Machine Learning

CENG 499
Introduction to Data Science

Erdoğan Doğdu

Machine Learning

- ML
  - Creating and using <u>models</u> that are learned from data
  - Predictive modelling
  - Data mining
- Model
  - a specification of a mathematical (or probabilistic) relationship that exists between different variables
- Goal
  - Use existing data to develop <u>**models**</u> that we can use to predict various outcomes for new data

Machine Learning

- Examples
  - Predicting whether an email message is **spam** or not
  - Predicting whether a **credit card transaction** is fraudulent
  - Predicting which **advertisement** a shopper is most likely to click on
  - Predicting which **football team** is going to win the Super Bowl

Machine Learning Models

- Supervised
  - there is a set of data labeled with the correct answers to learn from
- Unsupervised
  - there are no such labels
- Semisupervised
  - only some of the data are labeled

Overfitting and Underfitting

- *Overfitting*
  - producing a model that per- forms well on the data you train it on but that generalizes poorly to any new data
- *Underfitting*
  - producing a model that doesn’t perform well even on the training data
    - Keep searching for a working model

Overfitting and Underfitting

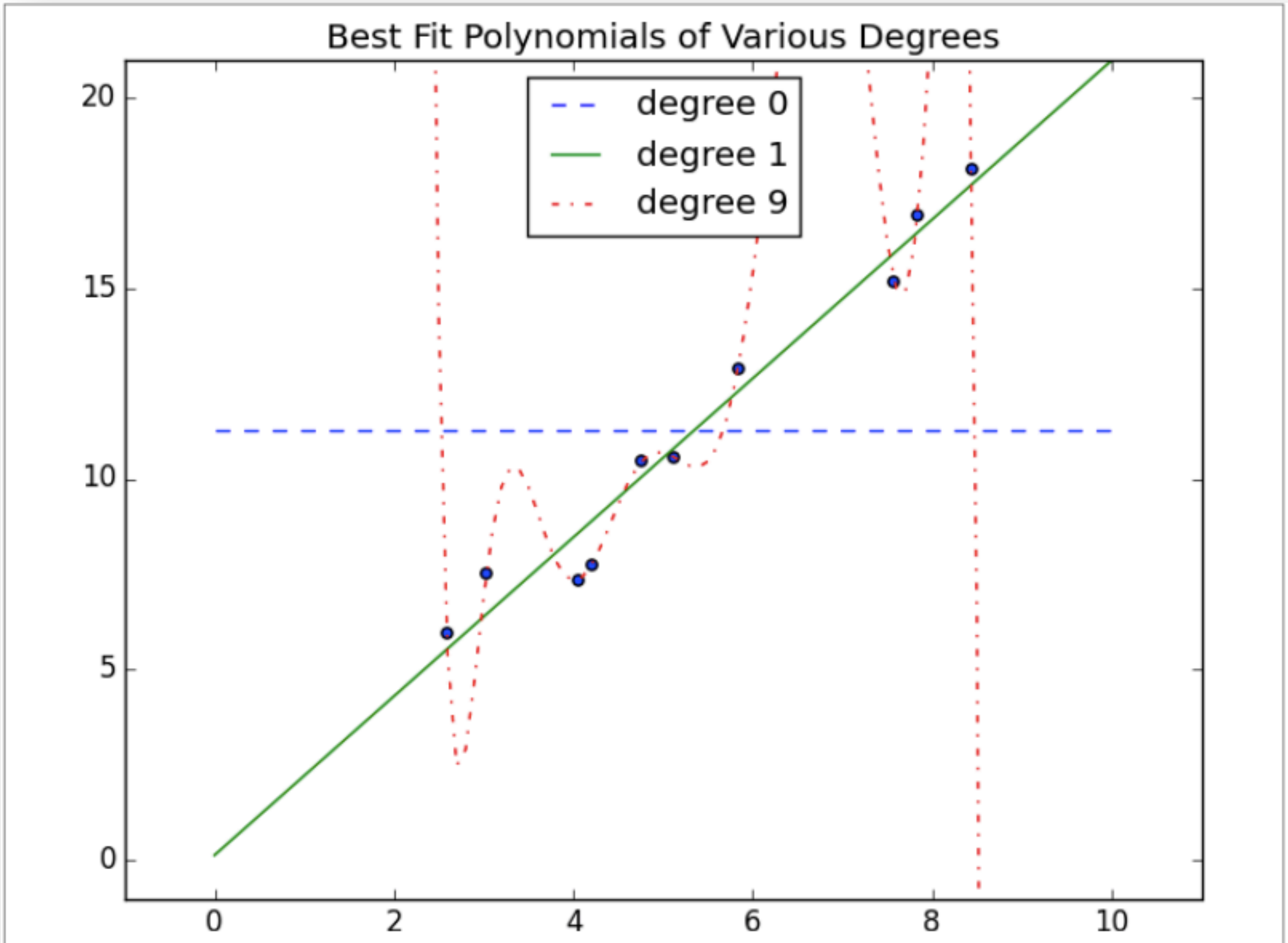

Figure 11-1. Overfitting and underfitting

Overfitting and Underfitting

- Models that are too **complex** lead to overfitting and don't generalize well beyond the data they were trained on
- Use different data to **train** the model and to **test** the model.
  - Example: Split data into 3 parts, use 2 for training, and 1 for testing (66% vs 33%)

Train vs Test Split

```python
def split_data(data, prob):
    """split data into fractions [prob, 1 - prob]"""
    results = [], []
    for row in data:
        results[0 if random.random() < prob else 1].append(row)
    return results
```

```python
def train_test_split(x, y, test_pct):
    data = zip(x, y)                              # pair corresponding values
    train, test = split_data(data, 1 - test_pct)  # split the data set of pairs
    x_train, y_train = zip(*train)                # magical un-zip trick
    x_test, y_test = zip(*test)
    return x_train, x_test, y_train, y_test
```

```python
model = SomeKindOfModel()
x_train, x_test, y_train, y_test = train_test_split(xs, ys, 0.33)
model.train(x_train, y_train)
performance = model.test(x_test, y_test)
```

Overfitting

- Still might be overfitting if
  - there are common patterns in the test and train data that wouldn't generalize to a larger data set.
- Or, if you are using test set to choose from a number of models,
  - Then split data into train, validate, test sets

Correctness

- Given a set of "labeled" data and a predictive model, data points lie in one of these categories:
  - **True positive**: "This message is spam, and we correctly predicted spam."
  - **False positive** (Type 1 Error): "This message is not spam, but we predicted spam."
  - **False negative** (Type 2 Error): "This message is spam, but we predicted not spam."
  - **True negative**: "This message is not spam, and we correctly predicted not spam."

Confusion Matrix

	Spam	not Spam
predict "Spam"	True Positive	False Positive
predict "Not Spam"	False Negative	True Negative

Accuracy

- Predict **leukemia** if and only if the baby is named **Luke** (which sounds sort of like "leukemia")

	leukemia	no leukemia	total
"Luke"	70	4,930	5,000
not "Luke"	13,930	981,070	995,000
total	14,000	986,000	1,000,000

```python
def accuracy(tp, fp, fn, tn):
    correct = tp + tn
    total = tp + fp + fn + tn
    return correct / total

print accuracy(70, 4930, 13930, 981070)     # 0.98114
```

Good?

Precision & Recall & F1

- **Precision**: how accurate our *positive* predictions were

```
def precision(tp, fp, fn, tn):
    return tp / (tp + fp)

print precision(70, 4930, 13930, 981070)    # 0.014
```

- **Recall**: what fraction of the positives our model identified

```
def recall(tp, fp, fn, tn):
    return tp / (tp + fn)

print recall(70, 4930, 13930, 981070)       # 0.005
```

- ***F1**: harmonic mean* of precision and recall and necessarily lies between them

```
def f1_score(tp, fp, fn, tn):
    p = precision(tp, fp, fn, tn)
    r = recall(tp, fp, fn, tn)

    return 2 * p * r / (p + r)
```

Model Correctness

- Choice of a model involves a **trade-off** between precision and recall.
  - A model that predicts "yes" when it's even a little bit confident will probably have a high recall but a low precision;
  - A model that predicts "yes" only when it's extremely confident is likely to have a low recall and a high precision.

Feature Extraction and Selection

- ***Features****:* inputs we provide to our model
- When your data doesn't have enough features, your model is likely to **underfit**.
- When your data has too many features, it's easy to **overfit**.

Feature Engineering

- Simple case: features are given
  - Example:
    - Given: Years of experience, Salary
    - Predict: Salary based on years of experience
- Extract features:
  - Spam detection
    - Given: Email texts
    - Extract features:
      - f1: Does the email contain the word “Viagra”?
      - f2: How many times does the letter d appear?
      - f3: What was the domain of the sender?

Model Type

- The type of features constrains the type of model
- Yes-No features
  - Naïve Bayes model
- Numeric features
  - Regression model
- Numeric or categorical features
  - Decision trees model

Feature Engineering

- Remove features
  - Dimensionality reduction
  - Regularization
- How to do all of these:
  - Experience
  - Domain expertise