Solution to the exam
DIT865/DAT340: Applied Machine Learning, March 15, 2018

Question 1 of 12: Predicting house prices (8 points)

A real estate firm would like to build a system that predicts the sale prices of a house. They create a spreadsheet containing information about 1,460 house sales in the Gothenburg area. In addition to the price, there are 79 features describing the house, such as *number of bedrooms*, *total indoor area*, *lot area*, *has a garage*, *location*, etc.

(a, 6p) Explain how you would implement a machine learning model that would solve this prediction task. You don't need to show Python code, but please give a description of the system and explain all steps you would carry out when developing it.

(b, 2p) Explain why the model you built is probably useless in the long run.

Solution.
(a) Here, the idea is that you describe the whole development process required when applying a machine learning approach.

We first convert the spreadsheet into a matrix. We can use a `DictVectorizer` for this purpose, or alternatively use the `pandas` library.

This is clearly a regression problem, so we need to pick a useful regression model. Maybe try out some linear model (`Ridge` or `Lasso`) and compare it to a more complex model? Tree ensembles are often useful for this type of data, so it's not unlikely that a `GradientBoostingRegressor` or `RandomForestRegressor` would give the best performance in this case.

Methodologically, we typically split off a test set. We select the best model (learning method, hyperparameter tuning) either by also using a development set, or via cross-validation. In scikit-learn, we have `GridSeachCV` and `RandomSearchCV` to help us tune parameters, for instance. Finally, we evaluate on the test set. The evaluation metric will be a regression metric such as mean squared error (scikit-learn: `mean_squared_error`) or R^2 (`r2_score`). It can be useful to compare our result to a baseline.

To get a full score, it is obviously necessary to say that you need to apply a regressor, but it's also necessary to describe something more than just saying "I'll use a linear regression model." It is important that you discuss how to evaluate, and something about how to select a model.

(b) "In the long run we are all dead", as Keynes remarked. More seriously, if we keep using this regression model in its current state, without retraining it on new data, it seems likely that it will grow worse over the years, as conditions on the housing market change. (Prices rise, or suddenly crash, fashions change, etc.)

Question 2 of 12: Predicting customer churn (8 points)

A mobile phone service provider would like to build a model that predicts whether a subscriber is likely to cancel the subscription in the next couple of months. (The jargon term for this is customer *churn*.) The purpose of this classifier is that the company wants to send these subscribers some special offers in order to dissuade them from dropping their subscriptions.

The company makes a training set by collecting some historical data, and then they implement a classifier using scikit-learn. This is how they declare their classifier:

```
pipeline = make_pipeline(
    DictVectorizer(),
    StandardScaler(),
    SelectKBest(k=100),
    LogisticRegression()
)
```

Can you describe the purpose of each of the four steps in the pipeline?

Solution.

- `DictVectorizer`: converts the features (which are assumed to be stored as attribute–value dictionaries) into a matrix.
- `StandardScaler`: rescales the features, in order to take care of features that differ in magnitude. Such differences in magnitude can cause trouble for some types of classifiers (including `LogisticRegression`, which we use here). This particular scaler will subtract the mean and divide by the standard deviation for each feature.
- `SelectKBest`: selects a subset (of size 100 in this case) of the features. The subset of features that seem to be most strongly correlated with the output are selected. The purpose is to get rid of useless features, which makes it easier for the classifier to learn.
- `LogisticRegression`: the actual classifier, which predicts whether the user will or will not drop the subscription.

Question 3 of 12: Evaluation (8 points)

A telecom company has developed a classifier for detecting whether a cell-phone network base station is faulty or not.

(a, 3p) We evaluate the classifier on a test set. Here is the confusion matrix. (In the table, F means *faulty* and NF *not faulty*.)

		Predicted	
		F	NF
Truth	F	10	5
	NF	20	965

Compute the accuracy of the classifier.

(b, 2p) What is the accuracy of a majority-class baseline? (The class *not faulty* is the most common in the training set.)

(c, 3p) Would you say that the classifier is more useful than the majority-class baseline? Explain why or why not.

Solution.
(a)

$$\text{accuracy} = \frac{10 + 965}{10 + 5 + 20 + 965} = 0.975$$

(b) In this case, all predictions are NF. (We add the column with the F predictions to the NF column.)

$$\text{accuracy} = \frac{985}{985 + 15} = 0.985$$

(c) Clearly, the classifier is more useful than the majority baseline even though the latter has a higher accuracy. For this type of task, precision/recall is more meaningful than the accuracy. After all, our classifier able to find two thirds of the faulty base stations (recall = 10/15), even though it overgenerates a bit (precision = 10/30).

Question 4 of 12: Overfitting (6 points)

(a, 3p) What is *overfitting* and why is it a problem?

(b, 3p) Give an example of a method to reduce the risk of overfitting.

Solution.
(a) According to the Oxford dictionary, overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably". Informally, our model is too specific for our training set and fails to generalize well to new instances.

(b) In the machine learning field there are countless methods to mitigate overfitting. A few examples:

- using a regularizer, such as the L_2 and L_1 regularizers we've seen for linear models;
- in decision trees, we may set a limit on the tree depth, or "prune" tree nodes that seem useless when considering a held-out set;
- using ensembles where we try to ensure the diversity of the sub-classifiers (e.g. bagging);
- dropout in neural network, which is also similar to "feature bagging";
- early stopping: terminate training when it seems that the model is starting to overfit.

Question 5 of 12: Decision tree classifiers (4 points)

We have a training set that includes three features, and we'd like to predict a binary output variable. Here is the whole training set.

x_1	x_2	x_3	y
A	X	P	True
A	X	Q	True
A	Y	P	False
A	Y	Q	False
B	X	P	True
B	X	Q	False
B	Y	P	False
B	Y	Q	False

If we train a decision tree classifier using this training set, which feature would be considered at the first "branch" of the decision tree (the top node)? Please explain why.

Solution.
The feature x_2 is selected, because it is the feature that has the strongest association with the output variable. Formally, we can compute a "purity" measure that shows how the feature splits the training set into subgroups that are more homogeneous than the subgroups induced by the other two features. Here's how two such purity measures can be computed:

	x_1	x_2	x_3
	A: 2/2	X: 3/1	P: 2/2
	B: 1/3	Y: 0/4	Q: 1/3
majority sum	5	7	5
information gain	0.05	0.55	0.05

You will get a full score here if you say (1) that x_2 is selected, and (2) that this is because x_2 best describes the output. It's of course clearest if you explain (2) by means of a measure, but if you can give another clear explanation, that is also acceptable.

Question 6 of 12: Pest control (6 points)

You are contacted by a food processing company that wants you to develop a classifier that detects whether a rat is present in an image. You collect a large dataset of images by crawling the web, and have annotators determine which images contain rats. This set of images can then be used as the training set for your classifier.

(a, 2p) Suggest a machine learning method to use for this classification task.

(b, 4p) After you have delivered your solution to the company, they get back to you and complain that when they evaluate on a new test set, they get precision and recall values that are much lower than what you reported to them. Explain what might have gone wrong.

Solution.
(a) Clearly a typical use case for a CNN, either by training a new CNN, or by reusing a pre-trained model. (If you have a good argument for some other classifier, that is also acceptable.)

(b) Your training set was trained by crawling images off the web. This probably means that your images tend to be quite different from the typical images the classifier will be exposed to in a real-world setting (which this case seems to be some sort of food processing factory). We have two different *domains*. [Full score if you can explain that it's about the training data having a different distribution from real life.]

Question 7 of 12: Neural network regression (5 points)

We'd like to train a regression model that is able to predict a numerical output y, given a numerical input x. The inputs are one-dimensional (that is, each x is just one number). Using scikit-learn, we train a neural network:

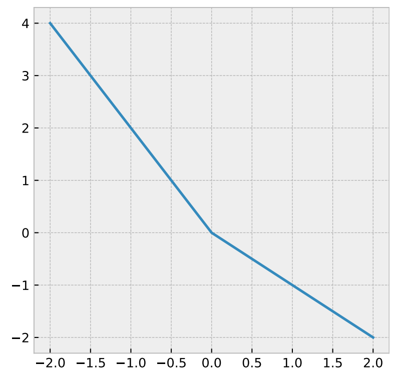
```
MLPRegressor(activation='relu', hidden_layer_sizes=2)
```

After training, the weight matrices for the two layers are

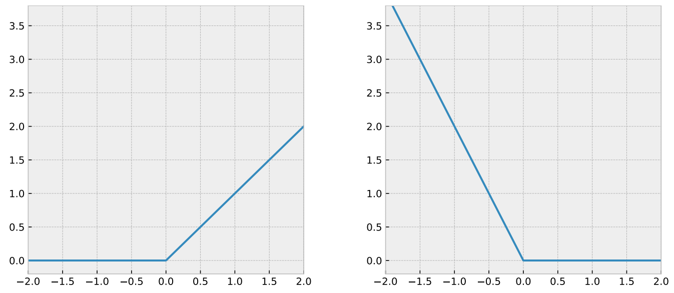
$$\begin{bmatrix} 1 \\ -2 \end{bmatrix} \qquad \begin{bmatrix} -1 & 1 \end{bmatrix}$$

(To clarify: the left matrix contains the weights for the two hidden units, and the right matrix the weigths for the output unit.) Can you plot the output of the model as x ranges from -2 to 2?

Solution.
The use of the rectified linear units in the hidden layer leads to a piecewise linear regression model, with a "kink" at 0.



Detailed explanation: for the two hidden units, after applying the ReLU, we get the following outputs:



The output is -1 times the first hidden unit, plus 1 times the second hidden unit. [During the exam, I got some questions about the output activation function. I was reluctant to give explicit answers because I think you should know that the output in a neural regression model is typically a linear function. After all, if the output unit had a ReLU, it would be impossible for this type of regression model to produce a negative output.]

To get a full score, the first plot is enough.

Question 8 of 12: A variant of the perceptron (5 points)

The following pseudocode shows a variant of the perceptron learning algorithm. This algorithm often works significantly better than the standard perceptron algorithm. Can you think of a reason why this is the case?

```
$w = (0, \ldots, 0)$
$w_s = (0, \ldots, 0)$
**repeat** $N$ times
```

```
  for $(\boldsymbol{x}_i, y_i)$ in the training set
    if $y_i \cdot \boldsymbol{w} \cdot \boldsymbol{x}_i \leq 0$
      $\boldsymbol{w} = \boldsymbol{w} + y_i \cdot \boldsymbol{x}_i$
    $\boldsymbol{w}_s = \boldsymbol{w}_s + \boldsymbol{w}$
return $\boldsymbol{w}_s / (N \cdot T)$
```

In the pseudocode, N is the number of epochs and T is the size of the training set.

Solution.
[NB: this question had an indentation typo in the printed version.]

This algorithm outputs the average of all the temporary models that we've observed during training. (The algorithm was originally described in the paper *Large Margin Classification Using the Perceptron Algorithm* by Freund and Schapire, *Machine Learning* journal, 1999.) This average is actually an ensemble, so it has advantages in terms of robustness over the standard perceptron algorithm. We can also note that the standard algorithm is adapted to the last few misclassified examples, and this problem is mitigated by this averaging process.

Question 9 of 12: Logistic regression (8 points)

Logistic regression (LR) is one of the most popular machine learning models. Training a LR model is done by finding the weight vector $\boldsymbol{w}$ that minimizes the function f in the following equation:

$$f(\boldsymbol{w}, \boldsymbol{X}, Y) = \sum_{i=1}^{n} L(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \frac{\lambda}{2} \cdot \|\boldsymbol{w}\|^2$$

As usual, $\boldsymbol{X}$ is a list of feature vectors $\boldsymbol{x}_i$ of all the instances in the training set and Y the corresponding outputs y_i (each output coded as +1 or -1). λ is a user-defined parameter.

The loss function L is defined

$$L(\boldsymbol{w}, \boldsymbol{x}_i, y_i) = -\log \frac{1}{1 + \exp(-y_i \cdot (\boldsymbol{w} \cdot \boldsymbol{x}_i))}.$$

(a, 4p) Explain why the loss function looks like it does.

(b, 4p) What happens to $\boldsymbol{w}$ if we replace $\|\boldsymbol{w}\|^2$ by

$$\|\boldsymbol{w}\|_1 = |w_1| + \ldots + |w_n|$$

in the objective function f above?

Solution.
(a) In the LR model, the probability of an instance $\boldsymbol{x}_i$ belonging to the positive class is defined by the sigmoid function

$$P(\text{positive}|\boldsymbol{x}_i) = \frac{1}{1 + \exp(-\boldsymbol{w} \cdot \boldsymbol{x}_i)}.$$

If the outputs are coded as +1 or -1, the probability of the desired output y_i is

$$P(y_i|\boldsymbol{x}_i) = \frac{1}{1 + \exp(-y_i \cdot (\boldsymbol{w} \cdot \boldsymbol{x}_i))}$$

Training LR means that we maximize the likelihood of the desired outputs in the training set. The purpose of the $-\log$ part is mainly to put this optimization problem in the same framework as the other learning problems (least squares, SVC, etc.): minimizing the sum of something, rather than maximizing the product of something.

To get 4 points here, you need to mention that this loss function is derived from the LR model's probability output.

(b) This is called the L_1 regularizer and it has the same effect as the *Lasso* model for linear regression. This type of regularizer will tend to produce a *sparse* solution where some feature weights are set exactly to zero, which means that these features can be removed entirely from the model.

Question 10 of 12: Word embeddings (8 points)

(a, 3p) What does it mean to *pre-train* word embeddings and why is this useful?

(b, 3p) It has been noted that word embeddings can encode some gender and ethnic biases.[1] Can you explain why this is the case?

(c, 2p) Can you think of an application where it may be useful to train "word embeddings," but the data is not text? (That is, the embedded objects are not words.)

Solution.
(a) Word embeddings are short vectors that encode the "meaning" of a word. Ideally, this "meaning" would be defined by the task we're trying to solve, but in many cases it can be useful to pre-train the embeddings.

Pre-training means that we apply a training algorithm (such as `word2vec` or Glove) that learns word vectors without annotated data, just a lot of text. This pre-training is based on the word's statistical properties: for instance, *coffee* and *tea* are similar because they have similar statistical profiles, co-occurring often with *drink*, etc.

Pre-training is useful because it allows us to use more data for training. (In principle, using pre-trained vectors is a semi-supervised learning setup.) Without pre-training, we are helpless when we encounter a word we haven't seen in our training set.

(b) As mentioned in (a), training word embeddings is based on co-occurrence patterns. For this reason, the embedding model will pick up the biases that are present in our everyday language. To exemplify: if nurses are stereotypically female, nurses will tend ot be mentioned in female-related contexts, so the word *nurse* will have a strong association with other female-related words.

(c) An embedding layer is often included when we need to feed "symbols" as input to neural networks, and the set of symbols is so large that it's impractical to apply one-hot encoding. Representing words is just the most classical example: there are a lot of possible words.

What other examples are there? Collaborative filtering is another example where we compute user and item embeddings. This is typically phrased as a matrix factorization problem, but it's an idea that is in effect very similar to word embeddings. Some other examples include graph embeddings where nodes are associated with vectors, atom embeddings in chemistry.

[1] See for instance Bolukbasi et al. (2016), *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings*, in Proceedings of Neural Information Processing Systems (NIPS).

Question 11 of 12: Sequence models (6 points)

(a, 3p) What does the notion of *state* mean for recurrent neural networks?

(b, 3p) What are the *encoder* and *decoder* in a neural machine translation system?

Solution.
(a) A RNN processes sequential data in a step-by-step fashion from the start to the end, and passes the state between time steps. The state is simply a vector that is part of the input to the RNN; it is intended to summarize what has happened previously in the sequence.

(b) In neural machine translation and related problems, the encoder is a RNN that "consumes" the input sequence and produces an encoded representation that summarizes this input. Based on this encoded representation, the decoder (another RNN) generates the output, one item at a time. (Often, the decoder also makes use of an *attention model* that can focus on specific parts of the input, instead of just using a single encoded representation.)

Question 12 of 12: Recommending fashion items (8 points)

An e-commerce platform that sells clothes wants to develop an image-based recommender system. Here are some examples of images of the company's items:

The image is the only piece of information that is available about each item. For customers, you have access to the purchase history but no other information.

On a high level, can you describe how you would implement such a recommender system?

Solution.
From the instructions, we can understand that we'd like to use a *content-based* recommender: the recommendations are based on features of the items, in this case the images. This is probably the right approach for recommending clothes: new items occur all the time, leading to the "cold start" problem for these items, ruling out a pure collaborative filtering approach.

As discussed, in a content-based approach, we need to describe the items in terms of features. A simple approach is to define features extracted by a pre-trained CNN. (Or train a CNN directly for this task.)

How exactly the recommendation is done will depend on how the system is intended to be used and how it is integrated in the user interface. If the intended use is "recommend to the customer some items similar to previously purchased items", then we can imagine that we compute a mean vector (centroid) of the features of the previously purchased items, and then recommend the non-purchased items most similar to this mean. In practice, it might be more useful to recommend items similar to what the user is currently looking at. Regardless of exactly how the recommendation is done, the image-based features allows us to find items that are similar to each other.

You have some freedom here: you just need to persuade me that you have a credible solution. The main thing to be clear about is how to extract features for the items.

This question is inspired by a paper by some researchers at Zalando, *Fashion DNA: Merging Content and Sales Data for Recommendation and Article Mapping* by Bracher et al (2016). In the approach described in the paper, images are used but also some manually defined features.