture :

• Two layers of neurons

• Input layer

• Output map layer

• Each output neuron is connected to each input neuron

Each neuron in a SOM is assigned a weight vector with the same dimensionality d as the input space.

SOM uses competitive learning to update its weights. We compute distance between each neuron (neuron from the output layer) and the input data, and the neuron with the lowest distance will be the winner of the competition.

Relative ordering of inputs is preserved by the ordering in the neurons such that:

– **Close neurons represent inputs that are "similar"**

– Far neurons represent inputs that are far apart

Understanding the dataset:

1. Number of Attributes: 18
2. Number of Instances: 101
3. Attribute Information: (name of attribute and type of value domain):

Animal name:	Unique for each instance
Hair	Boolean
Feathers	Boolean
Eggs	Boolean
Milk	Boolean
Airborne	Boolean
Aquatic	Boolean
Predator	Boolean
Toothed	Boolean
Backbone	Boolean
Breathes	Boolean
Venomous	Boolean
Fins	Boolean

Legs	Numeric (set of values: {0,2,4,5,6,8})
Tail	Boolean
Domestic	Boolean
Catsize	Boolean
Type	Numeric (integer values in range: [1,7])

“Type” attribute is the class of animals.

Class division of all the animals-

Class 1 (41) aardvark, antelope, bear, boar, buffalo, calf,
cavy, cheetah, deer, dolphin, elephant,
fruitbat, giraffe, girl, goat, gorilla, hamster,
hare, leopard, lion, lynx, mink, mole, mongoose,
opossum, oryx, platypus, polecat, pony,
porpoise, puma, pussycat, raccoon, reindeer,
seal, sealion, squirrel, vampire, vole, wallaby,wolf

Class 2 (20) chicken, crow, dove, duck, flamingo, gull, hawk,
kiwi, lark, ostrich, parakeet, penguin, pheasant,
rhea, skimmer, skua, sparrow, swan, vulture, wren

Class 3 (5) pitviper, seasnake, slowworm, tortoise, tuatara

Class 4 (13) bass, carp, catfish, chub, dogfish, haddock,

herring, pike, piranha, seahorse, sole, stingray, tuna

Class 5 (4) frog, frog, newt, toad

Class 6 (8) flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp

Class 7 (10) clam, crab, crayfish, lobster, octopus,

scorpion, seawasp, slug, starfish, worm

Code Snippets-

```
from minisom import MiniSom
```

We imported the **minisom** *library*

```
som = MiniSom(29,29, 16, sigma=1.5, learning_rate=.7,
activation_distance='euclidean',
topology='hexagonal', neighborhood_function='gaussian', random_seed=21)
som.train_batch(data, 1000, verbose=True)
```

The above code is used to train the Neural Network. It is trained on the 'data' dataset. And runs for 1000 epochs.

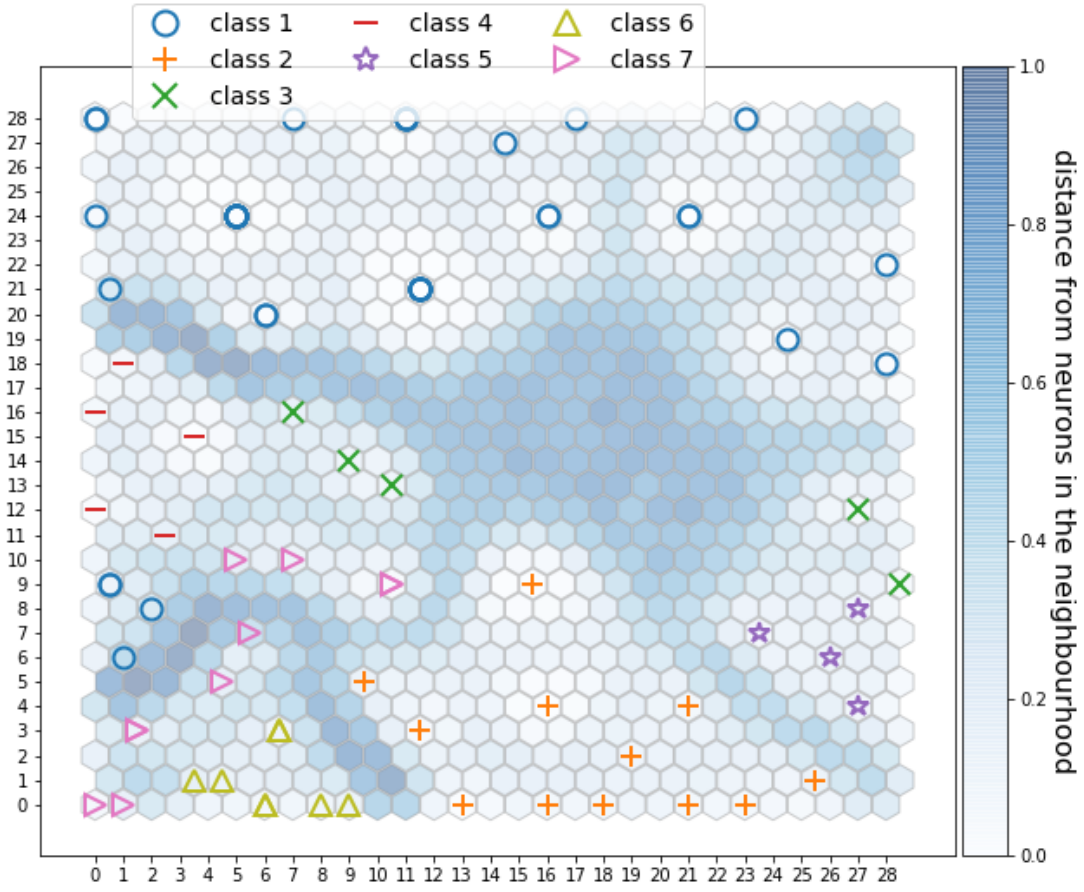
activation_distance='euclidean' - we use euclidean distance between the inputs and the neurons to find out the winning neuron.

topology='hexagonal - hexagonal topology was selected

neighborhood_function='gaussian'-it is used to determine the neighbours of the winning neuron.

Analysis:

Aim: To create a 2-dimensional visualization using self-organizing maps, from the input features. And see which class of animal is similar to which other class of animal. (Neurons which are closer are more similar)



Understanding the figure-

Darker hexagon color means that there is more separation between the hexagonal blocks (more separation between classes of animals).

Some observations drawn from the image:

We will use the fact that close neurons represent inputs that are “similar”.

1. Class 3 and class 5 are very close to each other -

They have a few similarities like both lay eggs(column 4), breathe(column 11). Some animals in class 3 and 5 can live in water as well. So overall, they do have some similarities.

2. Class 2 (Birds) do not have many things in common with other reptiles, mammals with respect to these attributes. Therefore they can be seen at a difference from the other classes.

Result-

We can draw conclusions from the SOM that which class of animal is more similar to which other class of animal.