Machine Learning & Discovery *ECS 271, Fall 2024*

HW1: Decision trees and KNN

Please note that you have to submit all of your codes and results with a PDF file containing the answer to the questions, plots, and reference to the files you used or programmed for a question in zip file. For your PDF, we encourage using LaTeX to produce your writeups. You'll need *mydefs.sty* and *notes.sty* which can be downloaded from the course page. You can use overleaf.com to write your latex file and save the result as PDF.

Important:

1. This is a one-person exercise. You are more than welcome to reach out to TA or your friends and teammates to seek help with questions, but plagiarism in any form will not be tolerated.
2. Using ChatGPT or any other large language model in answering these questions is against the Code of Conduct and will be reported.
3. For this particular homework, you are not allowed to use any external machine learning libraries in python, including but not limited to: PyTorch, Tensorflow, Scikit, etc. You are allowed to use numpy.
4. Your codes should run in under 5 minutes before producing the results.
5. In case you are asked to use a seed for your random generator, use the last 3-digits of your student ID number

1 Math Practice (NOT GRADED)

This is a simple math exercise to make sure you are familiar with the mathematical background required in this course. **This question is not graded, so in case you are not confident in answering any of these questions in less than a minute, let us know**. In the following exercise, you need to check if a statement is true or false. If you couldn't answer at least **8**, you probably need to take some extra time outside the class to beefing up your math.

1. $\frac{\partial}{\partial x} \log x = -\frac{1}{x}$
2. $p(a, b|c) = \frac{p(a|c)p(b|c)}{p(c)}$
3. $p(a \mid b) = p(a, b)/p(b)$
4. $\log x + \log y = \log(xy)$
5. $p(x \mid y, z) = p(x \mid y)p(x \mid z)$
6. $\int_{-\infty}^{\infty} \mathrm{d}x \exp[-(\pi/2)x^2] = \sqrt{2}$
7. $\log[ab^c] = \log a + (\log b)(\log c)$

8. $\frac{\partial}{\partial x}\sigma(x) = \sigma(x) \times (1 - \sigma(x))$ where $\sigma(x) = 1/(1 + e^{-x})$

9. The distance between the point (x_1, y_1) and line $ax + by + c$ is $(ax_1 + by_1 + c)/\sqrt{a^2 + b^2}$

10. $\left|\boldsymbol{u}^\top \boldsymbol{v}\right| \geq ||\boldsymbol{u}|| \times ||\boldsymbol{v}||$, where $|\cdot|$ denotes absolute value and $\boldsymbol{u}^\top \boldsymbol{v}$ is the dot product of $\boldsymbol{u}$ and $\boldsymbol{v}$

11. $C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$, where $C(n, k)$ is the number of ways of choosing k objects from n

12. $||\alpha\boldsymbol{u} + \boldsymbol{v}||^2 = \alpha^2 ||\boldsymbol{u}||^2 + ||\boldsymbol{v}||^2$, where $||\cdot||$ denotes Euclidean norm, α is a scalar and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors

2 Continuous Variable vs Discrete Variables

All the attributes we observed in the class so far for decision tree are binary like "Is it Morning?". Now lets assume that our attributes are numerical.

1. Let's say we want to consider the **Time** on which the class begins. One can argue that we can pick a threshold τ and use (Time $< \tau$)? as a criteria to split the data in two and make a binary tree. Explain how you might pick the optimal value of τ.

2. In the decision tree learning algorithm discussed in class, once a binary attribute is used, the subtrees do not need to consider it anymore. Explain why in using continuous attributes this may not be the case.

My answer for Question 1

In a decision tree, the goal is to make splits that are useful for future predictions. In this case, using a threshold τ to split the dataset into two categories is the objective. The optimal threshold can be selected by following these steps:

A. Sort the dataset by time (either in ascending or descending order).
B. Iterate through the potential thresholds, which are the unique time values.
C. Split the dataset at each selected threshold into two parts.
D. Calculate the variance of the two parts, Var_1 and Var_2.
E. Compute the total variance as the weighted sum of the variances of the two parts:

$$\text{Total Variance} = \frac{|Size_1|}{|Size_{\text{total}}|} \cdot \text{Var}_1 + \frac{|Size_2|}{|Size_{\text{total}}|} \cdot \text{Var}_2$$

F. Select the threshold that minimizes the total variance as the optimal threshold.

In this way, the threshold separates the time into two categories, ensuring that the variance within each category is minimized.

My answer for Question 2

Binary attribute only have two values there. When we split by this attribute into two, we already go through all the possible values there. However, for the continuous attributes, they have more than two values, which make them impossible to split up within a single time. After a single split, there might be other important points or threshold need to be evaluated. Hence, continuous attribute is more complex than binary attribute since we need to split at multiple points to create effective splits.

3 To Memorize or Not to Memorize:

Why using the training data as a dictionary or lookup table and referring to it is a bad strategy for learning? What is it called? How do we prevent it?

Why using the training data as a dictionary or lookup table and referring to it is a bad strategy for learning?

There are 2 points that make it as a bad strategy:

1. Using a dictionary or lookup table will make the model overfitting. In other words, the model lack of generalization. The model is prefect for the training dataset. But when comes to the new data, the model fail to generalize to them.
2. As the dataset goes larger, it will be hard to lookup the table since the table is growth with the dataset, which makes the prediction inefficient.

What is it called?

Overfitting

How do we prevent it?

1. pruning

Initially, we have the decision tree. We can do the pruning to the decision tree to reduce with the complexity of the model. If the attribute in the decision is not that useful, we can consider to pruning there. By reducing the complexity of the model to prevent with the overfitting.

2. cross-validation

Basically, we could divide our dataset into two part. One is used for training, the other is used for testing. If there are enough data within the dataset, we could divide into more groups to perform with cross-validation to prevent the overfitting.

3. Early stopping (refer to Wikipedia)

When we are doing the model training, if the loss is start to increase, we could choose to stop to train it. It can prevent the overfitting.

4 Visualize

What does the decision boundary of 1-nearest neighbor classifier for 2 classes (one positive, one negative) look like when the features are 1-D? How about 2-D and 3-D? Can you give a general form?

1) For 1-D, the dataset are in the line. So the decision boundary will be the middle point between the positive and negetive.
2) For 2-D, the dataset are in the plane. So the decision boundary will be the perpendicular bisector between the positive and negetive.
3) For 3-D, the dataset are in the space. So the decision boundary will be the perpendicular bisector plane between the positive and negetive.
4) For general form, it will be the perpendicular bisector hyperplane.

5 kNN and data manipulation

Does the accuracy of a kNN classifier using the Euclidean distance change if you:(a) translate the data (b) scale the data (i.e., multiply all the points by a constant), or (c) rotate the data? Explain. Answer the same for a kNN classifier using Manhattan distance[1] and Cosine Distance[2]. Make sure you provide mathematical proofs or at least some intuition that why your claim is correct.

Manhattan distance

$\mathrm{D}_{\mathrm{Manhattan}}(p,q) = \sum_{i=1}^{n} |p_i - q_i|$

1. Translate the data
Translation doesn't change with the distance calculation here. If we translate all the data within the dataset together, the distance will be the same. So, the accuracy will not be affected. The prove is $\mathrm{D}_{\mathrm{Manhattan}}(p',q') = \sum_{i=1}^{n} |(p_i + t_i) - (q_i + t_i)| = \sum_{i=1}^{n} |p_i - q_i|$

2. Scale the data Scaling will change with the distance calculation here. But if we scale them together, all the distant we get will scale together. So, the accuracy will not be affect since the distance are modified in the same way. Prove: $\mathrm{D}_{\mathrm{Manhattan}}(p',q') = \sum_{i=1}^{n} |k \cdot p_i - k \cdot q_i| = k \sum_{i=1}^{n} |p_i - q_i|$

3. Rotate the data Rotation will change with the distance calculation and the change is not the same for every points. Hence, the accuracy will be affected. Prove: $\mathrm{D}_{\mathrm{Manhattan}}(p',q') = \sum_{i=1}^{n} |(R \cdot p)_i - (R \cdot q)_i|$

Cosine Distance
$\mathrm{D}_{\mathrm{Cosine}}(p,q) = 1 - \frac{p \cdot q}{|p| \, |q|}$
Cosine distance only can be affected by the direction of the vectors. Meanwhile, Translation, scaling and rotation will not change the angle between vectors. Therefore, all of these three operation does not affect the calculation of cosine distance.

[1] `http://en.wikipedia.org/wiki/Taxicab_geometry`
[2] `https://en.wikipedia.org/wiki/Cosine_similarity`

6 Coding

Implement kNN in Python for handwritten digit classification and submit all codes and plots:

Download MNIST digit dataset (60,000 training and 10,000 testing data points) and the starter code from the course page. Each row in the matrix represents a handwritten digit image. The starter code shows how to visualize an example data point. The task is to predict the class (0 to 9) for a given test image, so it is a 10-way classification problem.

1. Write a Python function that implements kNN for this task and reports the accuracy for each class (10 numbers) as well as the average accuracy (one number).

   [acc acc_av] = kNN(images_train, labels_train, images_test, labels_test, k)

   where *acc* is a vector of length 10 and *acc_av* is a scalar. Look at a few correct and wrong predictions to see if it makes sense. To speed it up, in all experiments, you may use only the first 1000 testing images.

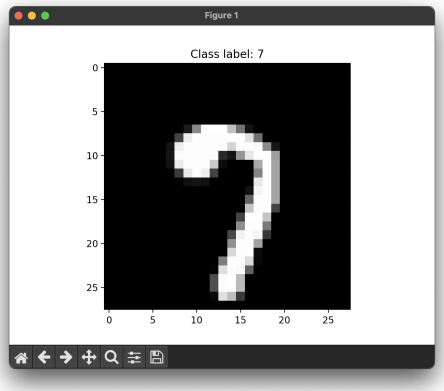
2. For $k = 1$, change the number of training data points (30 to 10,000) to see the change in performance. Plot the average accuracy for 10 different dataset sizes. In the plot, x-axis is for the number of training data and y-axis is for the accuracy.

3. Show the effect of k on the accuracy. Make a plot similar to the above one with multiple colored curves on the top of each other (each for a particular k in [1 3 5 10].)

4. Choose the best k. First choose 2,000 training data randomly (to speed up the experiment). Then, split the training data randomly to two halves (the first for training and the second for cross-validation to choose the best k). Please plot the average accuracy wrt k on the validation set. You may search for k in this list: [1 3 5 10]. Finally, report the accuracy for the best k on the testing data.

For the Starter

```
dataset = loadmat('MNIST_digit_data.mat')
imgTrain = dataset['images_train']
imgTest = dataset['images_test']
labTest = dataset['labels_test']
labTrain = dataset['labels_train']
# shuffle the dataset
imagesTrain, labelsTrain = shuffle(imgTrain, labTrain, random_state=255)

# Display one of the images (starter)
i = 1
display = imagesTrain[i, :].reshape((28, 28), order='F')
plt.imshow(display, cmap='gray')
```



Question 1: implement knn

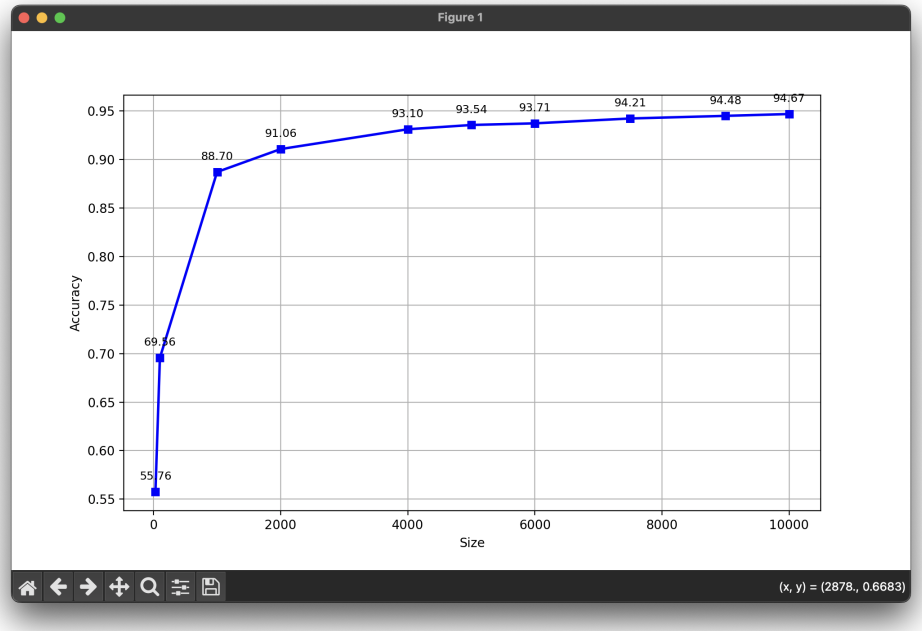
```
def kNN(trainData, trainLab, testData, testLab, k):
    def kNNPredict(imgInput):
        # calculate distances between the test data and all training data
        dis = []
        initDis(dis, imgInput)
        nearest = getNeighbors(dis, k) # get the neighbors

        labels = []
        for i in nearest:
            labels.append(trainLab[i])

        # Predict the label by majority vote
        return majorityVote(labels)

    def initDis(dis, imgInput):
        for img in trainData:
            tmp = getDis(imgInput, img)
            dis.append(tmp)

    # find k nearest neighbors
    def getNeighbors(dis, k):
        sorted = np.argsort(dis)
        return sorted[:k]

    # calculate Euclidean distance
    def getDis(test, train):
        diff = test - train
        squared = diff ** 2
        sumOfSquard = np.sum(squared)
        return np.sqrt(sumOfSquard)

    # predict the label by majority vote
    def majorityVote(nearest):
        cnt = np.bincount(nearest)
        return cnt.argmax()


```

```
    corrects, cnt = [],[]
    for i in range(10):
        corrects.append(0)
        cnt.append(0)


    # Iterate over the test set
    for i in range(len(testData)):
        if i % 100 == 0:
            print(f"Processing the image at {i}")

        imgTest = testData[i]
        if len(testLab.shape) > 1:
            label = testLab[i][0]
        else:
            label = testLab[i]
        cnt[label] += 1


        predictLabel = kNNPredict(imgTest)

        if predictLabel == label:
            corrects[label] += 1

    # calculate accuracy for each digit
    accTmp = []
    for i in range(10):
        if cnt[i] > 0:
            accTmp.append(corrects[i] / cnt[i])
        else:
            accTmp.append(0)

    return accTmp, sum(corrects) / len(testData)
```

Question 2: change the number of training data points

```
# Test for Question 2
print("Question 2 is running...")
sizes = [30, 100, 1000, 2000, 4000, 5000, 6000, 7500, 9000, 10000]
result = []

for size in sizes:
    print(f"size: {size}")
    acc, acc_av = kNN(imgTrain[:size], labels[:size], imgTest[:size],
       labTest[:size], 1)
    result.append(acc_av)

plt.figure(figsize=(10, 6))
plt.plot(sizes, result, marker='s', linestyle='-', color='b', linewidth=2,
    markersize=6)

for i, acc in enumerate(result):
```

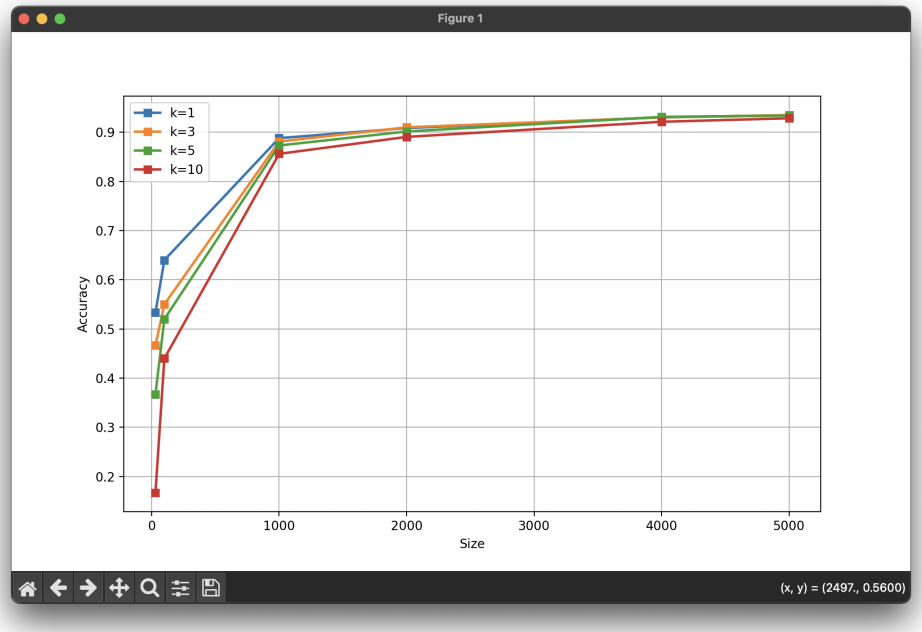
```
    plt.text(sizes[i], acc + 0.01, "{:.2f}".format(acc * 100), ha='center'
        , va='bottom', fontsize=9)

plt.xlabel('Size')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



Question 3: show the effect of k on the accuracy

```
# Test for Question 3
print("Question 3 is running...")
K = [1, 3, 5, 10]
sizes = [30, 100, 1000, 2000, 4000, 5000]
result = {k: [] for k in K}

for tmp in K:
    for size in sizes:
        print(f"k: {tmp}, size: {size}")
        acc, acc_av = kNN(imgTrain[:size], labels[:size], imgTest[:size],
            labTest[:size], tmp)
        result[tmp].append(acc_av)

plt.figure(figsize=(10, 6))

for k in K:
    plt.plot(sizes, result[k], marker='s', linestyle='-', linewidth=2,
        markersize=6, label=f'k={k}')

```

```
plt.xlabel('Size')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()
plt.show()
```



Question 4: the average accuracy wrt k on the validation set

Accuracy on the test data with the best k (1) is: 0.89

```
# Test for Question 4
print("Question 4 is running...")
K = [1, 3, 5, 10]
result = []

for tmp in K:
    print(f"k: {tmp}")
    acc, acc_av = kNN(imgTrain[0:1000], labels[0:1000], imgTest
        [1000:2000], labTest[1000:2000], tmp)
    result.append(acc_av)

plt.figure(figsize=(10, 6))
plt.plot(K, result, marker='s', linestyle='-', color='b', linewidth=2,
    markersize=6)

for i, acc in enumerate(result):
    plt.text(K[i], acc + 0.01, "{:.2f}".format(acc * 100), ha='center', va
        ='bottom', fontsize=9)

plt.xlabel('K')
```

```
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()


best = K[np.argmax(result)]
acc, acc_av = kNN(imgTrain[0:1000], labels[0:1000], imgTest[0:2000],
    labTest[0:2000], best)
print(f"Accuracy on the test data with the best k ({best}) is: {acc_av:.2f
    }")
```