ECE 239 Week 3

01/26/2018

Tianwei Xing, Cheng Zheng

OH: Tue. 3pm-5pm, Tue. 10am-12pm

Eng. IV 67-112

Outline

- KNN
- Softmax
- SVM

KNN

Non-parametric method:

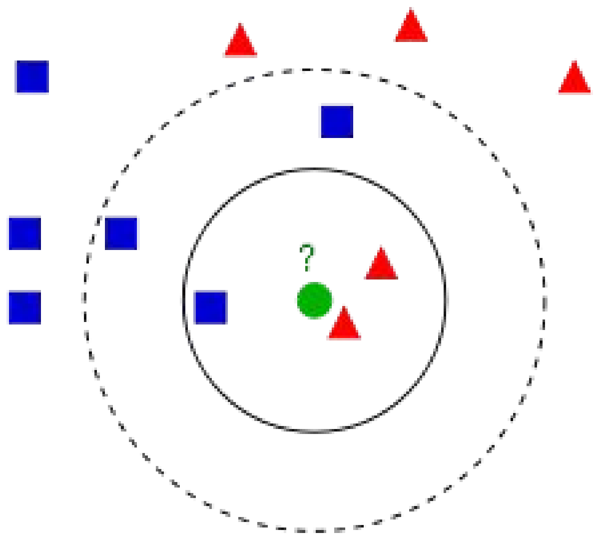
New instance is classified by a majority vote of its neighbors.

Pros & Cons

Distance:

Euclidean distance; (Text: Hamming)

Drawback: skewed dataset



KNN

Compute distance: L-n norm

Faster computation: Vectorization

- Reduce the nested for-loops (2 for-loops)
- Use broadcasting
- Expand $|| x\_i - x\_t ||^2 = (x\_i - x\_t)^T (x\_i - x\_t) = ?$

Predict labels: sort, find, vote

Find **optimal k and distance metric**: grid search?

Accuracy / Error-rate / N-fold cross validation

Softmax

A **generalization** of binary logistic regression classifier:

Produce a more intuitive output (normalized class prob)

$$\mathrm{softmax}_i(\mathbf{x}) = \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}}$$

for $a_i(\mathbf{x}) = \mathbf{w}_i^T \mathbf{x} + w_i$ and c being the number of classes.

If we let $\theta = \{\mathbf{w}_j, w_j\}_{j=1,\ldots,c}$, then $\mathrm{softmax}_i(\mathbf{x})$ can be interpreted as the probability that $\mathbf{x}$ belongs to class i. That is,

$$\Pr(y^{(j)} = i | \mathbf{x}^{(j)}, \theta) = \mathrm{softmax}_i(\mathbf{x}^{(j)})$$

Loss function: (cross-entropy loss)

$$\arg\min_{\theta} \sum_{i=1}^{m} \left(\log \sum_{j=1}^{c} e^{a_j(\mathbf{x})} - a_{y^{(i)}}(\mathbf{x}^{(i)}) \right)$$

Derived from MLE formulation: min -log(likelihood)

Softmax

Numerical stability:

$$\log k = -\max_i a_i(\mathbf{x})$$

$$\begin{aligned} \mathrm{softmax}_i(\mathbf{x}) &= \frac{e^{a_i(\mathbf{x})}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})}} \\ &= \frac{k e^{a_i(\mathbf{x})}}{k \sum_{j=1}^{c} e^{a_j(\mathbf{x})}} \\ &= \frac{e^{a_i(\mathbf{x})+\log k}}{\sum_{j=1}^{c} e^{a_j(\mathbf{x})+\log k}} \end{aligned}$$

Calculate the loss and grad: chain rule.

Vectorization: batch

Training: *W = W - learning_rate x grad*

Prediction: top scoring class

Gradient descent

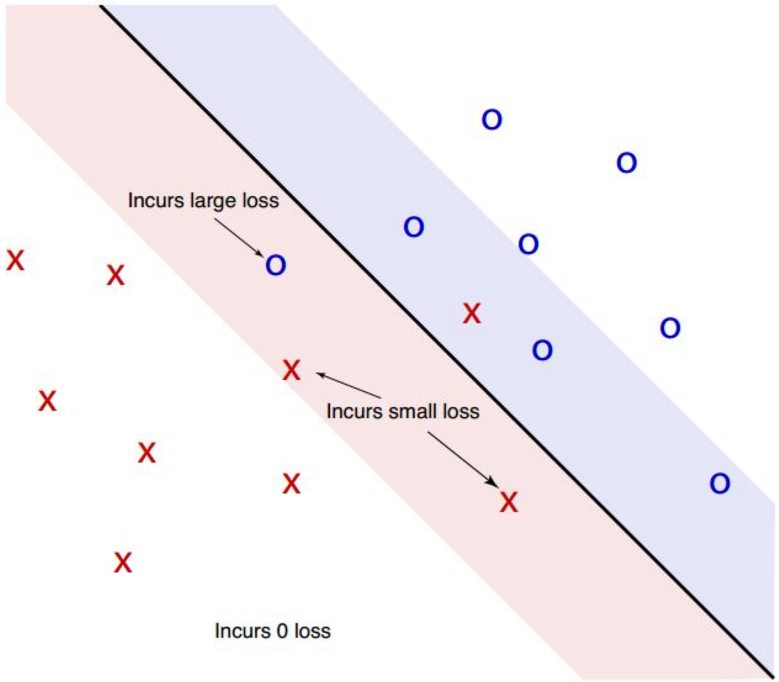
Hence, we arrive at gradient descent. To update $\mathbf{x}$ so as to minimize $f(\mathbf{x})$, we repeatedly calculate:

$$\mathbf{x} := \mathbf{x} - \epsilon \nabla_x f(\mathbf{x})$$

ϵ is typically called the *learning rate*. It can change over iterations. Setting the value of ϵ appropriately is an important part of deep learning.

Optimizing softmax classifier: Using different learning rates to train classifiers and test their performance on validation data

Support vector machine



Hinge loss:

$$L_i = \sum_{j \neq y_i} \left[\max(0, x_i w_j - x_i w_{y_i} + \Delta) \right]$$

i iterates over all N samples,
j iterates over all C classes,
Wj is the weights for computing score of class j.
yi is the index of correct class of xi
Δ=1 is the margin parameter.

Calculate the gradient

Covered in lecture, recall from lecture slides

Vectorization:

```
def fast_loss_and_grad(self, X, y):
  """
  A vectorized implementation of loss_and_grad. It shares the same
inputs and ouptuts as loss_and_grad.
  """
  loss = 0.0
  grad = np.zeros(self.W.shape) # initialize the gradient as zero

  # ================================================================ #
  # YOUR CODE HERE:
  #   Calculate the SVM loss WITHOUT any for loops.
  # ================================================================ #

  # ================================================================ #
  # END YOUR CODE HERE
  # ================================================================ #



  # ================================================================ #
  # YOUR CODE HERE:
  #   Calculate the SVM grad WITHOUT any for loops.
  # ================================================================ #

  # ================================================================ #
  # END YOUR CODE HERE
  # ================================================================ #

  return loss, grad
```

Hint: many of numpy function support broadcasting

numpy.maximum(x1, x2)

Gradient Descent: the same as softmax