# O3

ITMAL-01 Machine Learning

**Group 1**
Andreas Skov Jensen (201607648),
Emil Munch Quist (201606564),
Emil Kruse Sørensen (201609286)
14/04-2021

# Contents

**L07 - Capacity and under/overfitting** **2**
a) Polynomial fitting . . . . . 2
b) The concept of capacity and under/overfitting . . . . . 3
c) Scoring strategy in model evaluation . . . . . 3

**L07 - Generalization error** **5**
a) Relationship between capacity and error . . . . . 5
b) Construction of the error plot . . . . . 5

# L07 - Capacity and under/overfitting

In this exercise we have to work with the concepts of capavity and under/overfitting.

## a) Polynomial fitting

We have to go through the provide code and explain what happens. In stead of writing one large summary, we will break the code into smaller pieces and explain what happens step by step. The first part of the code is:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

def true_fun(X):
    return np.cos(1.5 * np.pi * X)

def GenerateData(n_samples = 30):
    X = np.sort(np.random.rand(n_samples))
    y = true_fun(X) + np.random.randn(n_samples) * 0.1
    return X, y

np.random.seed(0)

X, y = GenerateData()
```

Firstly different libraries/functions are imported via `import`. Then the function `true fun(X)` is defined. This is the function we want to approximate. The function `GenerateData()` generates sample points that lies very close to `true fun(X)` but not exactly on it. These sample points will used in the approximations. `np.random.seed(0)` makes sure the same random numbers will be generated every time. Finally the vectors `X` and `y` are generated via `GenerateData()`.

The next part of the code is:

```
degrees = [1, 4, 15]

print("Iterating...degrees=",degrees)
plt.figure(figsize=(14, 5))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)
    plt.setp(ax, xticks=(), yticks=())

    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)

    linear_regression = LinearRegression()
    pipeline = Pipeline([
            ("polynomial_features", polynomial_features),
            ("linear_regression", linear_regression)
        ])
    pipeline.fit(X[:, np.newaxis], y)
```

First it is chosen what polynomial degrees are used in the approximations in the vector `degrees`. In the next section a for-loop is started. Here a pipeline is created. In `polynomial features` the polynomial class is defined via `PolynomialFeatures()`. This will create a matrix containing

polynomial combinations. By choosing `include bias=False` no incerpecting term is included. Next in the pipeline the `LinearRegression()` class is chosen as regressor. Finally the data is fitted to the choosen polynomial via `pipeline.fit()`.

The next part is also part of the for-loop:

```
# Evaluate the models using crossvalidation
scores = cross_val_score(pipeline, X[:, np.newaxis], y, ...
scoring="neg_mean_squared_error", cv=10)

score_mean = scores.mean()
print(f"  degree={degrees[i]:4d}, score_mean={score_mean:4.2f}, ...
{polynomial_features}")
```

In this part of the code cross valdiation is implented via `cross val score`, to evaluate the different polynomial models. It chosen that the scoring method is `scoring="neg mean squared error"` which corresponds to MSE. Also setting `cv=10` splits the training data into 10 folds. Finaly the scores of the different iteartions and the mean score of each model is printed. The fourth order polynomial turns out to be the best approxiamtion, since it has the lowest score.

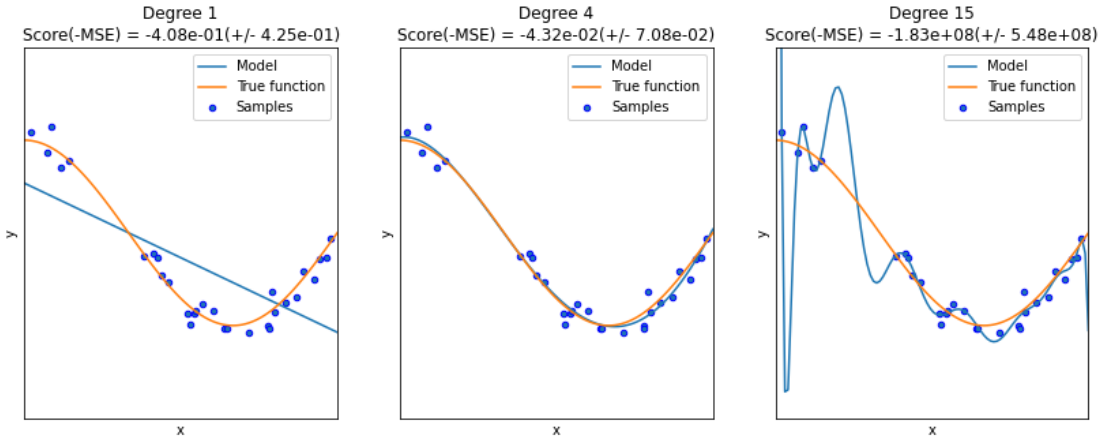


Figure 1: Polynomial fitting of different orders to part of $cos(x)$ function.

## b) The concept of capacity and under/overfitting

Now we have to explain capacity and under/overfitting conceptually. Underfitting occurs when the model we are trying to use on the data does not have the sufficient "resolution". This is what happens on Figure 1 for the linear model. Overfitting on the other side is when the model has to high "resolution". Then the model will start being capturing the noise in the data. This is what happens for Figure 1 for the 15th degree polynomial.
Capacity describes a models capability to describe data. So a model with a high capacity is capable of describing data with high complexity. For the polynomial models we have just seen, the capacity increases with polynomial degree. However just increasing model capacity is not necessarily good. As we have just seen on Figure 1 to high capacity can lead to overfitting wheres to low capacity can lead to underfitting. So one will have to find th eoptimal capacity.

## c) Scoring strategy in model evaluation

In `sklearn` model evaluation tools such as `cross val score` relies on a scoring parameter. In these model evaulations, the convention is that *"higher return values are better than lower return values"* [1]. Normally a high positive value of the MSE would mean that the model does not fit the

data well. However since we want to use MSE as the scoring function, we will take the negative value of the `MSE`, which leads to the use of `neg mean squared error`. This is line with scoring convention in model evaluations. If we try to run the code with `"mean squared error"`, we will get a lot of errors and finally, it outputs:

```
ValueError: 'mean_squared_error' is not a valid scoring value. ...
Use sorted(sklearn.metrics.SCORERS.keys()) to get valid options.
```

This makes sense since as the range for the scoring parameters now are $[-\infty : 0]$, and as the MSE returns a value greater than 1, this will produce an error. All of the above are in line with the fact, that the 15th degree polynomial is the worst of the three models, as it produces the largest negative scoring value.

# L07 - Generalization error

In this section we will look into training models and the optimal capcity of a model.

## a) Relationship between capacity and error

Below on Figure (2) we see the relationsship between errors and capacity.

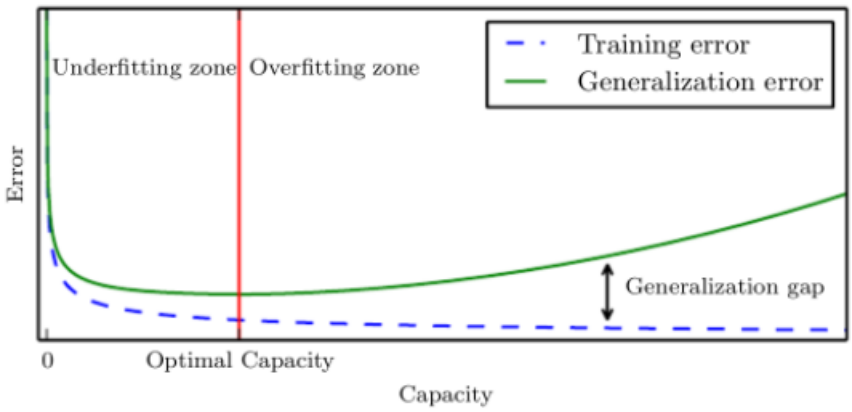


Figure 2: Figure 5.3 from Deep Learning, Ian Goodfellow, et. al. [DL]

As is was described in "L07 - Capacity and under/overfitting b)" capacity describes a models ability to describe data. The higher the capacity the higher complexity in data can be handled. On the x-axis of Figure (2) is model capacity, which increases to the right (for example higher order polynomial). On the y-axis of Figure (2) is the error. This could e.g. be given in RMSE or MSE.
The training error, represents the performance on the training set. The generalization error, represents the performance on the test set. Imagine our data came from some nth order polynomial. If the model is chosen as a polynomial of lower order than n, then it will have a hard time describing the data, and hence both training error and generalization error will be relativity large. This is called the underfittng zone. This is especially true on the far left of the graph, where the model is first constant (0th order) and linear (1st order).
However as the models polynomial order increases and approaches n, both training and generalization error decreases. When the model uses an nth order polynomial the generalization error is at its minima, this is called the optimal capacity, where the model performs best on the unknown test set - which is what we want.
After this the generalization error will increase due to overfitting, and we enter the overfitting zone. The training error continues to decline as the model of higher order than nth polynomial now will start to be influenced by the noise in the training data. The generalization gap is the difference between training and generalization error. It grows continuesly from the underfitting zone thorugh the overfitting zone.

## b) Construction of the error plot

In this section we will start by going through the provided code. We start with part I Firstly 100 data points of some 2nd order polynomial with some noise in it, is created via `GenerateData()`. This is then spilt into training and validation sets through `train test split`. Then through the pipeline `poly scaler` it is chosen that the model used for regression is a 90th degree polynomial. Then in

# References

[1] scikit learn. *3.3. Metrics and scoring: quantifying the quality of predictions.* `https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter`, 2021. Visited: 17-03-2021.