Lunds Tekniska Högskola
Institutionen för datavetenskap

EDAN95

# Tillämpad Maskininlärning
# Applied Machine Learning
# Example

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 Ensemble Learning Algorithms (JM): 1+1+1+1+1 = 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.

### 1.1 Bootstrapping

is a technique of manipulating training data consisting of:

a) given a series of training set instances, training classifiers on them and averaging their output;

b) given a series of training set instances, training classifiers on them taking into account misclassified instances;

c) sampling instances with replacement.

### 1.2 Boosting

is a technique of manipulating training data consisting of:

a) given a series of training set instances, training classifiers on them and averaging their output;

b) given a series of training set instances, training classifiers on them taking into account misclassified instances;

c) sampling instances with replacement.

### 1.3 Bagging

is a technique of manipulating training data consisting of:

a) given a series of training set instances, training classifiers on them and averaging their output;

b) given a series of training set instances, training classifiers on them taking into account misclassified instances;

c) sampling instances with replacement.

### 1.4 Ensemble learning

uses:

a) only decision trees as its building blocks;

b) only recurrent neural networks as its building blocks;

c) arbitrary classifiers as its building blocks.

### 1.5 AdaBoost

is a boosting algorithm that got its name:

a) from Ada Lovelace, a computing science pioneer from 19th century;

b) from the word “Adaptive”;

c) from the first names of its creators, Adam, Daniel and Ari.

## 2 Decision Trees (JM): 6+14 = 20p

Given the Quinlan set of instances (see Table **??**), build two decision trees properly classifying all the instances. Your trees should obey the following constraints:

a) The first tree should be of depth no more than three (the root node counting as 0). Feel free to guess it, as there are no constraints put on the method of finding it.

b) The second tree should minimize Gini impurity index in every node (a.k.a. CART algorithm).

The **Gini impurity index** may be defined as follows: it is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The Gini impurity index can be computed by summing the probability $p_i$ of an item with label $i$ being chosen times the probability $\sum_{k\neq i} p_k = 1 - p_i$ of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category. Thus Gini impurity index $G$ for a set of items with $M$ classes may be found as follows. Suppose $i \in \{1, 2, \ldots, M\}$, and let $p_i$ be the fraction of items labeled with class $i$ in the set.

$$G = \sum_{i=1}^{M} p_i \sum_{k\neq i} p_k = \sum_{i=1}^{M} p_i(1-p_i) = \sum_{i=1}^{M} p_i(p_i-p_i^2) = \sum_{i=1}^{M} p_i - \sum_{i=1}^{M} p_i^2 = 1 - \sum_{i=1}^{M} p_i^2$$

Some useful numbers:

$$\frac{12}{35} \approx 0.343 \quad \frac{2}{14} \approx 0.142 \quad \frac{4}{21} \approx 0.190 \quad \frac{3}{28} \approx 0.107$$

$$\frac{18}{49} \approx 0.367 \quad \frac{6}{14} \approx 0.428 \quad \frac{7}{15} \approx 0.466$$

# 3 Neural networks (PN): 10+10+5 = 25p

## 3.1 Loss

In this section, you will compute the loss of a multiclass classification. Given an image of a flower, the goal of your classifier is to determine its type among dandelion, rose, daisy, tulip, or sunflower. We suppose that your classifier is a neural network that uses a softmax activation in the last layer.

1. Let us suppose that Table **??** shows the output of your classifier for five images. Compute the sum of the numbers in each row?[1].

| | $\hat{\mathbf{y}}$ | | | | | $y$ |
|---|---|---|---|---|---|---|
| image # | dandelion | rose | daisy | tulip | sunflower | Truth |
| 1 | 0.7 | 0.1 | 0.05 | 0.05 | 0.1 | dandelion |
| 2 | 0.05 | 0.3 | 0.1 | 0.5 | 0.05 | rose |
| 3 | 0.1 | 0.6 | 0.1 | 0.1 | 0.1 | rose |
| 4 | 0.2 | 0.1 | 0.1 | 0.5 | 0.1 | tulip |
| 5 | 0.03 | 0.03 | 0.02 | 0.02 | 0.9 | sunflower |

Table 1: The results of a classification shown in the $\hat{\mathbf{y}}$ vector with a softmax activation. The true type of the image is shown in the $y$ column

2. What do the numbers in the rows mean?

3. Given the values in $\hat{\mathbf{y}}$, for each image, what would be the result of the classification? Give the name of the predicted flower for each row.

4. What would be the accuracy of your classifier?

5. The loss usually associated with the softmax activation is the categorical cross-entropy. It is defined as

$$H(P, M) = -\frac{1}{|X|} \sum_{x \in X} P(x) \ln M(x),$$

   where $P$ is the truth, and $M$ is the prediction of the model. Using the values in Table **??**, rewrite the formula. You will represent the truth as one-hot vectors, the prediction by appropriate values, and replace the multiplication by the dot product. You will use logarithms that you will not try to compute.

6. Give the final value of the loss. You will use logarithms that you will not try to compute.

[1] The figures are imaginary.

### 3.2 Building a Neural Network

In this exercise, you will build a feed-forward network with the Keras API to classify the data in Table **??**.

| **Object** | **Attributes** | | | | **Class** |
|---|---|---|---|---|---|
| | **Outlook** | **Temperature** | **Humidity** | **Windy** | |
| 1 | Sunny | Hot | High | False | 0 |
| 2 | Sunny | Hot | High | True | 0 |
| 3 | Overcast | Hot | High | False | 1 |
| 4 | Rain | Mild | High | False | 1 |
| 5 | Rain | Cool | Normal | False | 1 |
| 6 | Rain | Cool | Normal | True | 0 |
| 7 | Overcast | Cool | Normal | True | 1 |
| 8 | Sunny | Mild | High | False | 0 |
| 9 | Sunny | Cool | Normal | False | 1 |
| 10 | Rain | Mild | Normal | False | 1 |
| 11 | Sunny | Mild | Normal | True | 1 |
| 12 | Overcast | Mild | High | True | 1 |
| 13 | Overcast | Hot | Normal | False | 1 |
| 14 | Rain | Mild | High | True | 0 |

Table 2: The Quinlan dataset

The dataset will be stored in a dictionary:

```
dataset = [{'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
            'Windy': False, 'Class': 0},
           {'Outlook': 'Sunny', 'Temperature': 'Hot', 'Humidity': 'High',
            'Windy': True, 'Class': 0},
           ...]
```

To preprocess the dataset, you will rely on this code that transforms the categorical data in an `X` matrix and `y` vector:

```
X_cat = []
y = []
# Extracts the X_cat table and y vector
for x in dataset:
    X_cat += [{key: x[key] for key in x if key != 'Class'}]
    y += [x['Class']]

# Converts the X_cat list in a numerical matrix
vect = DictVectorizer(sparse=False)
vect.fit(dataset)
```

```
X = vect.transform(X_cat)
print(X)
[[0. 1. 0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0. 1. 0. 1. 0. 1.]
 [0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 1. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 1. 0. 1. 0. 0. 1.]
 [0. 0. 1. 1. 0. 0. 1. 0. 0. 1.]
 [0. 1. 0. 0. 0. 1. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 1. 1. 0. 0. 0.]
 [0. 0. 1. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 1. 0. 0. 1. 1.]
 [0. 1. 0. 1. 0. 0. 0. 0. 1. 1.]
 [0. 0. 1. 1. 0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 1. 1.]]
```

Write the program to train a model. You will ignore the imports:

1. Build a sequential feedforward network consisting of three dense layers. You will use the `Sequential()` and `Dense()` classes. The first layer will have 50 nodes, the second one 25, and you will determine the number of nodes of the last one. You will give the activation function in each layer;

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

3. Fit your model;

4. Evaluate it.

The complete program should consist of about seven lines of Python.

### 3.3 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 #
=================================================================
dense_1 (Dense)              (None, 50)                500
_________________________________________________________________
dense_2 (Dense)              (None, 25)                1250
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 25
=================================================================
```

```
Total params: 1,775
Trainable params: 1,775
Non-trainable params: 0
```

Explain the number of parameters: 500, 1250, and 25.

To simplify this analysis, your teacher has removed the optional bias (intercept) from the `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 option: `use_bias=True`.

# 4 Bayesian Learning / Classifiers (VK): 5+5+5 = 15p

A Bayesian Learner based on the Maximum Likelihood (ML) hypothesis is in many cases a good approximation for solving a classification problem given there is enough data, however, there might be cases in which it is "not enough". In such cases, a Maximum A Posteriori (MAP)-learner can be better. On the other hand, there are situations where your data might render the MAP-learner working exactly like the ML-learner.

1. Explain the difference between the two classifiers / learners!
2. Explain why the MAP-learner is considered better in the general case!
3. Give an example for a case where the difference does not really matter!

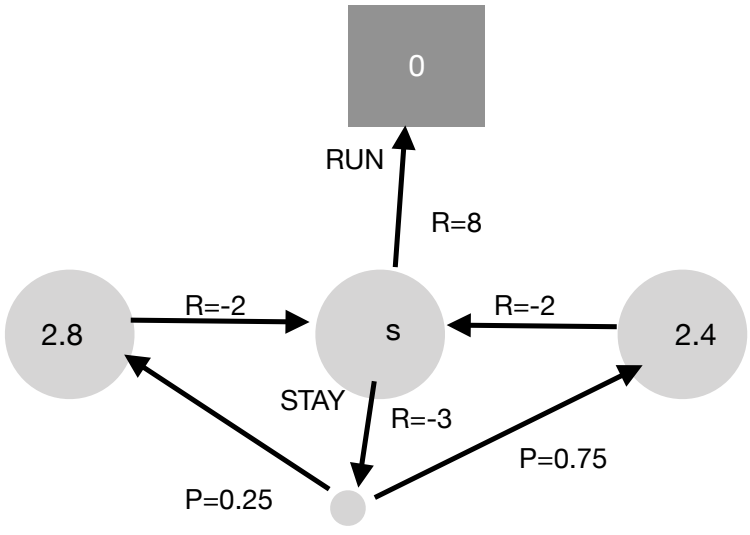
Hint: ML- and MAP-hypotheses $h_{ML}$ and $h_{MAP}$ are defined as follows, with $D$ being the observed data points (samples):

$h_{ML} = argmax_h P(D|h)$

$h_{MAP} = argmax_h P(h|D)$

## 5 Markov Decision Processes (VK): 5+5+5+5 = 20p

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 $\pi(s,a) = 0.5, \quad \gamma = 0.8$.
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 $\pi^*$



# 6 Reinforcement Learning / Q-Learning (ET): 3+4+3+5 = 15p

You have a toy problem with a little cartoon agent trying to learn to "walk" (as shown in the lecture). It lives in a world with 16 discrete states, in which one of four actions can be applied, always entailing a state change. Someone 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. However, you do not have access to the full reward- and transition-functions (well, you could compute them by applying the go-function for all states, since everything is discrete and finite, but you feel up for a little challenge). Hence, you decide to use Q-Learning to find the Q-function, i.e. the Q-Values of the state-action pairs, $Q(s, a)$. You set out to compute your Q-Values based on a *greedy policy*. Somehow, that does not really work out—the little agent seems to learn nicely not to fall over, so it avoids bad states, however, it seems stuck in useless behaviours, like lifting and setting down one leg all the time, or moving one leg back and forth without ever getting anywhere. Just when you start getting really frustrated, someone mentions something like $\epsilon$-greedy to you—obviously, there is a way to get out of this stupid behaviour!

1. What is a *greedy policy* in this context?
2. Explain what the helpful person meant! What is $\epsilon$ in this context, and how does it help?
3. Why can it be a good idea to adapt $\epsilon$ during the learning process? How can you adapt it?
4. Explain the entire update formula for $Q(s, a)$ as given below:

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