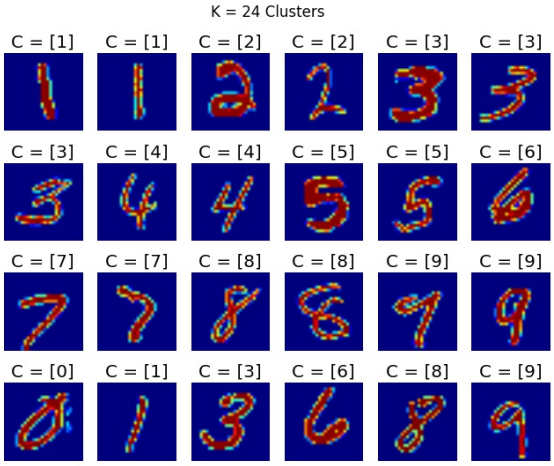
HW4: Decision Trees & K-Nearest Neighbor

In this exercise we will explore Decision Trees (DT) and K-Nearest Neighbor (KNN) classifiers. Most importantly, we will study how to select the optimal regularization parameters of a learner.

For this exercise we will explore two different datasets, hand-written digits and newsgroup text. The MNIST dataset is comprised of 28x28 pixel images of hand-written digits from the U.S. Census Bureau. Each instance is made up of 784 features (28 x 28 pixel images), each representing a pixel color value between 0 and 1. There are 10 classes to classify, digits 0-9. Here is an example of the hand-written digits you will classify.



The 20 newsgroup dataset is comprised of class 0 (“comp.graphics”) and class 1 (“comp.windows.x”) new wire segments. Some examples are:

y=0 articl repli program work comput distribut system mail code softwar applic inform call internet fax point includ group number address type center id research current video usa develop engin david robert page design accept author contact gmt

y=0 nntp write program file graphic distribut time code find softwar call thing point d sourc includ give function ftp librari inc draw suggest lot wrote site implement stuff access handl object book place year fast output exist routin document robert limit ad pretti easi mac level

y=1 nntp nt articl program find ve make internet server point motif interest gener manag start inc type widget client lot id expolcsmitedu case put result memori xpert handl event back place termin long network design perform great

y=1 nntp write nt window distribut mail softwar ca motif unix manag usa engin david handl object ms design xt

The features are 2997 distinct words (bag of words model) where each news wire case’s feature space would be the words that are found in case. (word present = 1, word absent = 0).

HW4: Decision Trees & K-Nearest Neighbor

A great deal of the work has already been done for you! All you need to do is implement the DT and KNN learning and prediction algorithms. We will test our learners using 5-fold cross validation because it is fast but you know in practice you should use 10-fold cross-validation. The cross-validation code is already written and provided.

To get you started several PyDev modules have been provided.

Run.py – This module is your main control program that trains and evaluates the models you will be generating. This has been completed for you so there is nothing to do in this module except understand it and run it.

DrawDigits.py – This is a utility module to draw digit graphs. Everything is complete with this module.

Tree.py – This class will represent you decision tree. Your algorithm will construct a Tree instance representing your learned decision tree model. There is nothing that will need to be added to this class, it is complete.

DT.py – This class is where you will implement the Decision Tree algorithm we have discussed in class. You will implement 3 methods, *entropy, construct, and predict* according to the pseudo code provided below. Your *construct* and *predict* methods will be implemented using recursion.

KNN.py – This class is where you will implement the K-Nearest Neighbor algorithm we have discussed in class. You will only need to implement one method in this class called *predict*.

Hand-in all code along with a pdf document with the answers and graphs to the questions below. All submitted as one zip file.

HW4: Decision Trees & K-Nearest Neighbor

Decision Tree Pseudo Code

1. Entropy

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i|t) log_2 p(i|t) \qquad \textit{Remember: } 0\ log_2 0 = 0$$

2. Construct

```
Function Construct(X, Y, threshold)
    (a) tree = Tree()
    (b) if (all the labels are the same) or (threshold >= number of training cases)
        then        /* leaf node */
                (a) set node to a leaf node
                (b) set label to majority label among training cases
        else /* branch  node*/
                (a) set node to a branch node
                (b) calculate impurity cost for all features over all training cases J(f, tr_f) = sum_{t=1}^{N} m_t/m * I_t
                (c) select the one feature with the minimum impurity
                (d) set node feature (split) to this feature position
                (e) split the training case using the selected feature with the
                    self.split_threshold. (left: X_S0 <= 0.5 right: X_S1 > 0.5)
                (f) tree.left = Construct(X_SO(data), X_SO(labels), threshold)
                (g) tree.right = Construct(X_S1(data), X_S1(labels), threshold)
        endif
    (c) return tree
```

3. Predict

```
Function Predict(tree, X)
        if (tree is a leaf node)
        then
                (a) (ŷ) = tree.label
        else /*
                (a) if  (X[split feature] <= self.split_threshold)
                    then    /* recurse on left branch */
                                (a) ŷ = Predict(tree.left, X)
                    else    /*  recurse on right branch
                                (b) ŷ = Predict(tree.right, X)
                    endif
        endif
        return ŷ
```

HW4: Decision Trees & K-Nearest Neighbor

KNN Pseudo Code

Euclidian Distance $\boldsymbol{d(x,\hat{x}) = \sqrt{\sum_{i=1}^{d}(x_i - \hat{x}_i)^2} = \sqrt{(x - \hat{x})(x - \hat{x})^T}}$

1) Calculate $\forall\ d(x,\hat{x}): tr(x,y) \in D$
2) Select $D_k \subseteq D, the\ set\ of\ k\ closest\ training\ examples\ to\ \hat{x}\ (i.e \min d(x,\hat{x}))$
3) $\hat{y} = \underset{c}{argmax} \sum_{tr(x_i,y_i)\in D_k} I(c = y_i)\ , choose\ the\ majority\ class\ from\ D_k$

Regularization

The Decision Tree regularization is controlled by determining how deep we will grow our tree models. At any point in the construction process, the first thing we check before growing any deeper is weather the number of unclassified training cases remaining N is less than a certain threshold. By varying the threshold, we control the depth of our Decision Tree model.

The KNN regularization is controlled by how many neighbors will participate in the vote.

Task1: Implement you Decision Tree classifier.

Task2: Implement your K-Nearest Neighbor classifier.

Task3: Analysis of Results

(a) Using the graphs that are produced, determine the optimal regularization parameter value for your KNN classifier. Explain why you think it is the optimal value?

(b) Using the graphs that are produced, determine the optimal regularization parameter value for your DT classifier. Explain why you think it is the optimal value?

(c) For the DT classifier, look at the error rate graphs (MNIST & 20NG) and determine if you see evidence of overfitting or underfitting? If so, for which regularization settings? Explain if these results make sense or if they seem odd.

(d) For the KNN classifier, look at the error rate graphs (MNIST & 20NG) and determine if you see evidence of overfitting or underfitting? If so, for which regularization settings? Explain if these results make sense or if they seem odd.

(e) Compare the results of the DT and the KNN for the MNSIT classification task. Which classifier and regularization parameter would you choose as the best?

(f) Compare the results of the DT and the KNN for the 20NG classification task. Which classifier and regularization parameter would you choose as the best?

HW4: Decision Trees & K-Nearest Neighbor

What to Hand in

You should hand in a *.pdf containing all of the necessary writeup items along with your evaluation results and graphs. Also hand in your DT.py and KNN.py implementations. If you implemented anything in helper functions in separate files, include those too. Basically, I should be able to run my version of run with your implementations. Zip this all up.