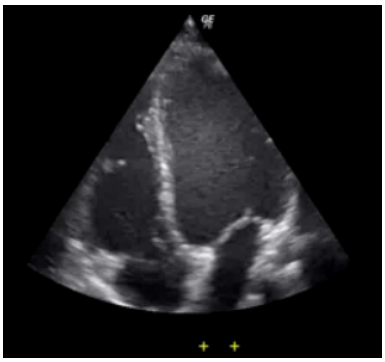
Solution to the exam
DIT866/DAT340: Applied Machine Learning, March 14–15, 2021

Question 1 of 12: Ultrasound images of the heart (12 points)

At a hospital, we would like to develop a machine learning system that processes ultrasound images of the heart in patients that are investigated for cardiovascular diseases. The figure below shows an example of such an image.[1]



The goal here is to develop a system that considers the *size* and *mobility* of the heart. In particular, with respect to the size aspect, the system should determine whether the heart has *normal size*, *slightly increased size*, *moderately increased size*, or *significantly increased size*. Similarly, with respect to the mobility, we'd like to determine whether the heart has *normal mobility*, *slightly reduced mobility*, *moderately reduced mobility*, or *significantly reduced mobility*.

The input to this system consists of raw pixel data. Although a wide range of measurements will typically be considered in a realistic medical scenario, we will only consider the image data here. Furthermore, in a real-world application we would consider a short "film" but here we will assume that we will just consider single static images.

(a, 6p) Explain how you would implement a machine learning model that can handle these prediction tasks. You don't need to show Python code or give exact values of hyperparameters, but please give a description of the system and explain all steps you would carry out when developing it.

(b, 2p) What metric(s) should we use to evaluate a system like this?

(c, 4p) Please express your opinion regarding any ethical issues you think are relevant in this context.

Solution.

(a) The first step is obviously to make sure we have a dataset. By whatever means available, we need to collect a reasonably large set of images. In the best case, this set should be representative for the population where we want to apply the finished product, but in many practical scenarios we will need to be happy with whatever data we can get: for instance, images from the patients who have previously been treated at this hospital. The availability

[1] https://commons.wikimedia.org/wiki/File:Ultrasound_of_human_heart_apical_4-cahmber_view.gif

of data may be affected by legal or ethical considerations: see the discussion in (c).

For an approach based on supervised ML, we need annotation for both categorization tasks. Let's assume we have some annotators with medical training. They annotate a "reasonable" amount of images: probably a few thousand at least. If we are very well-equipped on the annotation side, we may choose to double-annotate a subset of the images to improve the reliability of the annotation (e.g. by measuring inter-annotator agreement or to re-annotate conflicting annotations).

We will probably not need to do much data massaging here. It is common to normalize pixel values to a range [0,1] when working with neural networks for images, but that is not strictly necessary. It can also simplify the implementation if we standardize the images so that they all have the same dimensions.

Technically, we should use an image-oriented architecture such as a CNN. We'll probably consider the typical CNN bag of tricks and explore different architectures including various types of convolution and pooling layers, normalization and so on. We may try various data augmentation techniques typically used in CNNs such as varying contrast and light, rotating, adding noise etc. We may also try to utilize a pre-trained CNN (as a feature extractor or with fine-tuning), although it may be the case here that an ImageNet-trained model won't be very useful for this task. We may try different types of regularization including dropout and L_2 weight decay.

We are predicting two outputs here, and it is an empirical question whether it will work best to use two completely separate predictors here, or some sort of joint prediction model that would e.g. share the convolutional layers but use different output layers. Both types of solutions will be counted as acceptable here.

As discussed in (b) later, this is formally an *ordinal regression* task: we have categories with a natural order. This type of task can be seen as "in between" classification and regression. In practice, we will often implement a predictor for this type of task by pretending that it's either classification or regression. Most commonly we'll just see it as a classification task, and not really consider the natural order of the categories. This means that for the size and mobility, we'll have output softmax layers with a dimensionality of four, and try to minimize the cross-entropy loss. (There are also some specialized architectures for ordinal regression, but that's not really necessary in this discussion.)

On the methodological side, we should probably split off a validation and a test set. The validation set will be used during model selection to optimize all hyperparameters. We'll also consider the evaluation scores (see (b) below) on the validation set to avoid overfitting by terminating training when the evaluation scores seem to have peaked. The test set will be used for final evaluations after all tuning has been carried out.

The hyperparameters we'll need to adjust are various parameters defined by our architecture, mainly the number of filters in the layers, size of convolutional filters and pooling regions, etc. Then there will be settings for the learning rate in SGD, Adam etc., batch sizes and so on, and also for the regularization (dropout, L_2) and data augmentation.

(b) As mentioned above, these two prediction tasks are ordinal regression tasks: we have discrete categories, but these categories are ordered. Again, we may design our evaluation

protocol to reflect this. One approach would be to assign numerical values to the categories (e.g. 0 to 3) and then compute a regression evaluation metric (MSE, MAE, R^2). Or we might again choose to treat this as a classification task without considering the order of the categories, but this would then have the drawback that all errors are treated equally: that is, if the true category is "severely reduced mobility", the model would be punished equally for predicting "normal" as for predicting "moderately reduced mobility".

An ideal solution for (b) would at least mention that there is an order among the categories even if a classification metric is chosen in the end, but since this is the first question we will accept solutions that propose regular classification or regression metrics.

(c) Ethical and legal aspects are of course highly important in this particular application area, and in a medical context it seems natural to start with a Hippocratic attitude of "do no harm."[2] As usual when it comes to ethical discussions in this course, we don't think there is necessarily a right answer but you have to explain how you are thinking.

The legal aspects are fairly straightforward although they may of course raise significant practical obstacles. Typically, there will be significant paperwork that needs to be completed before we can access the data and we can start the project, and the project plan will typically be screened by some ethics board. Naturally, we need to keep the dataset in a protected environment where it can only be accessed by the relevant people. It is probably good practice to anonymize the data so that as few people as possible have access to any personal details unnecessary for the project. (In this case, we are just using the images and the expert judgments anyway and we don't need any other personal or demographic information.)

There are also some ethical aspects we can keep in mind beyond those that are regulated by law. Here is a non-exhaustive list of points to keep in mind:

- Do we have a training sample that is representative of the population as a whole? Otherwise, the system may become biased so that it performs worse for some demographic groups (e.g. by gender, race, socioeconomic status, age, ...).
- How will the system be applied when we run it "live"? Patients will probably prefer to be examined by a trained human expert rather than a purely automatic system, so it is probably easier to find acceptance for this system in a semi-automatic use case than in a fully automatic one.
- Do we have a mechanism so that we can explain a decision? Otherwise, it may be more difficult for the system's decisions to be trusted by medical professionals and the patients.

Question 2 of 12: Rescaling a feature (6 points)

We are using machine learning for some classification task. We are considering two different classifiers, a random forest and a neural network. For instance, in scikit-learn we would write something like the following:

```
rf = RandomForestClassifier(max_depth=3, n_estimators=5, random_state=0)
nn = MLPClassifier(activation='tanh', hidden_layer_sizes=10, random_state=0)
```

[2] https://en.wikipedia.org/wiki/Hippocratic_Oath

One of the features is based on a length measurement in meters. Let's assume that we represent these measurements as millimeters instead: that is, we multiply the numbers by 1,000. We then retrain the classifiers on the same dataset.

(a, 2p) Why is the random forest not affected by this change?

(b, 2p) Why is the neural network affected by this change?

(c, 2p) When working with neural networks, how can we make the training more robust in this respect?

Solution.

(a) When using numerical features in decision trees, including trees in ensembles such as RF, we will only consider whether the feature value is greater than or less than a threshold. To set the threshold, we'll see how different threshold split the data and compare these splits with respect to some quality criterion (for classification, typically the information gain or the Gini impurity). Since we only care about how a feature splits data into groups, it follows that we can apply any mathematical transformation to a numerical feature that does not change the order of the instances (that is: a monotonic function) and still get the same split. The threshold value would of course be different.

(b) Let's say we train the same neural network model (that is, using the same initialization) on these two datasets. If the feature values have been multiplied by a factor of 1000, it means that we'll start the optimization at a completely different point and we'll likely end up in some other local minimum of the objective. In particular, it seems likely that the latter dataset might give us a difficulty with "saturated" tanh units and the training process might find it difficult to get unstuck.

(c) Typically, we standardize numerical features when using neural networks. A common choice is to subtract the mean and divide by the standard deviation. For instance, we might use a StandardScaler in scikit-learn.

Question 3 of 12: Anomaly detection (6 points)

We develop a logistic regression classifier to detect anomalies in some equipment, and we collect a test set to evaluate this classifier. The table below shows 20 examples from this test set. The first column shows the gold-standard label and the second the probability of the class *abnormal* according to the classifier.

abnormal	0.58
normal	0.57
normal	0.32
abnormal	0.57
normal	0.23
normal	0.20
normal	0.58
normal	0.10
normal	0.19
normal	0.04
normal	0.01
abnormal	0.75
abnormal	0.80
normal	0.05
abnormal	0.97
abnormal	0.11
abnormal	0.54
abnormal	0.45
abnormal	0.92
normal	0.58

(a, 2p) Compute the precision and recall scores of this classifier with respect to the abnormal class.

(b, 2p) Describe a scenario where it could be important to have a recall close to 1.

(c, 2p) As described above, the classifier is a logistic regression model. It uses a collection of features based on various measurements in our equipment. How can we bring the recall of this classifier closer to 1? It's OK if this change leads to a drop in precision, but a trivial classifier that always outputs *abnormal* is not acceptable here.

Solution.

(a) The question is a bit underspecified, and we'll have to translate the probability values into decisions. The default behavior of LR classifiers is to return the prediction that the classifier thinks is most likely; for a binary classification task, this means that we return "abnormal" if the probability given in the table is at least 0.5 and "normal" otherwise. In (c), we will consider other threshold values.

With this threshold, we get a precision of 7/10 and a recall of 7/9.

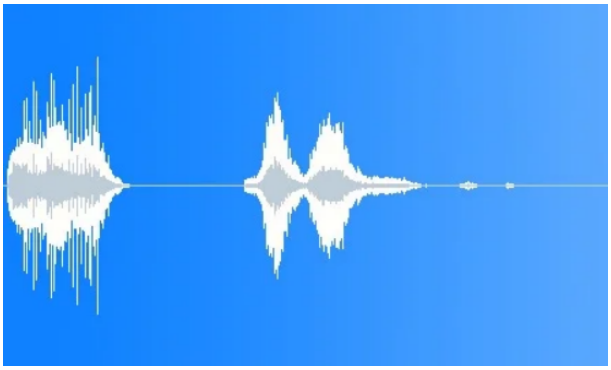
(b) Whether we consider the precision or recall more important depends on the real-world consequences of false positives or false negatives, respectively. In particular if this anomaly detection system is a part of some safety-critical system, for instance if the equipment we are considering is part of an airplane, it seems reasonable that we want the recall of anomaly detection to be very high, even at the cost of a reduced precision: it is better to carry out some redundant checks than to miss a critical error.

(c) The easiest option would be to just move the threshold. For instance, if we set the threshold to 0.1, we instead get a recall of 1.0 with this test set, but we'll pay the price in the precision

score which has dropped to 0.56. Alternatively, we might retrain the classifier with a loss function that gives more weight to the ”abnormal” class.

Question 4 of 12: Events in sound recordings (4 points)

We would like to develop a system that detects different types of events in sound recordings of people sleeping. We collect a number of recordings at 44.1 kHz. Such recordings may be visualized, as exemplified in the figure below.



In this particular segment, the person first snores and then takes two breaths.

The goal for the machine learning system is to detect the occurrence of a number of event types such as snoring, breathing, coughing, etc. For each event detected in the audio signal, we'd like the system to determine the event type (e.g. snoring) and the start and end points of the event.

(a, 2p) What kind of manual annotation do we need to create in order to build a training set for a supervised machine learning system that carries out this event detection task?

(b, 2p) If we develop a system to find such events in the recordings, how should we evaluate this system?

Solution.

This question is inspired by the company Sleep Cycle (`https://www.sleepcycle.com/`).

(a) More or less what the text already describes. For each recording, the annotator would mark the start and end points of time segments and assign an event type label. (An annotation UI might show a visualization such as the one in the figure, and allow the annotator to select the relevant time intervals and then label them.)

(b) Different solutions could be acceptable, depending on how we think about this task. The sensible choice would probably be to compute precision and recall scores at the event level:

P = number of correctly detected events / number of proposed events

R = number of correctly detected events / number of true events

By ”correctly detected” we would probably mean that a proposed event has the right type and

that the proposed start and end points are "close enough" to the gold standard, measured in some way.

Question 5 of 12: Feature selection (4 points)

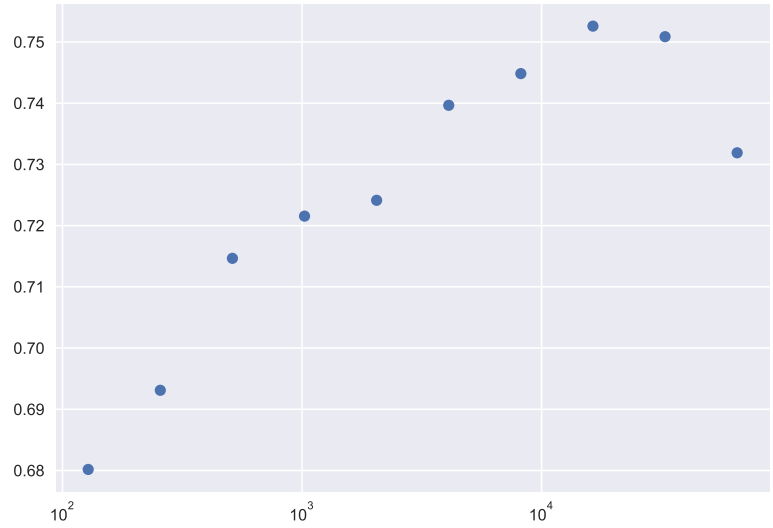
We have a training set for a supervised classification task. This training set consists of about 1,000 instances, where each input x consists of about 65,000 features and the output is a binary value. There is also a corresponding test set.

We use this data to train and evaluate a classifier using scikit-learn. Here is the code:

```
from sklearn.feature_selection import SelectKBest
from sklearn.svm import LinearSVC
pipeline = make_pipeline(SelectKBest(k=...), LinearSVC())
pipeline.fit(X, Y)

from sklearn.metrics import accuracy_score
acc = accuracy_score(Ytest, pipeline.predict(Xtest))
```

We explore some different values of the hyperparameter `k` in `SelectKBest` and compute the accuracy for each value. The figure below shows the result.



Explain the shape of the curve in this figure.

Solution.

In feature selection in general, we have a tradeoff between "throwing out the baby with the bathwater" and including too much redundant information that makes it harder to discern the meaningful patterns. When `k` is too low, the accuracy is lower because too many useful features are discarded. When it is too high, we run a higher risk of overfitting and it is harder for the learning algorithm to focus on the useful information.

The exact shape of the curve will depend on the problem we're trying to solve and the type of ML model we're using. (This curve has been computed using the student-annotated data

from Assignment 3 from 2020, where we wanted to distinguish pro-Brexit from anti-Brexit comments. We are using word n-gram features with n in [1, 3].)

Question 6 of 12: Data augmentation for images (4 points)

When training machine learning models for images, it is common to apply *data augmentation* techniques where we carry out some sort of modification to the images we have in the training set. For instance, we might increase the contrast of an image, mirror it along the x axis, or rotate it. In the Keras library, there are several types of data augmentation techniques available in the utility `ImageDataGenerator`, as shown below.

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False, samplewise_center=False,
    featurewise_std_normalization=False, samplewise_std_normalization=False,
    zca_whitening=False, zca_epsilon=1e-06, rotation_range=0, width_shift_range=0.0,
    height_shift_range=0.0, brightness_range=None, shear_range=0.0, zoom_range=0.0,
    channel_shift_range=0.0, fill_mode='nearest', cval=0.0,
    horizontal_flip=False, vertical_flip=False, rescale=None,
    preprocessing_function=None, data_format=None, validation_split=0.0, dtype=None
)
```

(a, 1p) What is the reason we use data augmentation for image-based ML tasks?

(b, 3p) What are the tradeoffs we need to think of when using this technique? For instance, what happens if we set the `rotation_range` in Keras to a value that is too low? What if it is too high?

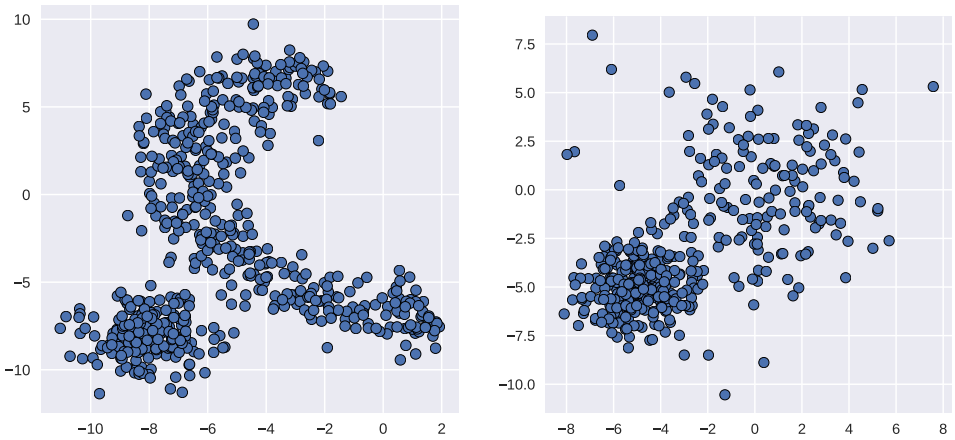
Solution.

(a) The key point is that by exposing the model to more variation in the training data, we may reduce overfitting.

(b) Obviously, if the modifications are insignificant, we won't see much benefit since the modified images are mostly the same as the original. On the other hand, if we modify the images too much, they might become unusable. In the worst case, the modification may affect the desired output label: for instance, if we would want to distinguish images taken during they day from those taken at night, augmentation techniques that affect the light of the image may be risky.

Question 7 of 12: Clustering (8 points)

(a, 1p) We have a couple of datasets and we want to apply some sort of clustering to find natural groups in the data. These datasets are two-dimensional and here are the scatterplots.



What do you think an ideal clustering algorithm should produce in these two cases? That is, what are the natural groups in the data in your view?

(b, 4p) Roughly, what do you think that K-means and DBSCAN would produce for these datasets? You can assume that we use the "best" hyperparameter tuning in both cases.

(c, 1p) How could you use a clustering algorithm such as K-means to cluster images? Describe what you would try to do if faced with this task.

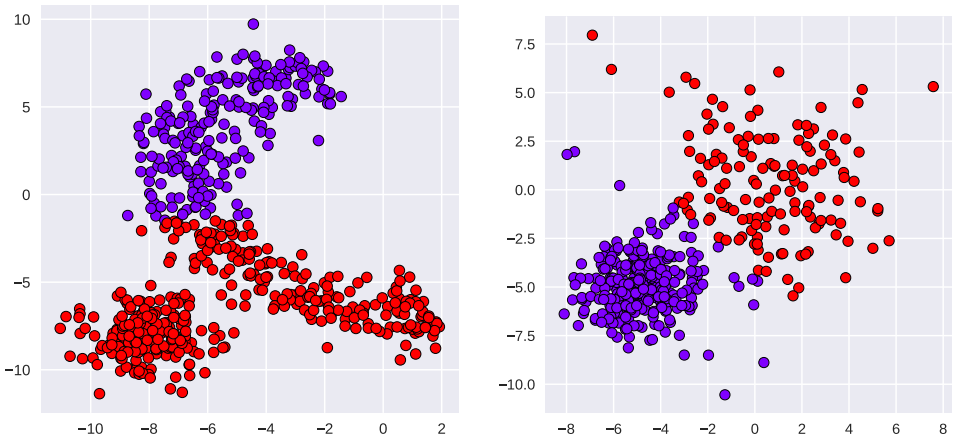
(d, 2p) We apply a clustering algorithm to a set of images of cars and bicycles, and now we wonder whether the clustering has automatically discovered how to distinguish the cars from the bicycles. How would you measure to what extent this is true?

Solution.

(a) As usual in clustering, the answer is somewhat subjective. In both cases, there should probably be two clusters. In the first case, the main groups are the "ball" at the bottom left and the bigger "C" shape, while in the second case, there is one "ball" of tightly packed points at the bottom left and another less distinct ball at the upper right.

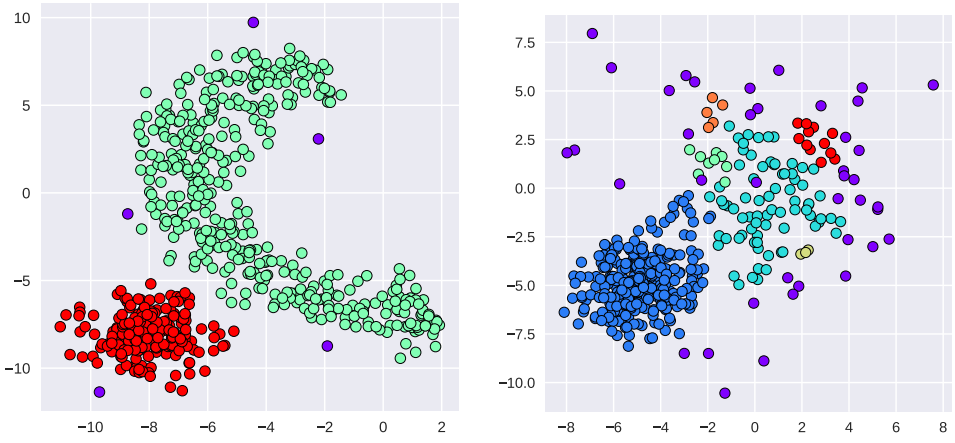
(b) For K-means, we assume that we use the ideal setting of $K = 2$. Here, we have a difficulty in the first case. K-means implicitly assumes equally sized ball-shaped clusters, and if we have an irregularly shaped group of data it will be hard for K-means to find it, at least if it is near another group. So in this case, the C-shaped group will not be identified as a distinct cluster by K-means, because some of the points in this group will be quite close to the "ball."

The second dataset isn't a problem for K-means and in this case the natural groups are both ball-shaped.



DBSCAN instead connect points that are near each other, as defined by some distance threshold, and the shapes of the clusters don't matter. In the first dataset, the points are packed fairly tightly and the natural groups are also quite well separated. So in this case, assuming that the distance threshold has been set correctly, DBSCAN can distinguish the C-shaped and the ball-shaped clusters.

The second dataset will be more challenging for DBSCAN, because the second and less distinct group is not completely separated from the first group. If we set the distance threshold to a large value, we will just get one big cluster, while if the threshold is low, we will get many small clusters. The figure shows the best result I could get, and as you can see, we couldn't really find the second "ball" as a single cluster here.



(c) In the simplest case, we may treat the pixel values as any kind of numerical features, and just feed the images directly as feature vectors into the K-means clustering algorithm. (The dimensions of the images would need to be standardized so that the feature vectors are of the same size.) This might not work so well, depending as usual on what we want to do. Probably, in this case we will group images based on similarity of color distributions. If this is what we actually want, we may transform the image into something like a color histogram.

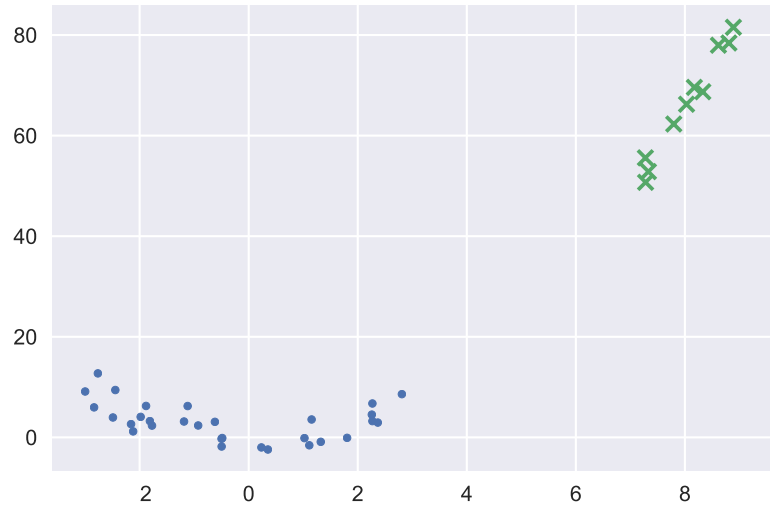
If we want to cluster based on the actual objects in the image, it is probably better to use some "smarter" architecture. We could represent each image by running it through the lower layers of a pre-trained CNN, for instance. In this case, we may try to use the output from different layers, depending on what kind of abstraction seems to be useful for the clustering.

(d) Assuming we have a test set where each image is tagged as a car or a bicycle, we may use an *external* clustering evaluation metric that compares the clusters to the annotated labels. There are some different metrics of this kind, such as the purity/inverse purity or the Rand score. (Internal clustering metrics such as the silhouette score are not relevant here.)

See `https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation`.

Question 8 of 12: Extrapolation (8 points)

The figure below shows a dataset for a regression task. Here, we would like to predict the output (the y axis) as a function of the input (the x axis).



The training set is represented by the blue dots in this figure. We train four different regression models using scikit-learn:

```
mlp1 = MLPRegressor(activation='relu', hidden_layer_sizes=16)
mlp2 = MLPRegressor(activation='tanh', hidden_layer_sizes=16)
rf = RandomForestRegressor(max_depth=5, n_estimators=10)
lin = LinearRegression()
```

The test set (green crosses in the figure) is sampled in a region that is completely distinct from the training set. Describe the potential behavior of each of these models when applied to the test set. We're writing "potential" here because there will be some variation in the models due to initialization, tweaking and so on; the idea here is that you should discuss and explain some ways that these models might possibly behave.

Solution.

Note: this is just $y = x^2$ with some noise added to x afterwards. All models could fit the data easily if we would add a new feature x^2 to the input, but the point here is to reason about the behavior of the models.

ReLU neural network: this is a piecewise linear regression model. The exact behavior will depend on initialization, tuning and so on. With some luck, we will fit the training data well

with a few line segments, and the last segment might extrapolate well into the test set region.

Tanh neural network: this is a linear combination of soft threshold functions. Most likely, the outputs are going to stay somewhere near the range of the training set. With some extreme luck, we might happen to fit one of the tanh units so that it activates near the end of the training set and stretches into the region of the test set. Again this depends on exactly how we tune and initialize, but there is at least a theoretical possibility that some model might extrapolate well into the test set region.

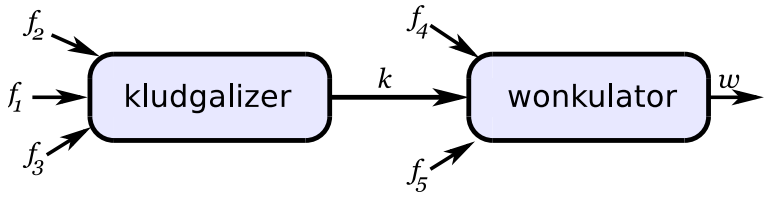
Random forest: because all thresholds and output values are defined by the training set, all the variation in the outputs will happen inside the region of the training set. When we move out of the training set region, the model's prediction will be a constant value defined by the mean of some subset of data points in the training set.

Linear regression: the model is represented by a straight line. The training is based on minimizing the squared error loss, which means that this line will do its best to stay near all the blue dots. So unless the model is very poorly fitted to the training data, the predictions are probably going to be somewhere near zero in the region where the test set points are located.

Question 9 of 12: Connected machine learning systems (6 points)

In an application at a Gothenburg industry, we want to develop two subsystems: the *kludgalizer* and the *wonkulator*. We want to use machine learning to implement both of these systems.

The kludgalizer depends on three input features f_1, f_2, and f_3 that are generated from specialized sensor hardware, and it produces a discrete output k. The wonkulator uses the features f_4 and f_5, also from hardware, in addition to a kludgalizer output k.



To develop these systems, we have a training set that is structured as a tabular file. Here, k and w are the desired kludgalizer and wonkulator outputs, respectively.

f_1	f_2	f_3	f_4	f_5	k	w
⋮	⋮	⋮	⋮	⋮	⋮	⋮

(a, 3p) If we train the wonkulator directly on this data, using the inputs f_4, f_5, and k from the training file, it is likely to perform quite poorly when we run the system "live." Explain why.

(b, 3p) Propose some way to mitigate this problem.

Solution.

Side note: this task is inspired by "pipelines," serially connected subsystems, in natural

language processing. When training NLP pipelines, the kinds of problems we've seen here are quite common. But I chose to formulate this in a more abstract way to focus more on the essence of the problem rather than a particular application.

(a) For the second subsystem, the w predictor, there is a discrepancy between the training scenario and the deployment scenario, because the k values will be perfect in the training data but will be automatically predicted, and probably imperfect, when the systems are running.

To understand the problem, consider a case where w can be computed in a deterministic manner as a function of k. In the extreme case, w might be equal to k in all instances in this training set. If this is the case, this will also be what the learned model represents. In such a case, if we make a mistake when predicting k, we are guaranteed to make another mistake when predicting w ("error compounding"). Hypothetically, our system might predict w in a more robust manner if we don't trust the k values too much, even if k gives perfect information about *w in the training set*.

(b) There are many approaches that can be tried. Here are some alternatives where we don't need to change the structure of the models.

- Dropout or noise: during training, we corrupt the k values in the training set with some probability.

- Cross-validation or "jack-knifing": we try to simulate a situation where the k values will be automatically predicted. We train a k predictor on the first half of the training set and predict the k values on the rest of the training set and vice versa. These predicted k values will now be used when training the w predictor.

In both cases, the w predictor might learn not to trust the k prediction completely, and try to find some useful information in f_4 and f_5 instead.

We could also try to change the model structure so that the w prediction does not depend on the k prediction in this "serial" way. We could predict the k and w values jointly in some way, using e.g. a probabilistic model or a neural network. For instance, we might think of a neural network where a latent representation is passed from the k prediction unit into the w prediction unit. However, redesigning the way that the subsystems are connected would of course make the system as a whole less modular and might not be what the company desires.

Question 10 of 12: Multiclass linear classification (6 points)

We want to build a classifier for multiclass problems: that is, when we have more than two classes. We use a linear classifier for this.

The model's parameters are stored in a weight matrix $\boldsymbol{W}$ with m rows and n columns, where m is the number of classes and n the number of features. Intuitively, each row is like a binary classifier that detects one of the classes. For an instance $\boldsymbol{x}$ (a feature vector), we can compute a score for the class y by taking the row $\boldsymbol{w}_y$ from $\boldsymbol{W}$ and then computing the dot product $\boldsymbol{w}_y \cdot \boldsymbol{x}$.

So to classify the instance $\boldsymbol{x}$, we search over all possible classes and select the one that gives

the highest score:

$$\hat{y} = \arg\max_y \boldsymbol{w}_y \cdot \boldsymbol{x}$$

There are many different loss functions we might use to train such a model. In our case, the loss function looks like this:

$$\text{Loss}(\boldsymbol{W}, \boldsymbol{x}, y) = \boldsymbol{w}_{\hat{y}} \cdot \boldsymbol{x} - \boldsymbol{w}_y \cdot \boldsymbol{x}$$

That is: for an instance $\boldsymbol{x}$ with the gold-standard class label y, we first compute the prediction $\hat{y}$ as defined above, and then the difference in scores between the predicted class and the gold-standard class.

(a, 1p) What are we trying to achieve when using this loss function? Explain in words.

(b, 3p) The subgradient of the loss function with respect to the weight matrix $\boldsymbol{W}$ is a matrix of the same size as $\boldsymbol{W}$. To express the subgradient, let's introduce the utility function $\phi(\boldsymbol{x}, y)$. For a feature vector $\boldsymbol{x}$ and a class y, this function returns an m-by-n matrix where the row corresponding to y is equal to $\boldsymbol{x}$, and the other rows are all zero.

The subgradient of the loss now simply becomes

$$\nabla_{\boldsymbol{W}} \text{Loss} = \phi(\boldsymbol{x}, \hat{y}) - \phi(\boldsymbol{x}, y)$$

Write the pseudocode (or Python approximation) for a stochastic gradient descent algorithm (with a minibatch size of 1) to train a classifier by minimizing the loss function on a training set. You can assume that the learning rate is a user-defined constant and that we don't use a regularizer.

(c, 2p) If we are not careful, a simple implementation of our training algorithm may be quite inefficient, in particular if the number of classes is quite large. Modify your pseudocode so that it is more efficient and avoids redundant computations. Alternatively, motivate why your previous solution is already efficient enough.

Solution.

The training algorithm we get by applying SGD to this loss is called the *multiclass perceptron*.

(a) The loss is positive when the prediction is incorrect and zero when it is correct. We want to train the model so that the score for the correct choice is higher than for all incorrect choices.

(b) We can just approach this in a "plug and play" fashion and we've seen plenty of examples of this kind in the course. So we just take the subgradient described above and "plug" it into the regular SGD algorithm. Remember the minus sign in SGD when we are minimizing! The result will be something like the following. (We don't include the pseudocode for ϕ here.)

Inputs: a list of example feature vectors $\boldsymbol{X}$
a list of outputs Y
learning rate η
$\boldsymbol{W}$ = zero m-by-n matrix
repeat

select a training pair (input $\boldsymbol{x}$, output y)
$\hat{y} = \arg\max_y \boldsymbol{w}_y \cdot \boldsymbol{x}$
$\boldsymbol{W} = \boldsymbol{W} + \eta\phi(\boldsymbol{x}, y) - \eta\phi(\boldsymbol{x}, \hat{y})$

In this implementaiton, we don't say exactly how we select the examples or how we decide to terminate here, but typically we will go through the training set a fixed number of times, or track how the loss function evolves after each epoch.

(c) The function ϕ that we defined above is useful to keep the mathematical notation compact, but from a computational perspective it is not smart to create a full matrix with mostly zeros where the only effect on $\boldsymbol{W}$ is at the y and $\hat{y}$ rows. The effect of adding $\phi(\boldsymbol{x}, y) - \phi(\boldsymbol{x}, \hat{y})$ is that we add $\boldsymbol{x}$ to the row y and subtract it from the row $\hat{y}$. So let's implement this directly. We may also note that when the prediction is correct, we don't update the model at all.

Inputs: a list of example feature vectors $\boldsymbol{X}$
a list of outputs Y
learning rate η
$\boldsymbol{W}$ = zero m-by-n matrix
repeat
  select a training pair (input $\boldsymbol{x}$, output y)
  $\hat{y} = \arg\max_y \boldsymbol{w}_y \cdot \boldsymbol{x}$
  if $\hat{y} \neq y$
    add $\eta \cdot \boldsymbol{x}$ to row y of $\boldsymbol{W}$
    subtract $\eta \cdot \boldsymbol{x}$ from row $\hat{y}$ of $\boldsymbol{W}$

The computational bottleneck now is the arg max, and it seems hard to improve it because we always have to compute the scores for all possible classes. But at least we have improved the update step and with this implementation, later epochs will probably be faster because we make fewer mistakes.

If you implemented it in this way in (b) already, then of course it's enough to say in (c) that you can't think of anything more efficient than this.

Generally, there are specialized solutions if the number of classes is huge, for instance introducing some sort of hierarchy among the classes. This is not what we're thinking of here.

Question 11 of 12: Approximation (5 points)

We have some sort of black-box system and we want to use a machine learning approach to train a new system that replicates the behavior of the original system as faithfully as possible. For instance, the black-box system might be a neural network that is too large to fit onto a mobile device, or some sort of proprietary system that we want to mimic.

How would you solve a task like this? Describe all relevant steps and what kind of resources you would use in the ideal case. How do you know you have succeeded?

To make the discussion more specific, we can assume that the black-box system is a classifier that gives a probabilistic output (similar to e.g. a neural network with a softmax output). Apart from this, we make no assumptions about the inner workings of this system.

Solution.

This task is more or less equivalent to what is known in the ML field as *knowledge distillation*, which was introduced by Hinton in 2015.[3] For instance, small "student" neural networks sometimes perform surprisingly well when trained to imitate much larger "teacher" NNs, and this seems paradoxical because why did we have to use that large model in the first place then? As of 2021, this is poorly understood in the research field. But just to be clear, you are not assumed to know anything about Hinton's paper or knowledge distillation here and common sense should be enough. We are actually taking a more general perspective here, since the original system can be anything and is not necessarily a ML-based model.

The task is a bit vague and we need to clarify some issues: what do we mean by "replicate faithfully"? How do we measure this?

A simple solution might be that we use the original model to label some dataset, and then train the new model in a straightforward manner. What data should we use? In the ideal case, we have a fairly large sample from the domain where we want to apply the system. We could really use any type of classification model in this approach. To see how well we are approximating, we'd set aside a test set and apply a regular classification evaluation metric such as the accuracy, precision/recall etc.

However, the first solution does not take the system's behavior into account perfectly, and to fully mimic the original system we also need to take the predicted probability distributions into account. For instance, if the original system is quite uncertain in its prediction for some instance, we would like the new system to represent the same uncertainty. So we probably need to define some loss function that takes the probability distributions into account: we would like the predicted distributions to be as similar as possible. Such a loss could be, for instance, the cross-entropy between the two distributions, or more generally any approach to measure the distance between probability distributions. You might even just compute the sum of squared errors over the probability vectors here, although I think that will work poorly... In this approach, we are a bit more restricted in what kind of model to use, since we need to have something that works with our loss function. A neural network or a linear model would be the natural choices here, but any model is OK if you explain how you deal with the loss. We'll probably also use this loss function to evaluate how well we're approximating, but we could of course also look at accuracies etc.

In some cases, we might take an analytical approach and prove that the outputs from some small model are "close enough" to a larger model, but this seems difficult to do in the general case and you'd probably have to make certain assumptions about the structure of the "teacher" and "student" models. Measuring some quantity on a dataset seems much easier.

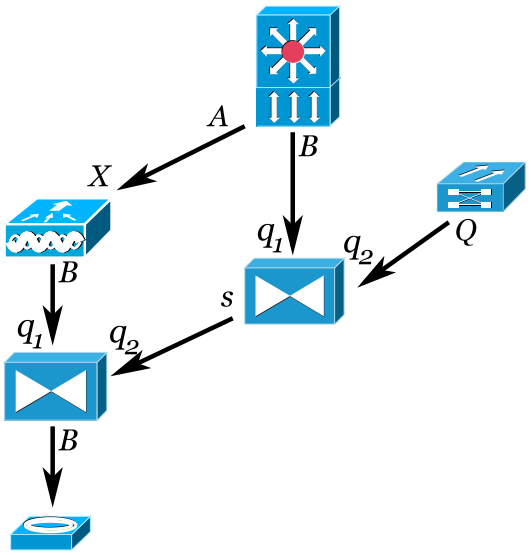
Note about grading: for a full score, you will need to consider the probability distribution in some way, but you will get some points if you propose the simple solution described above.

Question 12 of 12: Classifying networks (5 points)

A large Swedish telecom company manufactures different types of components that can be assembled as a network structure. Each component belongs to a certain type, and has a number of inputs and outputs.

[3] `https://arxiv.org/abs/1503.02531`

The following figure shows an example of such a network.



In some scenarios, we would like to apply a classification or regression model to the network as a whole. For instance, we might want to develop a classifier that determines whether the network is connected correctly or not.

(a, 4p) How would you develop such a classifier?

(b, 1p) Can you think of an application that does *not* involve a network of components, but resembles this type of problem on a structural level?

Solution.

This task is inspired by the Master's thesis *Pattern Recognition of RBS Configuration Topologies in Radio Access Networks* by Ellinor Rånge and Joacim Linder.[4]

Formally, we have a classification problem where the input is a graph. We have discussed graph classification problems in some previous exams, see for instance the last question in this exam:

`http://www.cse.chalmers.se/~richajo/dit866/files/DIT866_DAT340_2019_mar.pdf`

and the solution

`http://www.cse.chalmers.se/~richajo/dit866/files/mar_2019_solution.txt`

(a) As we have discussed for other questions in this exam, we need to have training data in order to use a supervised ML setup. The training instances would consist of examples of networks, and for each network we can see the desired output: discrete or continuous, depending on whether we are trying to solve a a classification or regression task. As in question 1, we would need to use domain experts to annotate this dataset if the outputs we want to predict are not already available in the data.

[4] `https://odr.chalmers.se/handle/20.500.12380/251547`

Then, we want to build the predictor. How can we provide a network as an input to a ML model? On a high level, there are two main approaches:

- Defining a feature function that extracts ”useful” properties from the graph with respect to our prediction task, and then apply any standard ML algorithm. Designing this feature function will involve any domain expertise we might have, as well as trial and error.

Intuitively, we might build a feature representation in a ”bag-of-words” manner by representing what types of components are present in a graph, as well as pairs, triples, etc of components. We might also consider graph-structural features such as the presence of cycles, lengths of paths, etc.

See e.g. `http://www.cs.rpi.edu/~zaki/PaperDir/SADMJ12.pdf`

- Using a specialized ML architecture for graph-based ML, such as graph CNNs or graph kernels. See e.g. `http://proceedings.mlr.press/v48/niepert16.pdf` and `https://arxiv.org/pdf/1903.11835.pdf`.

The second of these approaches of course hasn't been covered in the course, so probably students' solutions will be related to the first approach.

(b) Here, the idea is that you should mention some prediction task where the input would be some sort of graph or network. Again, you might look back at the previous exam. Most work in graph-based ML seems to be related to the classification of molecules, e.g. distinguishing toxic/non-toxic chemical compounds. There are applications in social media analysis as well.