Brief answers to the Applied Machine Learning exam of August

2018. Some of these are just solution sketches and your own solutions would

need to be more extensive than this.

\*\*\* PART 1 \*\*\*

\* Q1 \*

(a)

- This seems like a typical classification problem based on a small

set of features, some of which are numerical and some are

categorical.

- Encode the features as a matrix, for instance by using a

DictVectorizer or by using dummy variables in Pandas

- Select one or more classification algorithms. I would probably first

try gradient boosting and random forests with this type of data, but

it's useful to try other algorithms as well, such as various linear

classifiers and neural networks.

- Ideally, we do a careful model selection to determine which learning

algorithm to use, as well as tuning hyperparameter. We use

cross-validation or use a validation set.

- Reserve a part of the data as a test set. Typical classification

problem, so the accuracy seems to be the most obvious evaluation

metric. Alternatively, precision/recall for each of the classes.

[- About training: the question is not very explicit about the

collection of the training data, but there seems to be an existing

dataset that can be used for training. Ideally, this dataset should be

collected under circumstances similar to those in which the classifier

is intended to be applied.]

(b)

- a convolutional neural network

\* Q2 \*

There could be several reasons, including

- We already have a very good model of the domain; why guess when we

already know?

- The development of a dataset for training would be more costly than

developing a rule-based predictive system.

- In a safety-critical system where a mistake is deadly. In practice,

it is rare that ML models achieve a perfect accuracy.

\* Q3 \*

- DictVectorizer: convert a dataset into a matrix. Before this

conversion, the dataset is represented as a set of named attributes.

- StandardScaler: subtract the mean from each feature and divide by

the standard deviation; this is done to make learning easier for

certain types of learning algorithms that have problems when features

differ in magnitude. Actually, this step is useless with a decision

tree.

- SelectKBest: rank the features according to some "usefulness"

metric, and remove all features except the 100 top-ranked.

- DecisionTreeRegressor: the model that will make a numerical

prediction based on the features from the previous step. In this case,

as is obvious from the name, this predictor is a decision tree.

\* Q4 \*

(a)

Accuracy = (945+20)/1000 = 0.965

Precision = 20/(20+30) = 0.40, Recall = 20/(20+5) = 0.80

(b)

The majority class is '-', so this is what this classifier will always

predict. To get the table for this classifier, move over the 20 and 30

values to the right column. Then we get

Accuracy = 975/1000 = 0.975

Recall = 0/25 = 0

The precision is 0/0 (top left cell, divided by sum of left column)

which is undefined but is sometimes conventionally set to 1.

\* Q5 \*

Because the "residence" feature has a stronger statistical association

with the output "has car?" than the other features. Just by looking at

the residence features, we are able to predict the output correctly in

all cases except one. This answer is actually enough for a full score,

but it's of course even better to compute some metric that formalizes

this idea, such as the majority score or the information gain.

\* Q6 \*

(a)

Seems like a straightforward case for collaborative filtering with

explicit feedback. This can be solved using a simple matrix-based

approach, as we saw in the lecture.

(b)

A CF solution of this kind works best when the sets of users and items

(music albums in this case) are stable and there aren't many users and

items entering and leaving the system frequently. We also need to

assume that each user's taste is fairly static.

\*\*\* PART 2 \*\*\*

\* Q7 \*

(a)

I don't answer the "do you agree" part here, and what we consider

interpretable is probably a matter of taste. However, it is true that

it can be practically difficult to understand what is the exact

function of some part of a neural network. (For CNNs, there are

various algorithms to visualize what is going on in different parts of

the model.)

In decision trees, by contrast, it is easy to see what the model

"represents" (in terms of a set of decision rules) and also why a

particular instance was classified in a certain way.

In linear models, each feature is associated with its own weight, which

intuitively can be interpreted as a measure of how much this

particular feature will affect the output of the classifier or

regressor.

[Arguably, decision trees and linear models when the set of features

is large will also be very complex and hard to interpret.]

(b)

The objective function in NNs have multiple local minima, which means

that the result of the training will be different depending on how we

initialize. For reproducibility, e.g. if we want that the numbers in

our tables in a research paper can be recreated, it is useful to

control the initialization in a well-defined way, for instance by

specifying the random seed. [Side note: this is often very complicated

when GPUs and multithreaded algorithms are involved.]

\* Q8 \*

(a)

To reduce the risk of overfitting by forcing the model to be "more simple"

according to some notion of simplicity.

(b)

Here, I use the term "regularization" in a broad sense, meaning any

method to mitigate overfitting.

- a regularizer added to the objective, such as L2 or L1

- early stopping: terminate training when validation loss does not improve

- dropout: randomly "hide" features or hidden units during training

\* Q9 \*

The resulting classifier will have the form

score(x) = sum\_i w\_i\*x / N

where w\_i is the weight vector for perceptron model i, and N is the

number of models in the ensemble. This ensemble is in itself a linear

model, so it is not more \*expressive\* than any single perceptron, and

it won't be able to solve e.g. the XOR problem.

How can we know the ensemble is in itself a linear model? We can

create an equivalent linear classifier by computing the average of all

weight vectors. [Formally speaking, we rely on the bilinearity of the

dot product operation, but it's not necessary to mention this explicitly.]

We can't give guarantees that the ensemble will be better, but we will

often see an improvement in practice over a single perceptron. For the

perceptron specifically, there is often a bit of overfitting towards

the last instances it considered during training, so the randomness

introduced by the bagging algorithm is quite likely to be beneficial.

\* Q10 \*

See the "clarification document" for Assignment 4

http://www.cse.chalmers.se/~richajo/dit866/files/a4\_clarification.pdf

\* Q11 \*

(a)

1. Treat it as a classification problem, for instance with six

categories A-F for the ECTS grade.

2. Treat it as a regression problem. Then we need to map each

category to a number, such as 1-6 for the A-F grading scale.

The drawback of treating this as classification is that we discard the

ordering information about the categories. (When training, if

considering an instance labeled as A, it is as bad to predict a B as

predicting an F.)

The drawback of treating this as regression is that the choice of a

numerical scale will influence the regression model. It may be

necessary to think carefully of how this mapping is done. For most

types of regressors, the output will be a floating point number, so we

also need to round it to an integer when making predictions.

(b)

As in (a), we could approach the evaluation task as a classifier

evaluation (e.g. accuracy) or regression evaluation (MSE, MAE, R2

etc.) Similar issues arise as in (a).

\* Q12 \*

(a)

A simplistic answer would be to propose the accuracy score, because

this is a classification task. However, it is probably better to use a

score that takes the hierarchy into account. For instance, if the true

label of the document is "Jazz music", and the classifier predicts

"Classical music", this is probably better than if would have predicted

"Football".

For example, for each instance we could compute the distance (number

of steps) in the hierarchy between the true category and the predicted

category. The distance between "Jazz" and "Classical" would be 2, for

instance. Then we could compute the mean of these distance scores for

all instances.

(b)

1. "Flat" classifier. We disregard the hierarchy and train any

multiclass classifier using the 325,000 categories.

2. Hierarchical classifier where we train one separate classifier for

each inner node in the tree. For instance, we train a separate

classifier for documents about music, which will decide if a document

is about jazz, classical, world, etc. At prediction time, we would

start at the root and apply the classifiers hierarchically.

The drawback of (1) is that we disregard the hierarchy during

training. (This is similar to the discussion about ordinal regression

above.) Also, for some classifiers it may be computationally difficult

to handle 325,000 categories. For instance, in a linear model or

neural network, we would need 325,000 weight vectors.

I can't think of substantial drawbacks of (2) compared to (1). It will

likely be much faster, although again this depends on what type of

classifier we use. It is likely that the amount of training data will

be quite small for the sub-classifiers that correspond to very

narrow and specific categories (there will be fewer jazz documents

than music documents, etc.) but this is also a problem for the "flat"

classifier.