# Machine Learning

Centre for Data Science, ITER
Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India.

## Contents

## Introduction

- Data science is mostly turning business problems into data problems.
- Collecting data and understanding data and cleaning data and formatting data.
- After all these, machine learning is almost an afterthought.

## Modeling

- Modeling is a specification of a mathematical (or probabilistic) relationship that exists between different variables.
- e.g. A cookbook recipe entails a model that relates inputs like “number of eaters” and “hungriness” to quantities of ingredients needed.
- The business model is probably based on simple mathematical relationships: profit is revenue minus expenses

## Machine Learning

- It is creating and using models that are learned from data. Also, used to predict something.

1. Whether an email message is spam or not.
2. Whether a credit card transaction is fraudulent.
3. Which advertisement a shopper is most likely to click on.
4. Which next movie is good to suggest.

## Types of Machine Learning

- Supervised Machine Learning: Dataset has labeled data and using machine learning techniques on it.
- Unsupervised Machine Learning: Dataset is unlabeled, and machine learning technique is used on it.
- Semi- supervised Machine Learning: Data is partly labeled.
- Reinforcement Machine Learning: After a series of predictions, the model gets to know how well it is doing.

## Overfitting and Underfitting

- **Overfitting:** producing a model that performs well on the data you train it on but generalizes poorly to any new data.
- This could involve noise in the data.
- **Underfitting:** producing a model that doesn't perform well even on the training data.
- **Noise:** Data with a large amount of additional meaningless information in it called noise.

## Overfitting and Underfitting

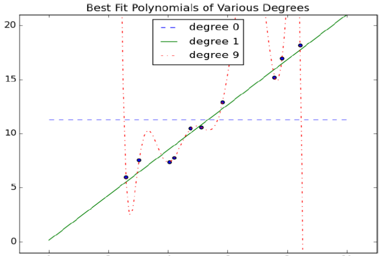

*Figure 11-1. Overfitting and underfitting*

- Degree 0 severely underfits the data.
- Degree 9 severely overfits the data.
- Degree 1 strikes a nice balance, pretty close to every point and even if some new data comes in, this line might be close to it too.

## Train and Test Data

- Most fundamental approach involves using different data to train the model and to test the model.
- The simplest way to do this is to split the dataset, so that (for example) two-thirds of it is used to train the model.
- Remaining one-third can be used to test the model.

## Splitting the data

```
import random
from typing import TypeVar, List, Tuple
X = TypeVar('X') # generic type to represent a data point
def split_data(data: List[X], prob: float) -> Tuple[List[X],
    List[X]]:
  """Split data into fractions [prob, 1 - prob]"""
  data = data[:] # Make a shallow copy
  random.shuffle(data) # because shuffle modifies the list.
  cut = int(len(data) * prob) # Use prob to find a cutoff
  return data[:cut], data[cut:] # and split the shuffled
      list there.
data = [n for n in range(1000)]
train, test = split_data(data, 0.75)
# The proportions should be correct
assert len(train) == 750
assert len(test) == 250
# And the original data should be preserved (in some order)
assert sorted(train + test) == data
```

```
Y = TypeVar('Y') # generic type to represent output variables
def train_test_split(xs: List[X],ys: List[Y],test_pct:
    float) -> Tuple[List[X], List[X], List[Y],List[Y]]:
   # Generate the indices and split them
   idxs = [i for i in range(len(xs))]
   train_idxs, test_idxs = split_data(idxs, 1 - test_pct)
   return ([xs[i] for i in train_idxs], # x_train
        [xs[i] for i in test_idxs], # x_test
        [ys[i] for i in train_idxs], # y_train
        [ys[i] for i in test_idxs]) # y_test
xs = [x for x in range(1000)] # xs are 1 ... 1000
ys = [2 * x for x in xs] # each y_i is twice x_i
x_train, x_test, y_train, y_test = train_test_split(xs, ys,
    0.25)
# Check that the proportions are correct
assert len(x_train) == len(y_train) == 750
assert len(x_test) == len(y_test) == 250
# Check that the corresponding data points are paired
    correctly
assert all(y == 2 * x for x, y in zip(x_train, y_train))
assert all(y == 2 * x for x, y in zip(x_test, y_test))
```

- If the model performs well on the test data, then one can be more confident that it's fitting rather than overfitting.
- There should not be common patterns in the test and training data that wouldn't generalize to a larger dataset.
- Choosing a model that performs best on the test set should be done very precautiously.

**To check model suitability**

Split the data into three parts: a training set for building models, a validation set for choosing among trained models, and a test set for judging the final model.

## Correctness

- Given a set of labeled data and a predictive model, every data point lies in one of four categories.

1. True positive: “This message is spam, and we correctly predicted spam.”
2. False positive (Type 1 error): “This message is not spam, but we predicted spam.”
3. False negative (Type 2 error): “This message is spam, but we predicted not spam.”
4. True negative: “This message is not spam, and we correctly predicted not spam.”

## Confusion Matrix

| **Confusion Matrix** | Spam | Not Spam |
|---|---|---|
| Predict 'spam' | True Positive | False positive |
| Predict 'not spam' | False Negative | True Negative |

- Using the data available in the confusion matrix, various things can be calculated, given below.
- Accuracy$= \frac{correct}{total} = \frac{tp+tn}{tp+tn+fp+fn}$
- Precision=How accurate our positive predictions can be$= \frac{tp}{tp+fp}$
- Recall=Fraction of positives model identified=$\frac{tp}{tp+fn}$
- Precision and recall can be combined to form f1_score.
- f1_score=$\frac{2*p*r}{p+r}$ ;Harmonic mean of precision and recall and lies between them

- Usually the choice of a model involves a tradeoff between precision and recall.
- Generally, a high recall may relate to low precision.
- So, choosing a correct threshold is important, as if one tries to increase the precision (one will have to involve stricter constraints to be positive) which will eventually lead to positives not meeting the threshold, i.e. decreasing the recall.

## Bias-Variance Tradeoff

- High bias: A model which makes lot of mistakes for pretty much any training set.
- Low variance: If any two randomly chosen dataset gives similar results, it is said to have low variance.
- High bias and low variance typically correspond to underfitting.(Remember, degree 0)
- Low bias and High variance will correspond to? (degree 9)
- High bias indicates to add more features.
- High variance indicates to remove features.

## Feature extraction and selection

- Without enough features, model is likely to underfit.
- With too many features, it's easy to overfit.
- Features are whatever inputs we provide to our model.
- Prediction a mail is spam or not involves feature extraction, such as if it contains OTP, if it is asking for bank details, etc.
- Combination of experience and domain expertise comes into play in case to good feature extraction.
- It can also require dimension change(increase or decrease).

## Upcoming methods for feature extraction

- The Naive Bayes classifier we'll build in Chapter 13 is suited to yes-or-no features, like the first one in the preceding list.
- Regression models, which we'll study in Chapters 14 and 16, require numeric features (which could include dummy variables that are 0s and 1s).
- Decision trees, which we'll look at in Chapter 17, can deal with numeric or categorical data.

## References

[1] Data Science from Scratch_First Principles with Python by Joel Grus, *O'Reilly*.

Thank You
Any Questions?