Lec6_Image_Classification

Classification: class label, predict label of unseen sample based on learned from training sample

Supervised learning: training set + validation set → model → testing set + label

K nearest neighbors (kNN Classifier)

- k=1: assign the unseen with the label of its nearest neighbor
- k>1: assign the dominating label among these of the k nearest neighbors

Bayesian classifiers

$P(A|x)$: prob of A is observed when seeing an x - The Conditional Probability (Likelihood)

$P(B|x)$: prob of B is observed when seeing an x

$P(A|x) = P(x|A)P(A)/P(x) \quad P(A) = P(A)/P(A+B) \quad P(x|A) = P(x)/P(A)$

$P(B|x) = P(x|B)P(B)/P(x) \quad P(B) = P(B)/P(A+B) \quad P(x|B) = P(x)/P(B)$

Decision function: x is A if $P(A|x) > P(B|x)$, B otherwise

Advantages: • Fast, Extendable to multi class problems • Requires less training examples, Works well for categorical data

Disadvantages: • Features are assumed to be independent to each other (not true in real world applications) • Zero frequency problem

Support vector machines (SVM)

Linear Separators

$$[W - 1]^T[xy] + b = -y + wx = 0$$
$$w^Tx + b = 0 \quad f(x) = sign(w^Tx) + b$$

Margin seperator:

$$r = \frac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$$

support vector: Examples closest to hyperplane超平面 margin ρ of the separator: distance b/t support vectors

Maximum Margin Classification: Probably Approximately Correct(PAC theory), Implies that only SV matter; other training examples are ignorable.

Soft Margin Classification: not linearly separable

Slack variables ξ i can be added to allow misclassification of difficult or noisy examples, resulting margin called soft

Non-linear SVMs: Φ:x → φ(x)

Kernel Functions

Linear: $K(x_i, x_j) = x_i^T x_j$ - Mapping Φ: x → φ(x), where φ(x) is x itself

Polynomial of power p: $K(x_i, x_j) = (1 + x_i^T x_j)^p$ - where φ(x) has (d+p,p) dimension

Gaussian (radial-basis function): $K(x_i, x_j) =$ - where φ(x) is infinite-dim: every point mapped $e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$

Higher-dimensional space still has intrinsic dimensionality (mapping is not onto, but linear seperators in it correspond to non-linear seperator in original space

Rock Paper Scissors

Tutorial

```
## extract features
sift = cv2.SIFT_create()
def get_feat(img):
    # return hog(img)
#     return sift_feat(img)
    return gray_histogram(img)

def calc_distance(x, y):
    return L2_distance(x, y)
#     return L2_distance_sift(x, y)

def sift_feat(img):
    kps, des =  sift.detectAndCompute(img, None)
    responses = [kp.response for kp in kps]
    order = np.argsort(responses)[::-1]
    return np.array(des[order[:30]])

def gray_histogram(img: np.array, norm: bool = True) -> np.array:
    if img.shape[-1] == 3:
        img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    hist = np.array([len(img[img == i]) for i in range(256)])
    if norm:
        return hist / np.size(img)

    return hist
def L2_distance(x, y):
    return ((x - y) ** 2).sum() ** 0.5
def L2_distance_sift(x, y):
    dist = ((x[:, None] - y[None, :])**2).sum(axis=-1).min(axis=-1)
    dist.sort()
    return dist[:15].mean()
```

```
for sample in samples['train']:
    sample.feat = get_feat(sample.img)
for sample in samples['val']:
    sample.feat = get_feat(sample.img)

# **kNN #works only for few imgs, don't have model for classification(2/4)**
def kNN(test_sample, train_samples, k=3):
    distances = []
    for sample in train_samples:
        distance = calc_distance(test_sample.feat, sample.feat)
        distances.append((distance, sample))

    distances = sorted(distances, key=lambda x: x[0])[:k]

    label_count = {}
    plt.figure(figsize=((k+1)*4, 4))
    plt.subplot(1, k+1, 1)
    plt.title(f'{test_sample.fname} in G{test_sample.label}(val)')
    plt.imshow(test_sample.img)

    for i, (distance, sample) in enumerate(distances):
        plt.subplot(1, k+1, i+2)
        plt.title(f'{sample.fname} in G{sample.label}(train)')
        plt.imshow(sample.img)
        label_count[sample.label] = label_count.get(sample.label, 0) + 1
    max_label, max_count = -1, 0

    for label, count in label_count.items():
        if count > max_count:
            max_label, max_count = label, count
    test_sample.pred = max_label

**# train with SVM**
model = SVC(kernel='rbf')

train_samples = [sample.feat for sample in samples['train']]
train_labels = [sample.label for sample in samples['train']]
model.fit(train_samples, train_labels)

# test with SVM
test_samples = [sample.feat for sample in samples['val']]
results = model.predict(test_samples)
for sample, result in zip(samples['val'], results):
    sample.pred = result
    print(sample.fname, 'with label', sample.label, 'is predicted as',
sample.pred)
# display the results (target images with the predicted labels and ground truth
label)
```