Image Classification

Mohit Deshpande

March 16, 2017

1 Introduction to Machine Learning

Before machine learning and artificial intelligence, computers were very unintelligent machines. Even though they were good at computing numbers very fast, they had to be told *exactly* what to do. However, people started asking the question: "what if we could teach machines to how to learn?" This arose mostly from the realm of science fiction which already depicted robots as being sentient beings. There are tons of popular science fiction stories about robots.

So the goal of machine learning and artificial intelligence is to embed some knowledge into our computer. We don't want to nor can we program and hard-code every possible scenario and example that we could encounter. This would be very time-consuming and, in some cases, downright impossible! Instead, we want to give our input to a machine learning model and have it tell us some insights about our data. But if we

Artificial Intelligence and Machine Learning have tons of different subfields within them.

- Neural Networks
- Deep Learning
- Clustering
- Reinforcement Learning
- Decision Trees
- Support Vector Machines

- etc.

It's a very popular and rapidly-expanding field.

2 Supervised Classification

One particular type of machine learning is **classification**. With **classification**, we have a set of labeled data and we're trying to determine how a new data point fits into our existing set. For example, consider this scatter plot.

Scatter plot with X and O.

We have X's and O's as our labeled set of data. So given a new point, we want to determine if it should belong to the X's or the O's.

3 Image Classification

Image classification is the problem of assigning labels to an input image. Given any image, we want to assign some labels to it. Remember that computers see images as pixels so we're trying to add some higher-level meaning to an image: particularly what's in the image, like suppose we wanted to classify a bird.

This isn't an easy task by any means though. There are quite a bit of challenges that come with this:

- Scaling
- Occlusion
- Illumination
- Background
- etc.

A really good image classifier should be robust enough to handle these cases. If we wanted to build a classifier, we're taking a **data-driver** approach. This means that we're providing our AI with labeled examples of what a

bird looks like. To take this approach, however, we'll need a large dataset of labeled examples. Many datasets like this actually exist so you won't have to do any labeling yourself!

Given this training data, we have to split it into training and testing data. We'll give our AI plenty of labeled examples and training it or learn a model. Then, we have to evaluate how well our classifier has learned by using the testing data and comparing the output of the classifier to the true output called **ground truth**.

4 NN Classifier 1

The simplest kind of classifier is called the **Nearest Neighbors classifier**. It's pretty rare to see this in practice, but it's also simple enough for us to understand really well and we can take many of the principles we learn from this classifier and extend it to more complicated learning algorithms. Here's the algorithm: given a test image, the nearest neighbor classifier will compare it to every training image that it has seen, find the training image that it most closely resembles, and return the label of that closest training image.

But how do we compare images? We can use the ℓ_1 distance to measure the similarity of the images. In this context, we can write ℓ_1 like the following. Usually what we do, so that we don't confuse this with matrix norms, is to take the 2D matrix and stretch it out into one large column and take a single sum.

$$d_1(I_1, I_2) = \sum_{p=0}^{m+n} |I_1^p - I_2^p|$$

Let's do an example of this.

$$A = \begin{bmatrix} 4 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1 \\ 7 & 7 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 7 \\ 7 \end{bmatrix}$$

$$d_1(A, B) = \sum_p | \begin{bmatrix} 4 \\ 2 \\ 3 \\ 4 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \\ 7 \\ 7 \end{bmatrix} | = \sum_p \begin{bmatrix} 4 \\ 1 \\ 4 \\ 3 \end{bmatrix} = 13$$

This isn't the only, there are other ways we can compute this image distance. For example, ℓ_2 is another one that is similar.

$$d_2(I_1, I_2) = \sqrt{\sum_{p=0}^{m+n} (I_1^p - I_2^p)^2}$$

5 k-NN Classifier

We can generalize our nearest neighbor classifier to the **k Nearest Neighbor Classifier**. Instead of just picking the label of the closest image, instead, we get the k closest images and have them all vote on what the label should be! For example, suppose I had a kNN classifier with $k = 5$. If I had an input image of something that looks like a cat, maybe my closest images have the labels dog, cat, cat, horse, airplane. If we tally up the votes, we have 2 votes for cat, 1 vote for dog, 1 vote for horse, and 1 vote for airplane. So we assign this new image the label of cat because we have the most votes for cat!

This has the effect analogous to image smoothing. To see this, let's go back to a scatter plot example. If we used nearest neighbor, we would get very jagged edges. But, if we used kNN, then we might incorrectly classify a few data points, but the resulting boundary is *much* smoother. As we

increase the value of k, we get a smoother boundary. In fact, if we use $k = 1$, we get the nearest neighbor classifier!

But what value of k do we choose? Which one is the "best"? We'll revisit this very soon, but suppose we have a good value of k.

Usually, the k Nearest Neighbor Classifier is almost always better than the nearest neighbor classifier.

Pros	Cons
Simple to implement & understand	Not as accurate as state-of-the-art
No training time	Long time to test

Figure 1: Pros and cons of k-NN classifier.

There are better approaches to image classification like convolutional neural networks, but they are *far* more complicated and intricate than our simple k-NN classifier. That being said, the complexity doesn't go to waste because they do produce more accurate models.

6 Training Protocol & Practices

I previously posed the question "what value of k do we use for our kNN classifier?" This is an example of what we call a **hyperparameter**. Usually machine learning algorithms can have quite a few hyperparameters. For example, even our distance function ℓ_1 or ℓ_2 can be considered a hyperparameter. These are choices aren't that obvious to pick so we have to try different values and see what values produce the best model.

When tuning hyperparameters, *we should not use the testing set!* The testing set should only be used after tweaking your AI to get the final evaluation. But if we can't use the testing set, what do we use? It's common to partition our data into 3 sets: training, validation, and testing. The testing set is what you have to hide under a rock until the very end. Hyperparameter tuning and general AI tweaking can be done on the validation set.

The reason we don't use the testing set is because we could get **overfitting**. If we did use the testing set, we'll be showing our AI the testing many times as we try to tweak our AI. But if w keep using the testing set, then characteristics of our testing set start to bleed over into our training and into our AI and our AI won't actually be learning, it'll just be memorizing. Suppose I showed you some flash cards to test your knowledge, but, when

you answered incorrectly, I showed you the same flash cards in the same order. Then, at some point, you wouldn't actually be learning, you'd just be memorizing the order of the flash cards and the correct answers. The same thing can happen to our AI. This is why, when we do the formal evaluation, we show it data it has never seen before.

In practice, we actually train using something more complicated called **cross validation**. Basically, we can partition all of our data into testing data and several **folds**. Then we reserve one for the validation set and the rest for training. Then we can use the validation set to train the hyperparameters. To prevent the same overfitting from happening to our validation set, we iterate over which sets we use as training and which sets we use as validation.

image showing cross validation