

HW1: Decision trees and KNN

1 Math Practice (NOT GRADED)

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. $p(a | b) = p(a, b)/p(b)$
4. $\log x + \log y = \log(xy)$
5. $p(x | y, z) = p(x | y)p(x | z)$
6. $\int_{-\infty}^{\infty} dx \exp[-(\pi/2)x^2] = \sqrt{2}$
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. $|\mathbf{u}^\top \mathbf{v}| \geq \|\mathbf{u}\| \times \|\mathbf{v}\|$, where $|\cdot|$ denotes absolute value and $\mathbf{u}^\top \mathbf{v}$ is the dot product of \mathbf{u} and \mathbf{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. $\|\alpha \mathbf{u} + \mathbf{v}\|^2 = \alpha^2 \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2$, where $\|\cdot\|$ denotes Euclidean norm, α is a scalar and \mathbf{u} and \mathbf{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 let's 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 .

For a continuous attribute like **Time**, we can consider all the unique values of **Time** in our training data. For example, the training data may have times 9:00, 11:00, 12:00, and 16:00. Unlike the yes/no case where we only have a single potential boundary, we now can consider splitting at any time between the times in our training data. A simple way to do this would be to take the midpoint between each

pair of consecutive unique values in the training data. In our example, this would mean our candidates for τ would be 10:00 (between 9:00 and 11:00), 11:30 (between 11:00 and 12:00), or 14:00 (between 12:00 and 16:00). We can then evaluate each candidate τ as we did for binary attributes, but this time across all candidate thresholds of a continuous attribute. In other words, we assign a score like misclassification error (what we did in lecture) to each candidate τ and pick the one that optimizes our chosen score.

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.*

In the binary attribute case, once the attribute is used to split the data, no more useful information can be gained from splitting on it again. For example, if we split on the attribute "Morning?", then all data points on one side of the split will have the "Morning?" attribute set to true, and all data points on the other side will have it set to false. Therefore, if split again on "Morning?" down either branch, no new information will be gained since all the data points on that branch already have the same value for that attribute.

However, with a continuous attribute like **Time**, even after splitting once, there is still information to be gained down either branch. For example, if you split at $\tau = 12 : 00$, then on one side you have all data points with **Time** $\geq 12 : 00$. Splitting on **Time** again at a different time can help further partition the data. For example, on the branch where **Time** $\geq 12 : 00$, the data can be partitioned at a new split on **Time** such as 14:00.

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?

Using the training data as a dictionary or lookup table and referring to it is a bad strategy for learning because it will fail to generalize to unseen data. The analogy covered in class was of a student memorizing answers to questions she has seen in lectures instead of understanding the underlying concepts. When presented with new questions on an exam which are similar but not identical, the student will struggle to answer them correctly.

This phenomenon is known as **overfitting** where a model learns the training data too specifically including noise and outliers, rather than capturing the general patterns that would allow it to perform well on new data.

We can utilize several strategies to prevent overfitting in decision trees:

- **Validation Set:** We can split the training data into an additional validation set. For example, we can have 60% training, 20% validation, and 20% testing. We perform training on the training set, but then evaluate the model's performance on the validation set. If the error on the validation set starts increasing while the training error continues decreasing, it indicates 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?

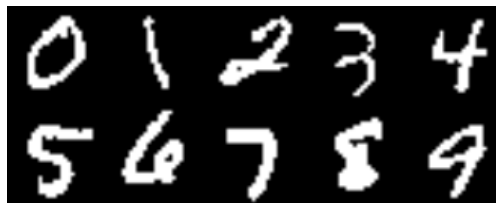
5 kNN and data manipulation

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 distance¹ and Cosine Distance². Make sure you provide mathematical proofs or at least some intuition that why your claim is correct.

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).

$[acc \text{ } acc_av] = kNN(images_train, labels_train, images_test, labels_test, k)$

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.

¹http://en.wikipedia.org/wiki/Taxicab_geometry

²https://en.wikipedia.org/wiki/Cosine_similarity