Lunds Tekniska Högskola
Institutionen för datavetenskap

EDAN95
HT 2018

Tillämpad Maskininlärning
Applied Machine Learning
Tentamen 2019–01–08, 08.00–13.00, MA:08

You can give your answers in English or Swedish.
You are welcome to use a combination of figures and text in your answers.
100 points, 50% needed for pass.

1 Boosting (JM): 5p

Name the steps of a boosting algorithm (e.g., AdaBoost) and shortly (in one sentence) explain their function.

2 k-Means (JM): 10p

Given the following data set: $\{(1.0, 1.0), (1.5, 2.0), (3.0, 4.0), (5.0, 7.0), (3.5, 5.0), (4.5, 5.0), (3.5, 4.5)\}$ cluster it into two clusters using k-means algorithm. You may begin with the clusters initialized to the two points in the data set that are farthest apart (but feel free to use some other seed that you deem useful). Explain what you do.

3 K-nearest neighbour (JM): 5p

Multiple choice: Please answer each of the following five sub-questions by **writing down** the letter(s) corresponding to the correct answer(s). Note that a wrong answer results in negative points, hence, it can give a negative result for the sub-question and thus cancel out positive points from other sub-questions, but the overall result for the question cannot become negative.

1. K-nearest neighbour is called a "lazy" classification algorithm because:
   (a) the amount of computations it performs is minimal;
   (b) in learning phase the teaching instances are just stored and all the computation takes place during classification;
   (c) it considers only the nearest instances and does not pay attention to all learnt instances.

2. In KNN classification the majority vote is taken to determine the class membership. In KNN regression the value of the target function is determined by:

   (a) majority vote of values for k nearest neighbours;

   (b) median of all values for the instances classified the same way as the query;

   (c) mean of the k nearest neighbour values, weighed by the distance to the query.

3. K-nearest neighbour uses:

   (a) necessarily a euclidean distance defined over the space of all features;

   (b) necessarily a euclidean distance defined over the subspace of all numerical features, leaving out the non-numerical ones;

   (c) an arbitrary distance function defined over the space of all features.

   (d) an arbitrary distance function defined over the subspace of all numerical features, leaving out the non-numerical ones.

4. K-nearest neighbour is

   (a) sensitive

   (b) not sensitive

   to the number of each class elements, so that more numerous classes get preference.

5. K-nearest neighbour is

   (a) sensitive

   (b) not sensitive

   to redundant features.

Further questions on following pages

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

4.1 Convolutional Neural Networks

In this exercise, you will build an elementary convolutional feed-forward network to classify a set of images. You will use the Keras API.

Principles. In this section, you will explain some key concepts of convolutional neural networks.

1. Describe briefly what a 2D convolution is.
2. Compute manually the convolution of the image in Table 1 with the kernel in Table 2. You will tell how to deal with the image borders.

0	0	0	0	0	0	0
0	1	2	3	4	5	6
6	5	4	3	2	1	0
0	1	2	3	4	5	6
6	5	4	3	2	1	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

Table 1: Image

1	1	1
1	1	1
1	1	1

Table 2: Kernel

3. Describe what a max-pooling operation is.
4. Apply this operation to the image resulting from the convolution. You will use a 2 × 2 mask.

Programming. In your program, you will use the CIFAR10 dataset available from Keras. Each image has a 32 × 32 dimension and represents an object among 10 categories numbered from 0 to 9: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The images have three color channels: red, green, and blue. The input shape is then (32, 32, 3), where the last dimension is to take into account the colors.

The code to load and preprocess the images is given below:

```
from keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = 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)
```

You will design a rudimentary architecture to train a model to categorize the images. You will use the Keras classes and functions with the appropriate arguments. If you are not sure about the argument order, just write them in any order and comment what you are doing in your program. You will ignore the Python imports.

1. Build a sequential feedforward network consisting of:
   - A convolutional layer with a kernel of 3×3. The depth of the output feature map – the number of kernels you will train – will be of 64;
   - A max pooling layer with a mask of 2×2;
   - A convolutional layer with a kernel of 3×3 with a depth of 128;
   - Two dense layers.

   You will use the `Sequential()`, `Conv2D()`, `MaxPooling2D()`, and `Dense()` classes. The first dense layer will have 128 nodes and you will determine the number of output nodes of the last one. You will give the activation function in each layer;

2. You need to adapt the multidimensional output of a convolutional layer to a dense layer. Tell the name of this adaptation layer and insert it in your network;

3. Compile your network. You will select an optimizer as well as a loss;

4. Fit your model;

5. Evaluate it using the appropriate evaluation function.

4.2 Analyzing a Neural Network

You will now analyze the parameters of your model. Calling the `model.summary()` function results in this table:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 64)        1728
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 128)       73728
_________________________________________________________________
flatten_1 (Flatten)          (None, 21632)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2768896
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1280
=================================================================
Total params: 2,845,632
Trainable params: 2,845,632
Non-trainable params: 0
_________________________________________________________________

Non-trainable params: 0
```

Explain the number of parameters: 1728, 73728, 2768896, and 1280.

To simplify this analysis, your teacher has removed the optional bias (intercept) from the `Conv2D()` and `Dense()` layers. This can be done with the `use_bias=False` option. Should you try to reimplement this program, you will find slightly different numbers with the default `use_bias=True`.

4.3 Precision and Recall

Table 3 shows the confusion matrix resulting from a classification experiment. In this exercise, you will clarify how to read such a matrix and compute the precision and recall for the three classes. Just write the fractions, for instance 1/3 and not 0.33.

1. Describe how to interpret the matrix in Table 3;
2. What would be a perfect matrix? Give the corresponding values for Table 3;
3. For a given class, the recall is defined as the true positives divided by the sum of the true positives and false negatives. For Class 1, how

True\Predicted	Class 1	Class 2	Class 3	Precision	Recall
Class 1	60	30	10		
Class 2	5	30	5		
Class 3	20	10	50		

Table 3: A confusion matrix

many true positives are there (correctly classified as Class 1)? And false negatives (wrongly classified as not being in Class 1)?

4. For the three classes, compute the recalls. (Write them in the form of fractions);

5. For a given class, the precision is defined as the true positives divided by the true and false positives. For Class 1, how many true positives are there? And how many false positives (wrongly classified as being in Class 1)?

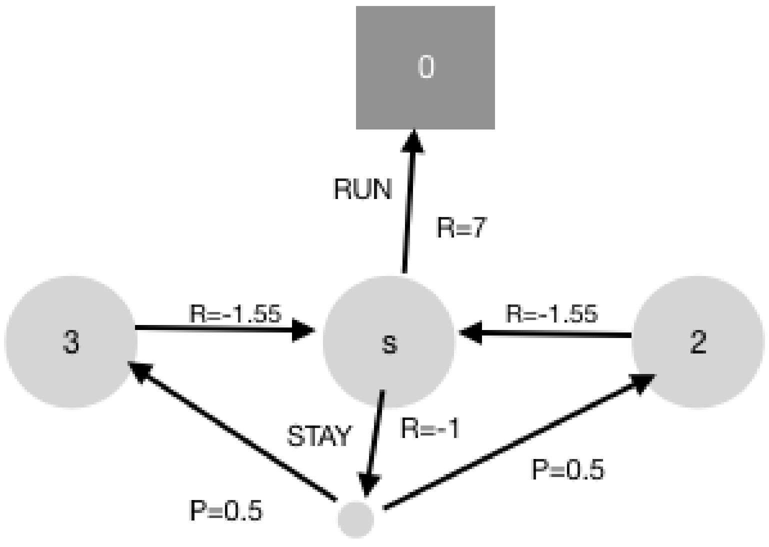
6. For the three classes, compute the precision.

7. Define what the F1 score is. (Just define it do not compute it).

Further questions on following pages

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

1. Please provide the definition of the Markov Decision Process as presented in the lecture!
2. What is the definition of the
   a) Policy?
   b) State Value function?
   c) Action value function?
3. Please write down the recursive Bellman Expectation Equation for the
   a) State value function!
   b) Action value function!
4. Consider the graph given below, assume that the given state values are optimal. Please calculate for state s the state value $v_\pi(s)$ for $\pi(s, RUN) = 0.4, \quad \pi(s, STAY) = 0.6, \quad \gamma = 0.9$!
5. Assume that the provided state values are optimal. What is the optimal state value for s, $v^*(s)$? Please provide your complete explanation by computing q^* and π^*!
6. In the graph, one of the rewards is inconsistent with the state value $v^*(s)$. Which one is it, and what should be its value?



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

You have a toy problem with a little cartoon agent trying to learn to "walk" (as shown in the lecture) with which you want to explore some reinforcement learning approaches. It lives in a world with 16 discrete states, in which one of four actions can be applied, always entailing a state change. A friend has provided you with a "go"-function for the agent, that given an action applies this action to its current state and gives you back the new state and a reward. Apparently, some states are really bad, and only very few state-action pairs would actually make the agent move forward in its cartoon world. You also got an implementation of some sort of probably suitable learning algorithm, but as your friend said when giving it to you, it is in "research-code" shape and not really documented. When you open the source file, you realise that your friend was not exaggerating—no comments is one thing, but not using any kind of telling variable names really makes this a bit of a mess. So, you try to figure out what your friend provided you with and also, what your general options are to solve your problem.

1. Analyse and explain the function *learning* given as (Python) code snippet below by relating the function and variable names calls to their counterparts of a method used within reinforcement learning shown and discussed in the lecture(s) and give the name of this method!
2. Given the problem (learn to walk) and the mentioned "go"-function, you figure that you will not be able to solve the problem by simply using the function *learning*. Why? What can you do to use the function *learning* without modifying it to solve your learning problem?
3. Is there an algorithm that can work directly with what has been given to you (i.e. the agent's "go"-function)? Which one?
4. Explain the main idea behind *Temporal Difference* learning.
5. One specific form of *Temporal Difference* learning is Q-learning, another is SARSA-learning. Explain the difference between these two (formulas given below)!

$$Q - learning : Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma max_{a'} Q(s',a') - Q(s,a) \right]$$

$$SARSA : Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a) \right]$$

The Python code snippet (function *learning*) to analyse and explain in question part 6.1 follows on the next page. Also given is a help function *argmax*, that you do not need to explain.

```
# the implementation of some function 'learning' that should
# make the agent learn
# based on some vector 'd' (of size 16)
# and some matrix 'e' (of size 16x4)
def learning(d,e):
    c = 0.95
    res = [None for a in d]
    res2 = [0 for a in d]
    converged_at = -1

    for epi in range(1000):

        for a in range(len(res)):
            res[a] =
                argmax(lambda b:
                       e[a][b] + c*res2[d[a][b]],
                       range(len(d[a])))

        sum_sq = 0.0
        for a in range(len(res2)):
            b = res[a]
            tmp = e[a][b] + c*res2[d[a][b]]
            sum_sq += (res2[a]-tmp)**2
            res2[a] = tmp

        if( math.sqrt(sum_sq) <= 1e-5):
            converged_at = epi
            break

    return res,res2,converged_at


# argmax is a help function to calculate the argmax
# over a list of arguments given by a function
def argmax(f, args):
    mi = None
    m = -1e10
    for i in args:
        v = f(i)
        if v > m:
            m = v
            mi = i
    return mi
```