EEE4114F: Introduction to Machine Learning

Supervised Learning

Jarryd Son
Dept. Electrical Engineering
University of Cape Town

May 2020

Contents

Supervised Learning Continued | 1

Previously we look at an overview of the machine learning landscape before diving deeper into supervised learning. We looked at how linear regression solves a regression task using a supervised learning approach. It is great for it's simplicity and interpretability (easy to see which features are important based on their learned coefficient), although the assumption that the data fits a linear model is rather limiting. These types of algorithms that assume the solution has a specific form belong to a class of methods called *parametric* methods [1].

1: Chapter 2 of [1] as well as [2] provide good explanations of parametric and non-parametric models

Figure 1.1 is to remind you of the different tasks and scenarios. We will continue our study of supervised learning with algorithms that solve classification tasks.

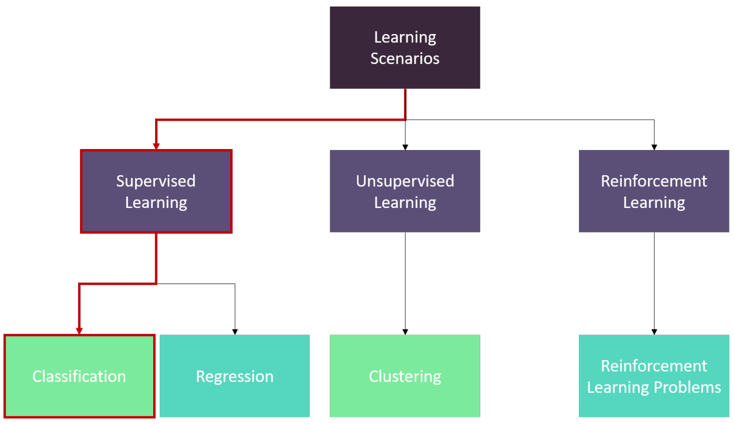


Figure 1.1: Review of the learning tasks and scenarios. We are going to look at a simple algorithm that solves classification tasks next

The next algorithm we will look at is the k-Nearest Neighbour algorithm.

1.1 k-Nearest Neighbour Algorithm

The k-Nearest Neighbour (kNN) [2] algorithm is quite a different approach to learning. Rather than looking through all the training data to produce a model that will then predict an output, the kNN algorithm stores the training data and uses it as a database to compare with incoming unseen data. This type of leaning is rather aptly named *lazy* learning, as we do not actually learn anything until we are tested. The alternative to this is *eager* learning where the learner digests the training data to learn a model which is then used to predict unseen test data.

2: not be confused with the abbreviation for neural networks which we will look at later

Think of this as a *lazy* approach to learning for an open-book test. You would have access to content to learn from (training data), however, seeing as it is open-book you decide to answer the test be comparing the test questions with the training content you have access to. Compare this to learning the content before being tested, as was the case for linear regression.

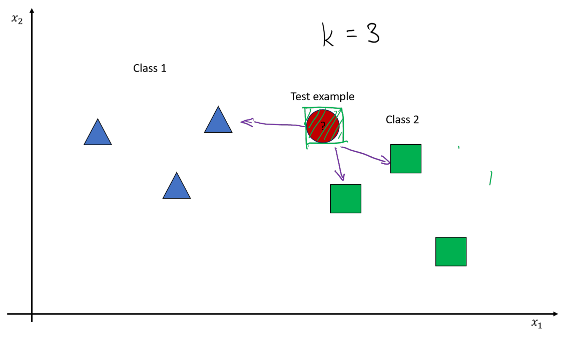


Figure 1.2: Illustration of the basic idea behind kNNs. In this example the number of nearest neighbours to consider is $k = 3$

It is a very simple approach, but can be very useful in certain cases. It is typically used for classification [3] but can easily modified to solve regression tasks. The general idea for a kNN is to determine which training examples an unseen test example is closest (similar) to. Once this has been determined, a voting process is used classify the new test example. For example see Figure 1.2

3: We will focus on its use for classification.

To formalize the algorithm [4] we have some labelled training dataset:

4: See Chapter 2.2.3 of [1] and [3] for more details

$$\mathcal{D} = \{(\mathbf{x^{[i]}}, \mathbf{y^{[i]}}), \ldots, (\mathbf{x^{[n]}}, \mathbf{y^{[n]}})\}$$

Where n is the total number of training examples. Given some new test example, $\mathbf{x^{[t]}}$, and some distance metric, $d(\mathbf{x^{[a]}}, \mathbf{x^{[b]}})$, that measure the distance between two examples [5] We want to sample the k closest training examples (*nearest neighbours*) [6] to the test set to produce a new dataset:

5: remember we can represent examples by a vector of the various features we would like to examine

6: Hence the name k-Nearest Neighbours

$$\mathcal{D}_k = \{(\mathbf{x}^{[i]}, \mathbf{y}^{[i]}), \ldots, (\mathbf{x}^{[k]}, \mathbf{y}^{[k]})\}$$

Where k is a hyperparameter that is decided upon or tuned [7] to produce optimal results. Once $\mathcal{D}_k$ has been obtained one can perform classification or regression.

7: Hyperparameter tuning is not the same as learning parameters

Classification

The predicted class will be the outcome of a plurality vote [8] A simple way to describe the hypothesis function is

8: A majority vote traditionally assumes >50% for a single class. This is fine for a two class (binary) scenario, but for multi-class scenarios you want the highest percentage for a given class - not necessarily > 50%

$$h(\mathbf{x}^{[t]}) = mode(\{\mathbf{y}^{[i]}, \ldots, \mathbf{y}^{[k]}\})$$

Where the mode is the highest occurring label (class) from a given dataset. Or if you want,

$$h(\mathbf{x}^{[t]}) = \underset{j \in \{1,\ldots,c\}}{\arg\max} \sum_{i=1}^{k} \delta(j, \mathbf{y}^{[i]})$$

Where c is the number of classes and $\delta(a, b)$ is the Kronecker Delta:

$$\delta(a, b) = \begin{cases} 1, & \text{if a = b} \\ 0, & \text{if a} \neq \text{b} \end{cases}$$

The arg max function [9] is used because we want to output the class (label) itself and not the number of target classes that belonged to that class.

9: You will come across this function quite frequently when programming. It returns the index of the element with the highest value. The arg min function is similarly used to return the index of the element with the lowest value.

Regression

If you wanted to perform regression you could still determine the k nearest neighbours, except instead of calculating the mode of the labels you could calculate a real value such as the mean. This would be given by:

$$h(\mathbf{x}^{[t]}) = \frac{1}{k}\sum_{i=1}^{k}\mathbf{y}^{[i]}$$

Where for regression the labels are real-valued. You could use other functions such as the median etc. Whichever you think would yield the best results.

Algorithm 1: Create $\mathcal{D}_k$ for a single example

```
$n \leftarrow$ # of training examples
$k \leftarrow$ # of nearest neighbours to consider
$dist\_list \leftarrow []$
for $i = 1$ to $n$ do
    $d \leftarrow$ distance between $\mathbf{x}^{[t]}$ and $\mathbf{x}^{[i]}$
    append $d$ to $dist\_list$
end for
sorted_indices $\leftarrow$ argsort $dist\_list$ in ascending order
choose the k nearest training examples using sorted_indices
$\mathcal{D}_k \leftarrow \mathcal{D}$ sampled using k nearest examples
```

Manual example of kNN

Let's see this process step by step with a manual example. Suppose we have the classification task illustrated by the points shown in Figure 1.3.

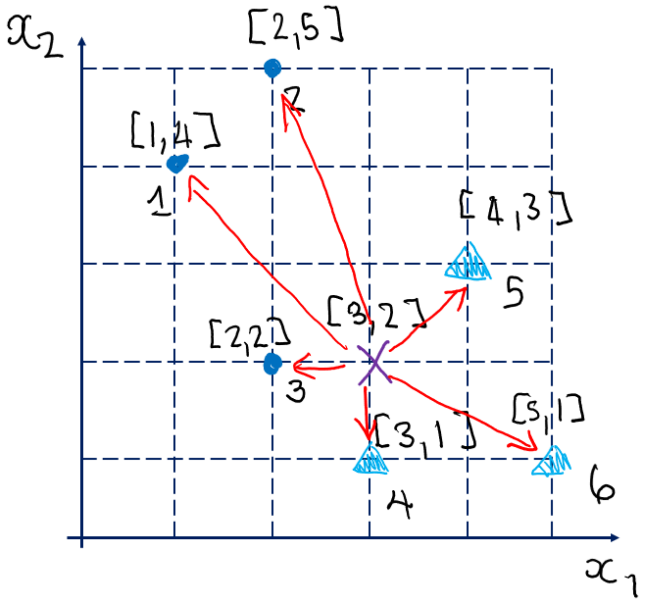


Figure 1.3: A manual example of how a kNN algorithm works

When classifying a new test example the process is as follows:

1. Calculate the distance from the test example to each training example.
2. Sort in ascending order, and keep track of which example they belong to i.e. indices.
3. Create dataset, $\mathcal{D}_k$ by sampling the k nearest examples from, $\mathcal{D}$, with their corresponding classes.
4. Determine the mode of the classes in $\mathcal{D}_k$.
5. The new test example is predicted to belong to this class.

In Figure 1.3 there are two possible classes. We could say that $y \in 0, 1$. Suppose the blue dots belong to class '1' and the green triangles belong to class '0' [10] The dataset would be described as:

10: The classes are arbitrary and in this case indicated by integers. There are other representations for classes that we will come across.

$$\mathcal{D} = \{([1,4],1), \dots, ([5,1],0)\}$$

Using the numbering conventions shown in the figure to indicate the indices of each training example [11]. The test example is given by $\mathbf{x}^{[t]} = [3,2]$. Suppose the Euclidean distance is chosen as the distance metric, then the list of all the distances between each training example and the test example would be:

11: This is completely arbitrary - just make sure the feature vectors and labels correspond

$$dist\_list = [2.83, 3.16, 1.0, 1.0, 1.41, 2.23]$$

If this is sorted by distance in ascending order, but the corresponding indices are returned, the result would be:

$$sorted\_indices = [3, 4, 5, 6, 1, 2]$$

If $k = 3$ then

$$\mathcal{D}_k = \{(\mathbf{x}^{[3]}, \mathbf{y}^{[3]}), (\mathbf{x}^{[4]}, \mathbf{y}^{[4]}), (\mathbf{x}^{[5]}, \mathbf{y}^{[5]})\}$$

The hypothesis function would then yield:

$$\begin{aligned} h(\mathbf{x}^{[t]}) &= mode(\{\mathbf{y}^{[3]}, \mathbf{y}^{[4]}, \mathbf{y}^{[5]}\}) \\ h([3,2]) &= mode\{1, 0, 0\} \\ &= 0 \end{aligned}$$

This should give you an even clearer idea about how kNNs work.

Decision boundaries

An example of the decision boundaries that could were learned by a kNN algorithm I implemented is shown in Figure 1.4 The misclassified examples appear reasonable. Some of the misclassified purple test examples appear to be outliers that fall within similar regions of the green class. The lack of separation in the center region intuitively makes classification of those examples difficult.

Some problems with kNNs

If you attempted to apply kNNs directly to an image recognition task (see Figure 1.5) with high-dimensional inputs [12] the kNN algorithm might struggle. You could potentially reduce the dimensions of the input, for

12: if each pixel is considered an individual feature the input dimensions are very large.

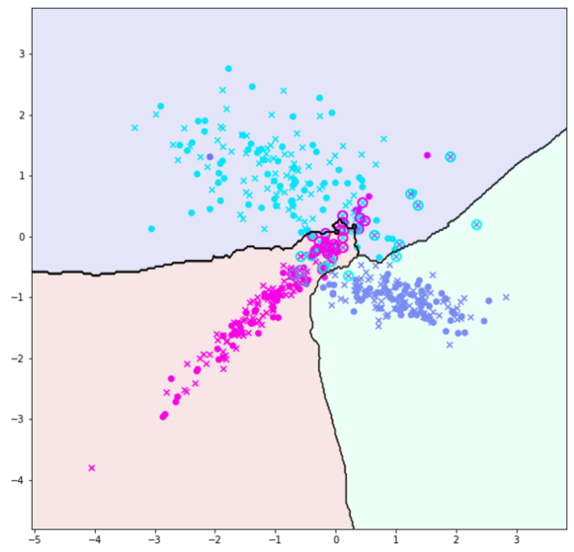


Figure 1.4: Possible decision boundaries on synthetic data with two features and three possible classes. Solid dots represent training examples. Crosses represent test examples. Circles around crosses indicate misclassified test examples with the colour of the circles indicating the ground truth class

example using the average pixel value across an entire image as one feature. But this would likely cause more issues if there is not enough useful information contained in the reduced number of features [13].

13: We will discuss some techniques for feature selection later

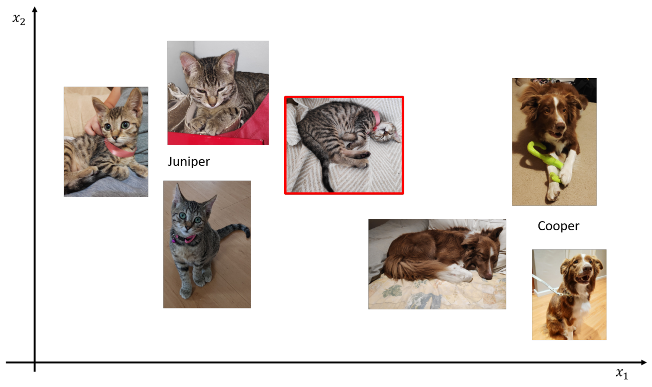


Figure 1.5: A diagram of an image classification task

kNNs need to store all their training data for use when predicting. This is not always feasible, for example, if you are deploying on an embedded system you might not have access to large amounts of storage or memory.

At this stage I would like to move away from specific algorithms to talk about model evaluation. How do we know if a model will behave well on unseen data, can we diagnose specific problems and can we solve them.

Model Evaluation 2

At this stage I would like to move away from learning algorithms and focus on model evaluation. I have touched on ideas such as generalization, however as we start building and training more learning algorithms we need more detail in order to correctly evaluate their performance. A primary goal for machine learning is to ensure that when we train a model we have a good estimate about how it will perform on unseen data. In this chapter we will describe a few important characteristics of machine learning models that describe how well they might generalize.

I have mentioned before that one relevant way to think about generalization is how students are evaluated at University [1]. If a student learns from a set of learning material such practice questions, worked examples, assignments etc. Evaluating them on this exact learning material provides a poor estimate of their future performance on tasks that are not exactly the same. Instead of considering a student to be competent for a given course based on seen material, we try evaluate them with previously unseen material. [2]

1: at least we hope it works this way

2: Of course it is possible that students are poorly evaluated and the same can be said of machine learning algorithms.

2.1 Overfitting and Underfitting

Overfitting and *underfitting* are terms used to describe how a model has performed on seen and unseen data.

Overfitting Models that exhibit good performance on seen data but perform poorly on unseen data are often considered to be *overfit*.

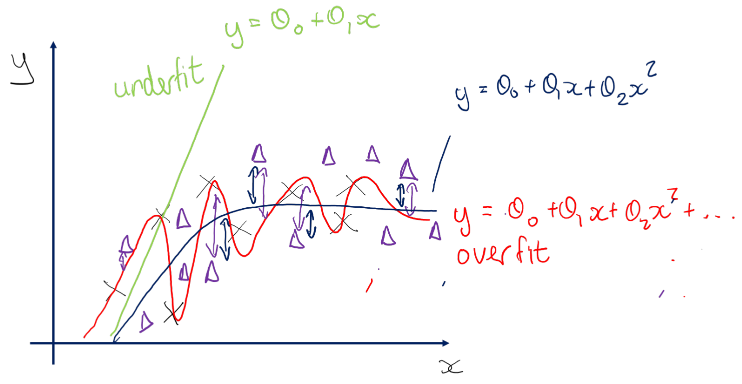
Underfitting Models that exhibit poor performance on both seen and unseen data are considered to be *underfit*

Thinking about the student analogy, if a student evaluates themselves on already studied training material and they perform well, but then go on to perform badly on unseen test questions they would be considered to have overfit their training data. if a student evaluates themselves on already studied training material and they perform poorly, it is unlikely they would perform any better on unseen test questions and they would be considered to be underfit.

Consider the scenario in Figure 2.1 The simple model is unable to perform well on the training data, nor is it capable of generalizing to the test data, which means it is underfitting. The high order polynomial is too flexible which leads to high performance on the training data, but there is a large error with regards to the test data. The 'sufficiently' complex model is flexible enough to produce a hypothesis that produces suitable training and test error.

Figure 2.2 illustrates how these concepts could be interpreted from an error curve plotted with respect to model capacity [3] Low capacity models have a small hypothesis space (small number of possible solutions) and might not be able to reach an optimal error because of this. High capacity

3: Capacity is an indication of the size of the hypothesis space. A higher capacity means the model could learn more possible solutions.



y
x
underfit
$y = \theta_0 + \theta_1 x$
$y = \theta_0 + \theta_1 x + \theta_2 x^2$
$y = \theta_0 + \theta_1 x + \theta_2 x^3 + \ldots$
overfit

Figure 2.1: An arbitrary regression task with a single input feature. The crosses indicate training examples and the purple triangles indicate test examples. The green line indicates a simple model with few parameters and features. The red curve indicates a high order polynomial. The blue curve indicates a 'sufficiently' complex model.

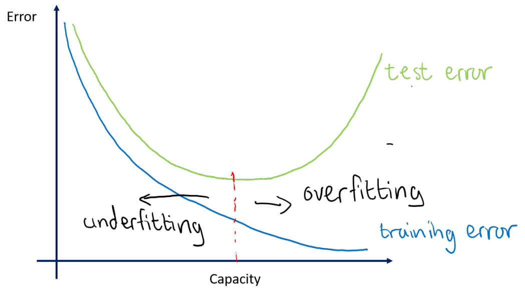


Figure 2.2: A plot of how training and test error compare to model complexity. In general the larger the capacity (hypothesis space) the greater the risk of overfitting.

models have a large hypothesis space, so they might find better solutions to the training data, but there is a potential that these solutions are too specific to the training data and the model may not generalize to unseen data. High training and test error is an indication the model is underfitting. A low training error but high test error is an indication of overfitting.

2.2 Bias and Variance

Bias and variance are terms that you will frequently come across to describe the performance of a machine learning algorithm.

Bias You can think of this type of *bias* as we do in electrical engineering when talking about parameters that are offset from some nominal value. In particular, consider we have learned some hypothesis function (parameterized by θ) that attempts to predict a ground truth function. The bias would be considered the difference between the expected value of the learned parameter and the true parameter. Bias would be an indication of how incorrect our learned parameters are from the true parameters.

Variance You can think of variance as a measure of how differently a given algorithm might learn depending on the training data it is presented with.

Figure 2.3 is a commonly used way to visualise what we mean by bias and variance. The top left would illustrate a model with low variance (small spread in the outcomes) and low bias (small offset from the objective i.e. bullseye). The bottom left would illustrate a model with low variance and

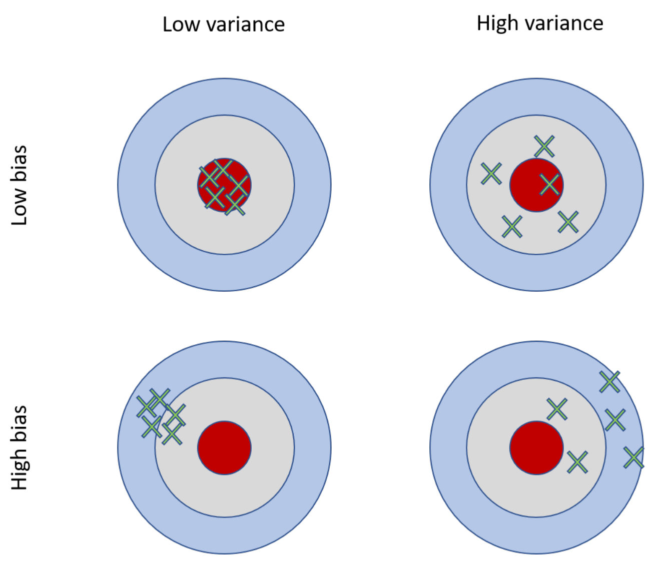


Figure 2.3: An illustration of the differences between bias and variance

high bias (with different training data is still produces similar outcomes, however those outcomes are not likely to meet the desired objective) The top right has low bias but high variance (the model could produce some outcomes that meet the objective, but it could also produce many undesirable outcomes depending on the data it learned from). Lastly, the bottom right has high bias and high variance (the model is unable to produce desirable outcomes and those outcomes vary greatly depending on the data it learned from).

The aim is to have both low bias and variance so that overall accuracy is high regardless of what data was used for training (more points on the bullseye) This should provide a clear idea about the kinds of generalization errors that may occur.

Bibliography

[1] Gareth James et al. *An introduction to statistical learning*. Springer, 2017 (cited on pages 1, 2).

[2] Sebastian Raschka. *What is the difference between a parametric learning algorithm and a nonparametric learning algorithm?* 2020. URL: https://sebastianraschka.com/faq/docs/parametric_vs_nonparametric.html (cited on page 1).

[3] Sebastian Raschka. *STAT 479: Machine Learning Lecture Notes*. http://pages.stat.wisc.edu/~sraschka/teaching/stat479-fs2019. 2019 (cited on page 2).