Lunds Tekniska Högskola
Institutionen för datavetenskap

EDAN95
HT 2018

Lösning: Tillämpad Maskininlärning
Solution: Applied Machine Learning
Tentamen 2019–05–02, 14.00–19.00

1 Boosting (JM): 5p

1c, 2b, 3a (+c, i.e., a by itself is enough for being accepted), 4c, 5b

2 k-Means (JM): 10p

Cluster means:
a) (1.83, 1.33), b) (3.625, 4.625), c) (5, 7)

3 Decision trees (JM): 10p

First split attribute becomes *Education.*

For branches College and HighSchool, no further split is necessary, the nodes are pure (class Medium for 5 samples in *Education*:Collegeand class Low for 4 samples in *Education*:HighSchool).

For the branch University, another split is needed, here the split can be done over either *Income* or *Status* (both result in two children with 0 impurity, 3 samples of class Low landing in either *Income*:Low or *Status*:Married, 3 of class High landing in *Income*:High or *Status*:Single).

4 Neural networks (PN): 35p

4.1 Convolutional Neural Networks

A suggestion for a solution to the programming task (originally a notebook, split into cells to see the output):

```
# Examination May 2019
## One-hot Encoding

### The corpus
corpus = [['Chrysler', 'plans', 'new', 'investments', 'in', 'Latin', 'America',
['Chrysler', 'plans', 'major', 'investments', 'in', 'Mexico', '.']]

### List of unique words
words = sorted(
    list(set([word.lower() for sentence in corpus for word in sentence])))
words

idx2word = dict(enumerate(words))
idx2word

word2idx = {v: k for (k, v) in idx2word.items()}
word2idx

zeros = [0] * len(words)
zeros

new_corpus = []
for sentence in corpus:
    new_sentence = []
    for word in sentence:
        temp = zeros.copy()
        temp[word2idx[word.lower()]] = 1
        new_sentence += [temp]
    new_corpus += [new_sentence]
new_corpus

## Categorization

from keras.datasets import reuters
from keras.preprocessing import sequence
max_features = 10000
maxlen = 500
batch_size = 32
print('Loading data...')
(input_train, y_train), (input_test, y_test) = reuters.load_data(
     num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

```
word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join(
    [reverse_word_index.get(i - 3, '?') for i in input_train[20]])
decoded_review

import numpy as np
y_train[1]
cat_nbr = max(y_train) + 1

from keras.utils.np_utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_train[1]

from keras.layers import Dense
from keras.models import Sequential
from keras.layers import Embedding
from keras.layers import LSTM, Bidirectional
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(Bidirectional(LSTM(32)))
model.add(Dense(cat_nbr, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])
model.summary()

history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=64,
                    validation_split=0.2)

results = model.evaluate(input_test, y_test)
results

import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

5 Bayesian Learning / Classifiers (VK): 20p

Sketch for a solution:

1. Please refer to the lecture slides (Lectures 11 and 12, HT2018), but in short the theorem is the same as Bayes' rule and would be in this context:

   $$P(y|\bar{x}) = \frac{P(\bar{x}|y)P(y)}{P(\bar{x})} for P(\bar{x}) \neq 0.$$

   It means that the probability for an observation belonging to class y can be computed knowing the probability (likelihood) for something belonging to class y having generated the observation (i.e., $P(\bar{x}|y)$) and the prior probability for the class P(y). The probability for making the observation can be ignored if the rule is applied in distribution form (then it essentially is absorbed by the normalising factor α).

2. Please refer to the lecture slides (see above), but it would be something like

   $$P(\bar{x} = class\, y) = argmax_y P(\bar{x}|y) \approx argmin_y ||\mu_y - \bar{x}||_2$$

   for the actual ML-estimate. The highest likelihood is approximated by the closest distance to the respective mean of the class.

3. For the likelihoods, use Gaussian normal distributions with the given means and compute the covariance matrix from the data (per class). Plug those into the ML-estimate.

4. 1) It will be a diagonal matrix, with only the variances per feature. Use the definition of the covariance to proof this. 2) Refer to the definition of the covariance (see lecture slides lecture 11 and 12, HT2018).

5. iid: "Independent and identically distributed" random variables are mutually independent and stem from the same distribution. If a respective assumption holds (at least with conditional independence), the computation of joint probabilities becomes tractable also for multivariate distributions, as it is merely the product of the probabilities / likelihoods.

6 MDPs / Reinforcement Learning (ET): 20p

1. 1) The policy π that the agent uses to find the next action a given the state s. 2) The utility (value) of the state s under the policy π.

2. As given in lecture 13, HT2018, the Bellman equation *under a fixed policy* is:

$$U^{\pi}(s) = r(s, \pi(s)) + \gamma U^{\pi}(\delta(s, \pi(s)))$$

   The general form was also accepted.

3. *Value Iteration* solves the equation iteratively:

$$U^{\pi}(s)_{k+1} = r(s, \pi(s)) + \gamma U^{\pi}_{k}(\delta(s, \pi(s)))$$

4. Transition function $\delta(s, a)$:
   $\delta(0,0) = 0$, $\delta(0,1) = 1$, $\delta(0,2) = 2$, $\delta(0,3) = 0$,
   $\delta(1,0) = 1$, $\delta(1,1) = 1$, $\delta(1,2) = 3$, $\delta(1,3) = 0$,
   $\delta(2,0) = 0$, $\delta(2,1) = 3$, $\delta(2,2) = 2$, $\delta(2,3) = 2$,
   ($\delta(3,0) = 1$, $\delta(3,1) = 3$, $\delta(3,2) = 3$, $\delta(3,3) = 2$, is not necessary, but can be given)

   Reward function $r(s, a)$:

$$r(s,a) = \begin{cases} 1 & for\,(s,a) \in \{(1,2),(2,1)[\text{and (3, 1), (3, 2) if actions in 3 are still considered}]\} \\ 0 & \text{for all other } (s,a) \end{cases}$$

5. See lecture slides lecture 13 and 14, HT2018, on Policy Iteration (pseudocode is given in lecture slides)

6. Q-learning (assign a value to each state-action pair, update iteratively until convergence, assume preferably an ϵ-greedy policy to balance exploitation vs exploration).