Brief answers to the Applied Machine Learning exam of March 2019. 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 fairly standard binary classification problem.

We can think of several features that might be useful here. Here is a non-exhaustive list of examples:
- grade point average, but this would probably have to be standardized or normalized so that different grading systems could be compared;
- whether the student applies from the same university or not;
- whether the student is studying full time or has a job on the side;
- whether the student will need to pay a tuition fee or not.

In terms of the implementation, the solution will probably look something like this:
- 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.

(b)
You would have to decide whether to compute an accuracy or a precision/recall score w.r.t. one of the classes. Let's do both:

accuracy = (13278+4874)/(13278+4874+1655+2298) = 0.82

precision and recall for the no-show class:

precision = 4874/(4874+1655) = 0.75
recall = 4874/(4874+2298) = 0.68

Arguably, you could also apply less widely used metrics such as sensitivity/specificity scores, etc.

* Q2 *
(a)
NNs are more powerful in terms of what kinds of mathematical relationships can be expressed. If the relationship between input and output variables is highly nonlinear, the linear model is likely to perform poorly.

(b)
Most types of linear models train faster and have unique solutions, and don't get stuck in local minima. They also have fewer "bells and whistles" and don't require as much tuning.

(c)
A linear model can be seen as a neural network with no hidden layer; conversely, we can say that a neural network consists of a sequence of nonlinear data transformations before a linear model. Any of the loss functions we use to train NN regression models (most commonly the squared error loss) can be used to train a linear model and vice versa.

* Q3 *
(a)

Probably, the company needs to collect a large collection of images from the cameras installed on their ships. They then need to hire annotators who go through these images and then mark and label the relevant objects. Ideally, the circumstances under which we collect data should be similar to the deployment scenario, and we also like to see data collected in different conditions (e.g. different light and weather conditions).

It seems likely that this dataset will be very imbalanced since the number of objects encountered at sea will be small. If the number of occurrences of the various object types is too small, it may be necessary to collect in a more "targeted" fashion and make sure that we have a diversity of examples and not just a lot of pictures of the blue sea.

(b)
We should probably carry out a double annotation and then measure to what extent annotators agree. We would say that two annotators "agree" about an object if they have selected the same label (e.g. "iceberg") and the bounding boxes are reasonably close (as defined by some threshold). This notion of agreement can then be used to compute an inter-rater agreement metric such as the kappa score or inter-rater precision/recall.

* Q4 *
This is quite common in practice and the reason is that the data distribution evolves over time. We may see changes in the input distribution (we collect individuals differently as time goes by) or in the input-output relationship (we label individuals differently as time goes by). Both issues will cause a machine learning model's performance to degrade over time unless we retrain continually.

* Q5 *
(a)
(Probably easier to draw than to explain...)
For all x values less than 1.8, the output is 1.4; for x values between 1.8 and 2.6, the output is 0.3; for x values greater than 2.6, the output is -0.2

(b)
Typically, if we allow the tree to grow deeper, the output curve will have a more complex shape with many more "jumps". But it will still be piecewise constant, as in the original model.

(c)
Random forests use *instance bagging* while training. This means that if we have a training set X, Y, each of the members of the ensemble (two trees in this case) will have been trained on a training set that was sampled (with replacement) from the original training set.

The purpose of using different training sets for each of the component trees is to encourage them to be fairly different, which will make the ensemble more robust and improve generalization.

* Q6 *
(a)
This seems like a convolutional neural network trained with a regression loss would be useful. Alternatively, "bin" the age variable into groups and train it as a classifier with a cross-entropy loss.

(b)
Assuming we are treating this as a regression problem, we should probably evaluate it using any of the usual regression evaluation protocols: mean squared error, mean absolute error, or R^2.

(c)
From the description, it seems that the Migration Agency buys this system "off the shelf", including the model that you trained on the dataset that you collected. The reason why the Migration Agency bought the system is apparently that they want to determine the age of young

asylum seekers, supposedly because they want to determine whether an asylum seeker is younger or older than some threshold age (e.g. 18 years).

This seems like a naive and problematic application of machine learning that is likely to cause problems. Issues that seem problematic include:
- It is not clear that the training set that you collected is representative of the data the model will be exposed to at deployment time: the age distributions may be different between the datasets, and the socioeconomic conditions and range of ethnicities may differ between the datasets in a way that makes it hard for the model to generalize well.
- While the model is probably quite good at distinguishing young people from older people, it seems unlikely that it will reliably distinguish people who are younger than or older than 18. The model's prediction should probably not be used directly, and it would probably be more useful to look at some sort of confidence measure.

*** PART 2 ***

* Q7 *
To keep the discussion simple, let's discuss a regression model with one input variable x.

Each of the values in the hidden layer will be computed as

ReLU(w*x + b)

where, as usual, w is a weight and b is a threshold (bias) term. If we expand the ReLU function, we get this pseudocode:

```
if w*x+b >= 0:
  the value is w*x+b
else:
  the value is 0
```

That is, depending on the value of x, the output of this hidden unit will either be a linear function of x, or a constant 0. We can note that if x is not exactly at the threshold, the function is linear in a small region around x.

The output of the complete regression model is a linear combination of the values computed in the hidden layers, each of which is computed as above. Recall that a linear combination of linear functions is itself linear, so again we can note that for an x value that is not exactly at one of the thresholds, the function will be linear in a small region around x.

To summarize, this explains why a NN with ReLU activations in the hidden layer will be piecewise linear.

Conversely, other common activation functions (sigmoid, tanh etc) are not locally linear in the way that a ReLU is: they have a smoothly curved shape. So the full NN in this case will not be piecewise linear as a ReLU-based NN.

* Q8 *
(This question is less relevant in the 2020 course since we haven't taught recommendation systems this year.)

The first issue -- that "lefties" will be exposed primarily to "lefty" news and vice versa -- seems to arise simply because that is what we expect recommendation systems to do. We could see this issue in content-based recommenders (you'll be recommended news articles from the same sources or with a similar content as those you previously read), as well as in collaborative filtering systems (you'll be recommended news articles that were read by people with a similar reading history to you).

The second issue -- the "extremization" of recommendations -- probably requires a bit of speculation about how the recommendation system selects what to show to you. For instance, it could be that the recommendation is based on a confidence score: if you tend to consume news with a left-wing political orientation, you would then be shown other news articles that the system is most confident would be suitable to your orientation. Presumably, those articles will have a very strong and unambiguous political stance.

In both of these scenarios, we can see that there is an interaction between the user population and the news recommendation system. It is also possible that the users themselves will be affected by the recommendations and that their political views will evolve as a result, which would reinforce these tendencies even more.

To summarize, these effects seem to be explainable by an interaction between the nature of the recommender systems and the changing preferences of the user population as a result of these systems.

It can be mentioned that there is a lively debate in social media studies whether it is actually true that we see these effects, and whether they have any actual effect on users or not: maybe users will form "bubbles" regardless of whether there is a recommendation system involved?

* Q9 *
(a)
This is the result of the first convolutional filter in the first layer applied to the top left corner of the image.

For instance, let's assume we have a 5x5 convolutional filter. For each pixel, we have 3 color values (R, G, B). We compute the value at this position in the feature map by applying a dot-product-like operation between the filter's weights and the pixel values in the top left 5x5 corner of the image. We then apply the ReLU function to the result of this dot product.

(b)
The pooling layers we use here will reduce the size of a feature map: for each 2x2 region, the pooling operation selects the maximal value observed in this region. So for each position in a feature map after a pooling operation, a 2x2 region in the preceding feature map was used to compute it.

The convolutional layers have the size 5x5, so they will reduce a region of size N to a feature map of size N-4. (Conversely, a region of size NxN in the output feature map has been computed from a region of size N+4 in the input feature map.)

- The top left position in the last feature map was computed by applying pooling to a 2x2 region in the preceding feature map;
- The 2x2 region was computed by applying a convolution to a 6x6 region;
- The 6x6 region was computed by applying pooling to a 12x12 region;
- The 12x12 region was computed by applying a convolution to a 16x16 region in the original image.

So, to summarize, the top left position in the last feature map is the result of applying a series of operations to the top left 16x16 region in the original image.

* Q10 *
(a)
If the predicted value yhat=w*x is "close enough" to the true value y, meaning that it is within epsilon from the true value, then the loss is zero. If the deviation from the true y is greater than epsilon, the loss grows as a linear function of the error.

(b) and (c) are more or less equivalent to Programming Assignment 2B, with a new type of loss function. We just write down the pseudocode here, and for a detailed explanation, please refer to the

clarification document for PA 2B.

(b)
We use a solution based on stochastic gradient descent with one
training instance at a time:

```
initialize w (for instance as all zeros)
for e in 1, ..., NumberOfEpochs:
    for x, y in TrainingSet:
         yhat = w*x
         if yhat-y > epsilon:
             w = w - eta*x
         else if y-yhat > epsilon:
             w = w + eta*x
```

The learning rate eta will either be a user-defined parameter or use
some sort of cooldown scheme as in the Pegasos algorithm we used in PA 2B.

(c)
We add a L_2 regularizer lambda*|w|^2 / 2 to the objective. The
pseudocode becomes even more similar to PA 2B, because we now have a "weight
decay" as in that assignment:

```
initialize w (for instance as all zeros)
for e in 1, ..., NumberOfEpochs:
    for x, y in TrainingSet:
         yhat = w*x
         w = (1-lambda*eta)*w
         if yhat-y > epsilon:
             w = w - eta*x
         else if y-yhat > epsilon:
             w = w + eta*x
```

* Q11 *
The question does not ask us to do this for any type of classifier, so
we assume that we can use whatever classifier we want. Our solution
here will be applicable to a linear model or a neural network, while a
tree-based model would require some other type of solution.

The simplest solution would be something like the following:

For a given instance x,
1) compute the classifier's output scores (either raw classification
scores w*x or the probabilities from a softmax output) for *all*
classes
2) remove the classes that are not included in LegalOutputs(x)
3) return the class that has the highest score for this instance

This solution is a bit inefficient if LegalOutputs typically
returns a small subset and the total set of classes is very large,
which is quite common in this type of classification problem.
In this case, a more well-designed solution will use a lookup table
that selects the weight vectors for the set of classes in
LegalOutputs(x). This will require a smaller number of multiplications.

Anyway, both solutions will be counted as passable.

* Q12 *
This question is inspired by the work of Fredrik Johansson and the
second figure is taken directly from his PhD thesis:

http://publications.lib.chalmers.se/records/fulltext/245375/245375.pdf

(a)
There are actually a number of machine learning methods that are
designed precisely for this type of problem, including graph kernels
and graph convolutional networks. For instance, here is a paper on
graph-based CNNs:

http://proceedings.mlr.press/v48/niepert16.pdf

and here is a survey of graph kernels:

https://arxiv.org/pdf/1903.11835.pdf

However, we don't have to use such a specialized solution and we can apply the tools we have covered in the course.

Maybe the simplest solution is to apply a "bag of words" feature representation to the SMILES string or molecular formula, and then a linear classifier? Basically, this representation will count the frequencies of the various atom types. The accuracy of this approach will obviously depend on the problem at hand, but I would probably believe that it won't work fantastically well for most molecule-based classification tasks. Probably, it will work better if we generalize it so that it counts pairs or triples of atoms instead.

We may also design features based on various graph-theoretic properties of the molecule; for instance, the number and sizes of strongly connected components, etc. Here is an example of such an approach:

http://www.cs.rpi.edu/~zaki/PaperDir/SADMJ12.pdf

A third approach could be to apply a convolutional or recurrent neural network (e.g. a LSTM) to the SMILES string.

Several students also proposed to treat this as an image classification problem, applying a CNN to an image representation of the molecule (either the ball-and-stick model or the skeletal formula). In such an approach, it seems that we need to take into account that we may rotate the molecule in different ways.

The question asks us to "argue for which of them you prefer". What will work best is of course an empirical question, but it seems that the approaches based on graph properties or SMILES strings will at least be computationally more efficient than approaches based on representing the molecule as an image.

(b)
Most of the literature in graph classification (including the papers linked above) seems to have molecule classification (of different types) as the main use case.

Still, we can probably think of different types of graph classification tasks where the object of classification is not a molecule. For instance, a couple of years ago I supervised a thesis at Chalmers where students applied clustering to radio access networks:

https://odr.chalmers.se/bitstream/20.500.12380/251547/1/251547.pdf

Electrical circuits can also be viewed as graphs and we could probably think of several classification tasks for circuits.