TDT4173 – assignment 1

Iver Jordal

1 Theory

Basic concepts

1. Machine learning problems

Definition: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."*

1. Recognizing cats on images
    a. Experience E: Database of human-labeled images of cats
    b. Tasks T: Correctly indicate which images have cats in them, and where the cats are
    c. Performance measure P: Percentage of correctly labeled images
2. Learning to write C
    a. Experience E: The source code of a linux kernel
    b. Task T: Write C code with correct syntax
    c. Performance measure P: Percentage of produced code that is syntactically correct

2. Inductive bias

What is inductive bias?

Inductive bias is the set of assumptions needed to be able to process input data that the system has not encountered before, and predict the output (correctly). If the system cannot do that, then it also has not learned anything other than the specific examples that it has been trained on. In that case, it is basically just a key-value store. That is why inductive bias is so important in machine learning.

The **candidate elimination algorithm** for learning in version spaces: This algorithm uses A) Maximally general hypotheses and B) Maximally specific hypotheses. During training, when a case is inconsistent with a hypothesis, then that hypotheses is removed and new hypotheses that take the new case in account may be added. The algorithm may converge to a single hypothesis that is correct for all training cases. What can we say about this algorithm's inductive bias? The learning algorithm has an idea about when and how hypotheses should be created and which of the hypotheses should be removed. It uses hypotheses to predict the output of unseen cases. Those hypotheses include assumptions used to predict outputs of unseen cases. The **bias is mainly in the representation**, which basically consists of conjuncts of expressions.

Learning of decision trees with **ID3**: Inductive bias in ID3 is that 1) shorter trees are preferred over larger trees and 2) trees that have high information gain near the root are preferred over those that do not. In other words, there is no representational bias, as in the CE algorithm, but **the bias is on the search**, which is greedy.

3. Overfitting

What is overfitting? Overfitting occurs when the learned model becomes really good at training cases, but the predictive performance is poor. For example, if the training data has a lot of noise, and the

learner eventually specializes the model to be really good at describing that noise, the learned model might be bad at predicting the output for unseen input.

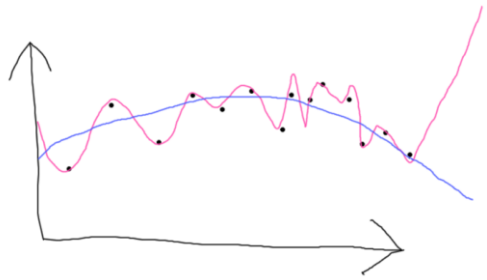
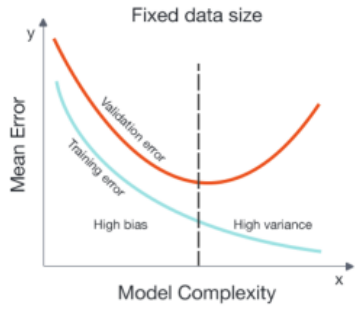


Figure 1: The pink curve represents an overfit model of the black data points. Its error for the training data is low, but generally, its predictive performance is poor. Observe that the pink curve goes up in the rightmost part, while that is probably not how the data series continues. The blue curve is simpler, and although the error is slightly larger for the given black data points, it will probably have much better predictive performance.

Bias vs. variance: In machine learning, there's a tradeoff between bias and variance. How complex should the model be? How much should it adapt to small fluctuations in the training data? If the model adapts to small fluctuations (=> more variance), it becomes more complex and thus prone to overfitting. However, in case of high bias, the model might become underfit, i.e. it doesn't learn all the relevant relations between features and target outputs well.

Generally, it is a good idea to use cross validation to limit overfitting. Stop training when the mean validation error starts going up.



Concept Learning

Attributes	**Values**
Body	Hair, Scales, Feathers
Birth	Live, Egg
Eats-Meat	True, False
Flies	True, False
Teeth	Pointed, Flat, None

1. Version space learning

$$S_0 = \{< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >\}$$

$$G_0 = \{<?,?,?,?,?>\}$$

Training example 1: Hair, Live, False, False, Flat => True (positive example)

$$S_1 = \{< Hair, Live, False, False, Flat >\}$$

$$G_1 = G_0$$

Training example 2: Feathers, Egg, True, True, None => False (negative example)

$$S_2 = S_1$$

$$G_2 = \{< Hair,?,?,?,?>,<?,Live,?,?,?>,<?,?,False,?,?>,<?,?,?,False,?>,<?,?,?,?,Flat>\}$$

Training example 3: Hair, Live, False, False, Pointed => True (positive example)

$$S_3 = \{< Hair, Live, False, False, ?\}$$

$$G_3 = \{< Hair,?,?,?,?>,<?,Live,?,?,?>,<?,?,False,?,?>,<?,?,?,False,?>\}$$

Now, let's use the system to classify the test examples:

Example	Body	Birth	Eats-Meat	Can-Fly	Teeth	Mammal
1	Hair	Live	False	False	None	**True**
2	Feathers	Egg	False	True	Pointed	**Don't know**
3	Scales	Egg	True	False	Flat	**Don't know**

2. Classifying three new examples

Since we only have partially learned concepts, we have to check all hypotheses. If all hypotheses classify an instance as either negative or positive, we can, with confidence, classify it accordingly. The reason why can be explained by the fact that if we keep training, the instance will still be classified the same. If the hypotheses do not agree on a single classification, we need more more training examples before we can classify the instance with confidence. The first new example was classified with confidence, but the two others were not, because the hypotheses did not all classify them similarly.

3. Asking for more training examples

If the system would be able to ask for training examples, it should ask for training examples that some of the hypotheses classify positive, but negative by others. This is the case for the last two new test examples. So one example combination of attribute values could be: Feathers, Egg, False, True, Pointed.

2 Programming

Linear regression

a)

Initial parameters W and b were chosen randomly. This is inspired by the way perceptrons initialize their weights. I don't know if it's a good idea for regression, though. I could probably have chosen something myself, such as zero. Anyway, I used a pseudo random number generator with the same seed for each run, so that I could get consistent results. The seed that I chose gave me the following initial parameters:

$$W = [0.3474337369372327, 0.26377461897661403]$$

$$b = -0.365635755888$$

When it comes to the learning rate, I set that to $\alpha = 0.1$

Finding a good learning rate is usually the result of experimentation. I tried 0.1 and 0.01 and both seemed to work well. At least those values are not too large. The error doesn't bounce around or diverge.

b)

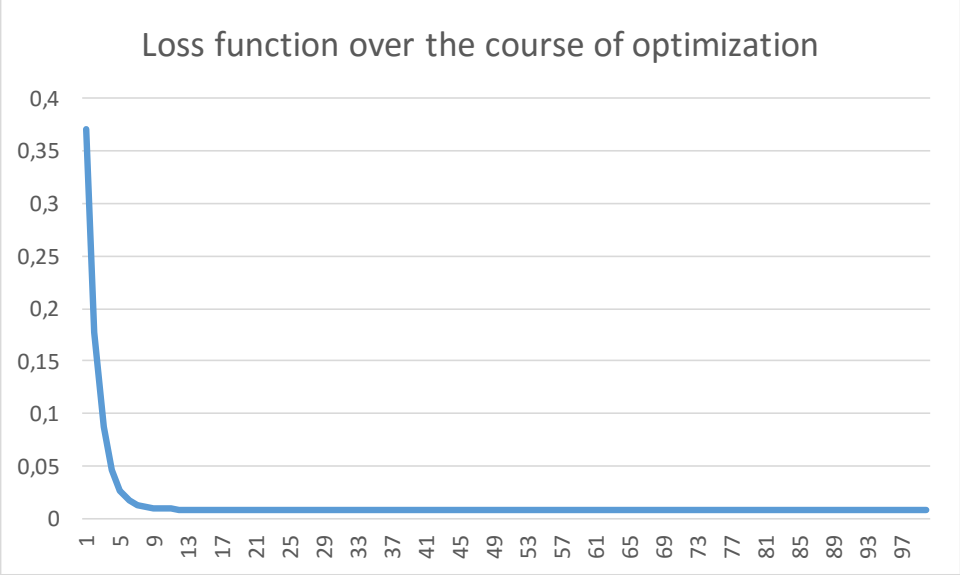
```
iteration 5
error 0.0260137583933
weights [0.5175476446878424, 0.44423970205321933]
bias -0.0434621272073

iteration 10
error 0.00930076885845
weights [0.5387148032624871, 0.4738321103624364]
bias 0.00485243408428
```

c)

After 100 iterations, I ended up with the following result:

```
iteration 100
error 0.00824354704877
weights [0.4802298601988901, 0.5208430670297846]
bias 0.0240532931613
```

The values are not very different from iteration 10, so they have pretty much converged. Indeed, a loss function plot also illustrates that fact:



Finally, the function that I ended up with is:

$$f_{W,b}(x_i) = 0.0240532931613 + 0.4802298601988901 * \mathrm{x}_{\mathrm{i},1} + 0.5208430670297846 * \mathrm{x}_{\mathrm{i},2}$$