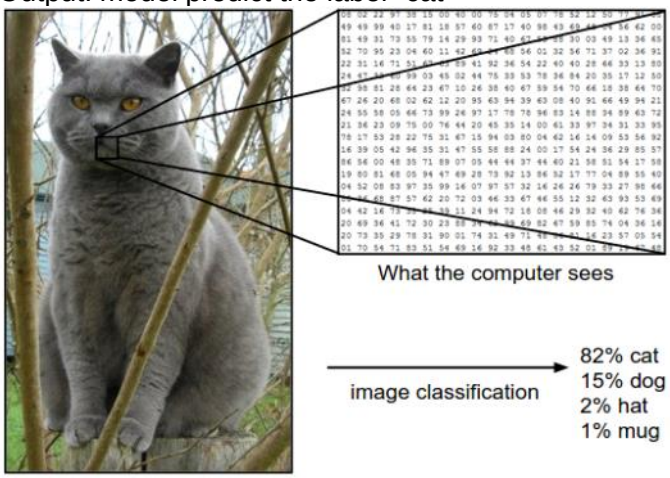
Image Classifier

2024年9月6日 18:05

Image Classifier [Link]

Input: Image of cat
Output: model predict the label "cat"

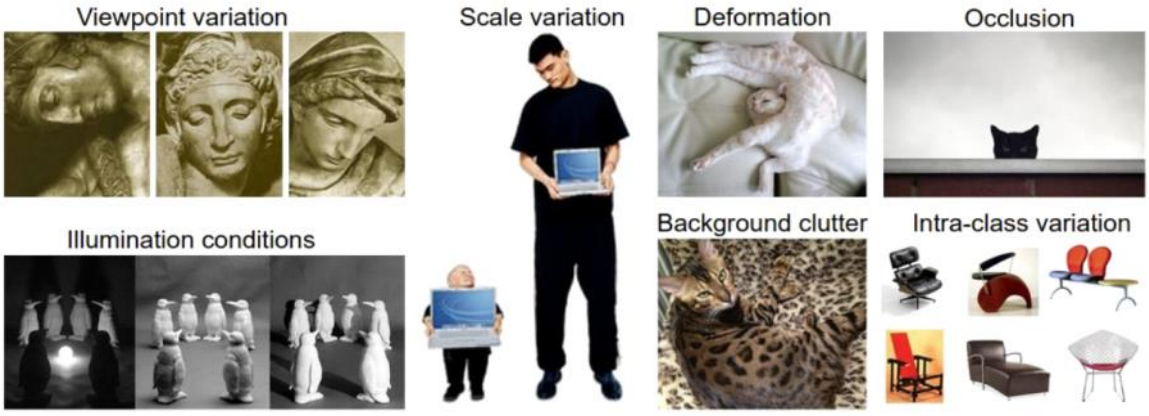


Images are 3d arrays, each pixel can be RGB (0-255), 3 RGB channels resulting a total size, of Width x Height x 3.

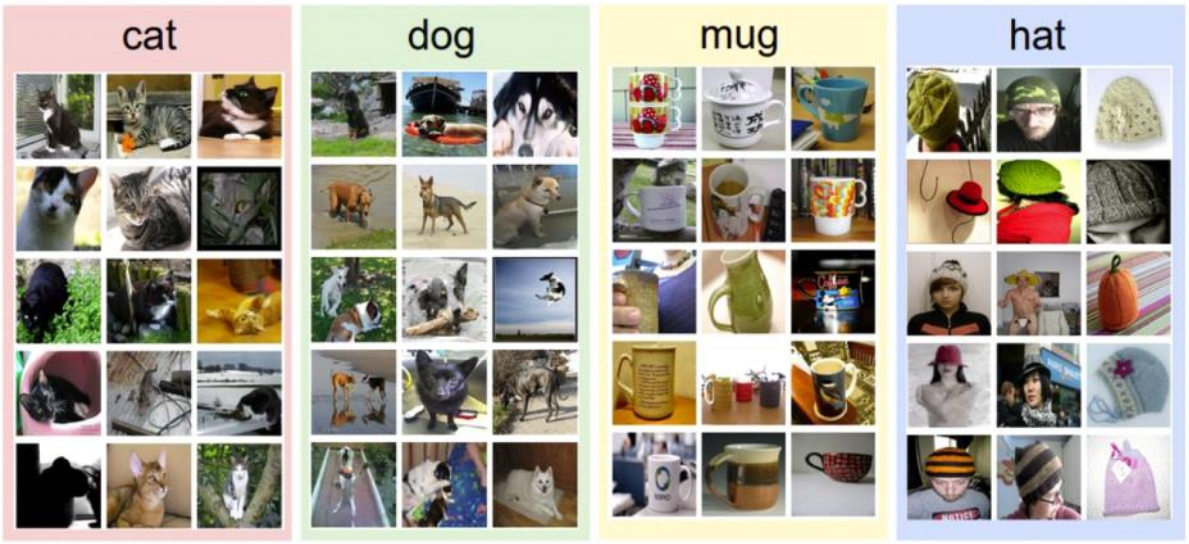
Feature Extraction: extracts features from image

Classification: feed extracted features into a classifier, it assigns a probability to each possible label, the highest probability is selected as a prediction

Challenges:



Data driven approach:
Provide computer with many examples of each class, then develop algorithms that look & learn from these examples, to learn the visual appearance of each class.



Steps:

- Input (Acquire training dataset, N images, K classes)
- Learning (Training classifier/model)
- Evaluation (Test classifier with unseen images, see its accuracy)

Nearest Neighbour Classifier:

With a training set (CIFAR-10) of 50,000 imgs, the remaining 10,000 imgs are test set.

Each img is a 32x32x3 matrix, compare 2 imgs (representing as I1, I2 vectors) by compute d1 distance (Distance Function):

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

Each element is subtracted elementwise, then take the summation

- If 2 imgs are identical, result is 0
- If 2 imgs are very diff, result will be large

test image					training image					pixel-wise absolute value differences				
56	32	10	18	-	10	20	24	17	=	46	12	14	1	→ 456
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200		12	16	178	170		12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	

Speed consideration:

- Train: o(1)
- Predict: o(N) - compare w/ every imgs (It's considered as slow)

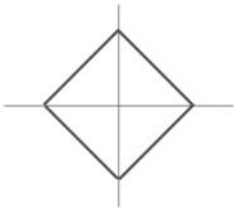
Another distance is L2 distance, where Euclidean distance is computed:

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

L1 vs L2 distance:

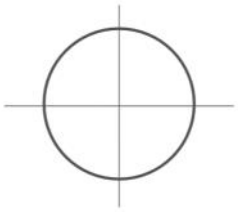
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



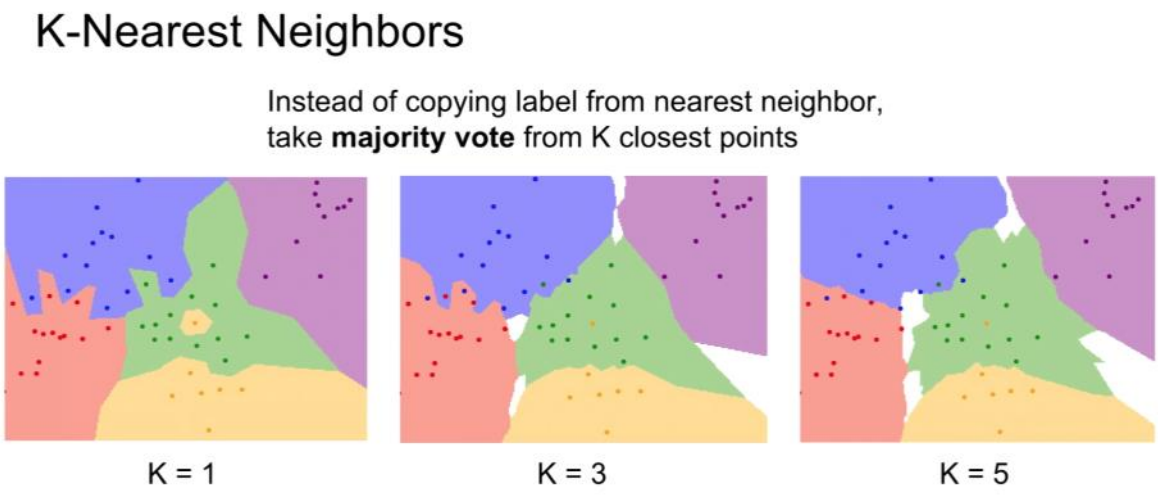
L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K Nearest Neighbour (kNN)

Idea: Find top k closest images, use them to vote for the label of img



Color Region: Decision Boundary of classifier with L2 dist, K=5 perform better than K=1 classifier (draw better boundaries)

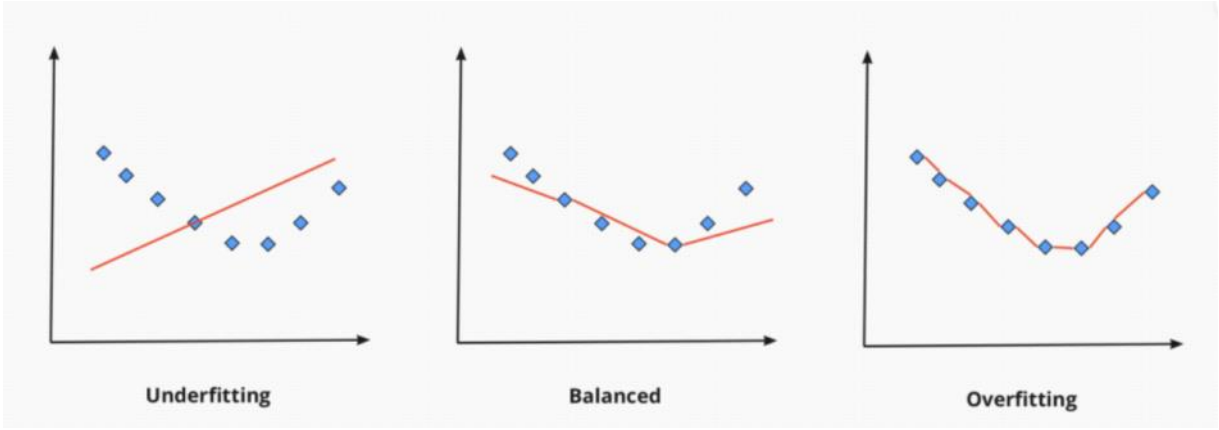
Hyperparameter Tuning

Hyperparameters: k value, L1/L2 dist, dot product, etc.

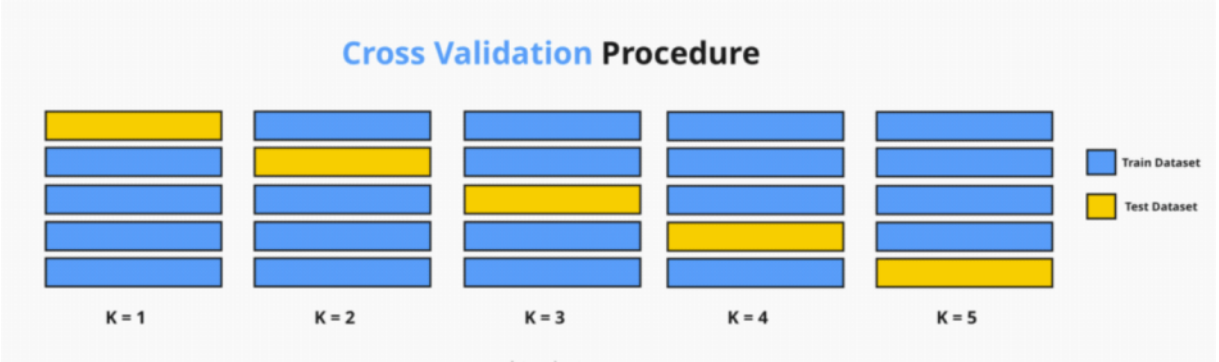
Generalization Idea: ML model is to work with unseen data, not works best with training data

Overfit:

- Model perform well on training data, but not unseen data
- Model might "memorize" the test set, leads to poor generalization
- Can use cross-validation to detect, eliminate overfitting



Cross-validation:
Use same datasets, but creating multiple training sets (creating subsets)



Usually, it's a 80%-20% split, divide data into multiple subsets, perform model testing, repeated until each subset of the dataset is completed.

We would then iterate over which fold is the validation fold, evaluate the performance, and finally average the performance across the different folds.

***In Practice:**
Due to intensive computation, single validation split is used. Typical train-test split is 70-30/80-20/90-10

Summary:

- We introduced the problem of **Image Classification**, in which we are given a set of images that are all labeled with a single category. We are then asked to predict these categories for a novel set of test images and measure the accuracy of the predictions.
- We introduced a simple classifier called the **Nearest Neighbor classifier**. We saw that there are multiple hyper-parameters (such as value of k, or the type of distance used to compare examples) that are associated with this classifier and that there was no obvious way of choosing them.
- We saw that the correct way to set these hyperparameters is to split your training data into two: a training set and a fake test set, which we call **validation set**. We try different hyperparameter values and keep the values that lead to the best performance on the validation set.
- If the lack of training data is a concern, we discussed a procedure called **cross-validation**, which can help reduce noise in estimating which hyperparameters work best.
- Once the best hyperparameters are found, we fix them and perform a single **evaluation** on the actual test set.
- We saw that Nearest Neighbor can get us about 40% accuracy on CIFAR-10. It is simple to implement but requires us to store the entire training set and it is expensive to evaluate on a test image.
- Finally, we saw that the use of L1 or L2 distances on raw pixel values is not adequate since the distances correlate more strongly with backgrounds and color distributions of images than with their semantic content.

Applying kNN in practice:

1. Preprocess your data: Normalize the features in your data (e.g. one pixel in images) to have zero mean and unit variance. We will cover this in more detail in later sections, and chose not to cover data normalization in this section because pixels in images are usually homogeneous and do not exhibit widely different distributions, alleviating the need for data normalization.
2. If your data is very high-dimensional, consider using a dimensionality reduction technique such as PCA (wiki ref, CS229ref, blog ref), NCA (wiki ref, blog ref), or even Random Projections.
3. Split your training data randomly into train/val splits. As a rule of thumb, between 70-90% of your data usually goes to the train split. This setting depends on how many hyperparameters you have and how much of an influence you expect them to have. If there are many hyperparameters to estimate, you should err on the side of having larger validation set to estimate them effectively. If you are concerned about the size of your validation data, it is best to split the training data into folds and perform cross-validation. If you can afford the computational budget it is always safer to go with cross-validation (the more folds the better, but more expensive).
4. Train and evaluate the kNN classifier on the validation data (for all folds, if doing cross-validation) for many choices of **k** (e.g. the more the better) and across different distance types (L1 and L2 are good candidates)
5. If your kNN classifier is running too long, consider using an Approximate Nearest Neighbor library (e.g. FLANN) to accelerate the retrieval (at cost of some accuracy).
6. Take note of the hyperparameters that gave the best results. There is a question of whether you should use the full training set with the best hyperparameters, since the optimal hyperparameters might change if you were to fold the validation data into your training set (since the size of the data would be larger). In practice it is cleaner to not use the validation data in the final classifier and consider it to be *burned* on estimating the hyperparameters. Evaluate the best model on the test set. Report the test set accuracy and declare the result to be the performance of the kNN classifier on your data.