Lunds Tekniska Högskola
Institutionen för datavetenskap

EDAN95
HT 2018

# Lösning: Tillämpad Maskininlärning
# Solution: Applied Machine Learning
# Tentamen 2019–01–08, 08.00–13.00

## 1 Boosting (JM): 5p

Boosting algorithm

① initialize
weights to be $\frac{1}{n}$ (for n samples)

② perform T times (T – number of weak learners to use)

Ⓐ find weak learner $h_t$ that minimizes error $\varepsilon_t$ the weighted sum of misclassified points

choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

Ⓑ Add to ensemble $F_t(x) = F_{t-1}(x) + \alpha_t h_t(x)$

Ⓒ Update weights

$w_{i,t+1} = w_{it} e^{-y_i \alpha_t h_t(x_i)}$

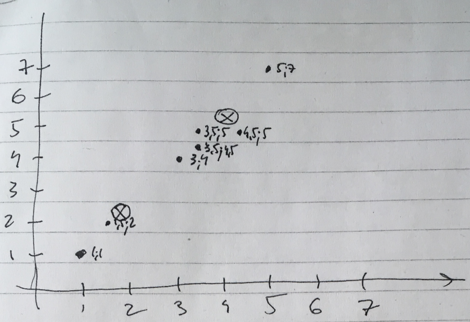
Renormalize such that $\sum w_{it+1} = 1$

output $F_T(x)$

## 2 k-Means (JM): 10p

a)

2a




① seeds (1,1) & (5,7)

cluster 1 (1;1) (1,5;2) (3;4)

cluster 2 (5;7) (3,5;4,5) (3,5;5) (4,5;5)

means

$(1+1{,}5+3)/3 = 5{,}5/3 = 1\frac{5}{6} \approx 1{,}87$

$(1+2+4)/3 = 7/3 = 2\frac{1}{3} \approx 2{,}33$

$(5+3{,}5+3{,}5+4{,}5)/4 = 16{,}5/4 = 4\frac{1}{8} = 4{,}125$

$(7+4{,}5+5+5)/4 = 21{,}5/4 = 5\frac{3}{8} = 5{,}375$

cluster 1 (1,87;2,33) : (1;1) (1,5;2)

(3,4) ?

cluster 2 (4,125;5,375) : (5,7) (3,5;4,5) (3,5;5) (4,5;5) b)

2b

$$\left(3-1\tfrac{5}{6}\right)^2+\left(4-2\tfrac{1}{3}\right)^2$$

$$\cancel{(3-1{,}87)^2+(4-2{,}33)^2}$$

$$(4{,}125-3)^2+(5{,}375-4)^2=$$

$$\left(4\tfrac{1}{8}-3\right)^2+\left(5\tfrac{3}{8}-4\right)^2=\left(\frac{9}{8}\right)^2+\left(\frac{11}{8}\right)^2=\frac{81}{64}+\frac{121}{64}=$$

$$=\frac{202}{64}<4$$

$$\left(1\tfrac{1}{6}\right)^2+\left(1\tfrac{2}{3}\right)^2=$$

$$=\left(\frac{7}{6}\right)^2+\left(\frac{5}{3}\right)^2=\frac{49}{36}+\frac{100}{36}=\frac{149}{36}>4$$

## 3 K-nearest neighbour (JM): 5p

1. b)
2. c)
3. c)
4. a)
5. a)

# 4 Neural networks (PN): 12+9+9 = 30p

## 4.1 Convolutional Neural Networks

A suggestion for a solution to the programming task:

```
#!/usr/bin/env python
# coding: utf-8

from keras import layers
from keras import models
from keras.datasets import cifar10
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels)
  = cifar10.load_data()

train_images = train_images.reshape((50000, 32, 32, 3))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 32, 32, 3))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3), activation='relu',
                        input_shape=(32, 32, 3), use_bias=False))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Conv2D(128, (3, 3), activation='relu', use_bias=False))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu', use_bias=False))
model.add(layers.Dense(10, activation='softmax', use_bias=False))

model.summary()

model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3),
                        activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Conv2D(245, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(2, 2))
```

```
model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10, activation='softmax'))

model.summary()

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=2, batch_size=64)

test_loss, test_acc = model.evaluate(test_images, test_labels)

print(test_acc)
```

# 5 Markov Decision Processes (VK): 4+4+4+5+5+3 = 25p

1. Please refer to the lecture slides

2. Please refer to the lecture slides

3. a) $v_\pi(s) = E_\pi \{R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\}$

   b) $q_\pi(s,a) = E_\pi \{R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\}$

4. $$v_\pi(s) = \sum_{a \in \mathbf{A}} \pi(a|s) \left( R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_\pi(s') \right)$$

   $v_\pi(s) = 0.4 \cdot 7 + 0.6 \cdot (-1 + 0.9\ (0.5 \cdot 2 + 0.5 \cdot 3)) = 0.2 + 0.6 \cdot (-1 + 0.9\ (1 + 1.5)) = 2.8 + 0.6 \cdot 1.25 = 3.55$

5. $$v^*(s) = \max_{a \in \mathbf{A}} \left( R_s^a + \gamma \sum_{s' \in \mathbf{S}} P_{ss'}^a v_*(s') \right)$$

   $q^*(s, \text{STAY}) = (-1 + 0.9\ (0.5 \cdot 2 + 0.5 \cdot 3)) = 1.25$
   $q^*(s, \text{RUN}) = 7$
   $v^*(s) = \max\{q^*(s, \text{RUN}), q^*(s, \text{STAY})\}$

6.

# 6 Reinforcement Learning / Q-Learning (ET): 10+5+3+4+3 = 25p

In general, all answers need to be motivated.

1. A: The function implements **Policy Iteration** as explained and exemplified in the lecture. **a** runs over states, **b** over the possible actions, **c** is the discount factor $\gamma$, **d** contains the transition matrix (*d(i,j)* contains the resulting state when taking action $j$ in state $i$), and **e** the reward matrix (*e(i,j)* is the reward $r$ for taking action $j$ in state $i$). The results of the function are then the (optimal) policy $\pi$ in **res** (i.e. *res(i)* is which action to take in state $i$), the values or utilities *v(i)* for all states in **res2** and the number of "episodes" (iterations) needed for the algorithm to converge (based on a stop criterion expressed in the change of values from one episode to the next) in **converged_at**.

2. A: The problem is that the function requires transition matrix and reward matrix explicitly as input, which are not given in the original material. One could simply use the go-function for all possible state-action pairs to retrieve both the transition and reward-matrices. This is, however, only possible as the state-action space is very limited.

3. A: Yes, one could use Q-learning (or, better $\epsilon$-greedy Q-learning), as it relies only on the output of the "go"-function and the problem specification (states and actions).

4. A: The main idea is to consider the fact that we know more about a state-action pair after having explored it than we knew before. A portion (regulated by the learning rate) of this knowledge gain is used as an update to the value of a state-action pair by spying one more step ahead from the state-action pair that is worked on. By more or less randomly exploring (walking through) the more or less complete state-action space, we update all of these values gradually for a number of such walks (episodes).

5. A: Q-learning always chooses the best (reward maximising) action a' for the computation of the new value (or follows a certain strategy like $\epsilon$-greedy to add some randomness), while SARSA follows an arbitrarily chosen but fixed policy through an entire sequence.