Machine Learning Introduction

for Natural Language Processing

Niklas Schnelle

8. November, 2017

Overview

Introduction

"... the [Analytical Engine] might compose elaborate and scientific pieces of music of any degree of complexity or extent." Ada Lovelace

Historical Background

As seen by the previous quote using computation for tasks such as composing, reasoning or understanding language is quite old. Historically research into such *"Artificial Intelligence"* has seen two fundamental approaches

- **Symbolic AI** based on logic and reasoning
  - Checkers and Chess playing programs (Christopher Strachey, Arthur Samuel 1950s)
  - Prolog (Alain Colmerauer, 1972)
- **Probablistic Models** based on methods from statistics sometimes inspired by neuroscience
  - Perceptron Algorithm (Frank Rosenblatt, 1957)

Machine Learning Approaches

Machine Learning works by learning a model from data, depending on the type of data different approaches are used

- **Supervised Learning** given examples $\{(x, y)\}$ learn to predict target label y from input x
  - Classification: y is a set of class labels (e.g. genres, part-of-speech, image labels)
  - Regression: y is a real number
- **Unsupervised Learning** given a set of data $\{x\}$ without an explicit target y
  - Uncover structure e.g. components of maximum variation (PCA), clusters
  - Create new representations such as an embedding in some vector space
- **Other Types**
  - Semi-Supervised, Reinforcement Learning etc.

Linear Regression

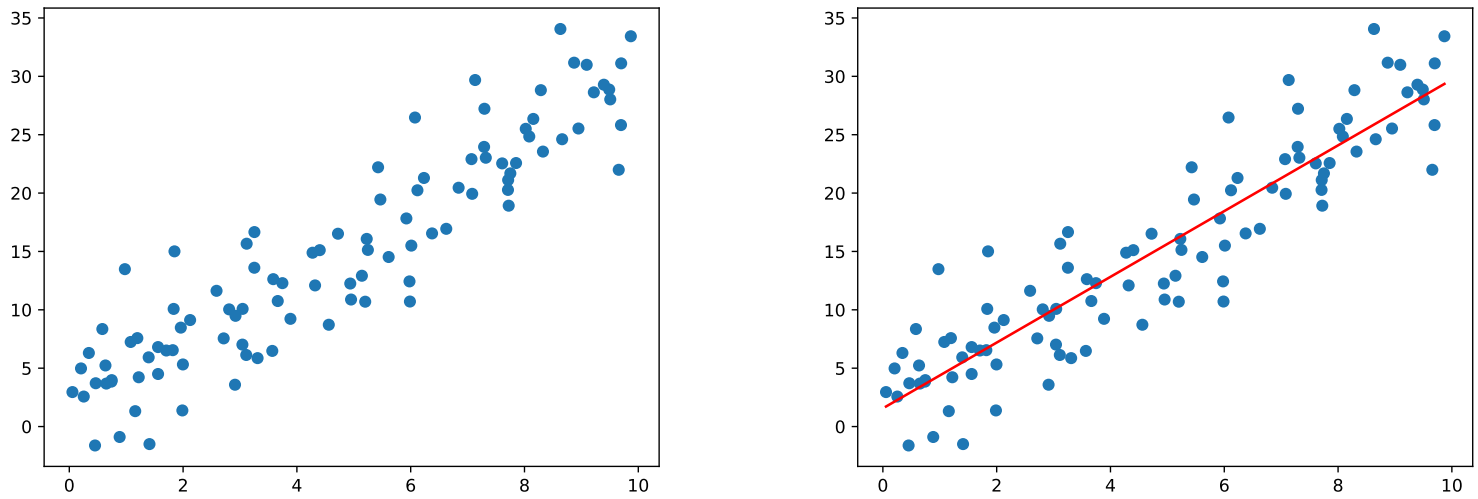

Figure: We want to estimate (for one author) the number of spelling errors for a document of arbitrary length. We may plot known spelling errors vs document lengths and then fit a line to estimate the rate of errors (the m in $f(x) = mx + c$).

Linear Regression

First Model Linear Regression: Given an input vector $x \in \mathbb{R}^d$ we want to learn a mapping to an output value $y \in \mathbb{R}$

- Model with parameters $\beta = (\beta_0, \beta_1, \ldots, \beta_d)^T \in \mathbb{R}^{d+1}$

$$f(x) = \beta_0 + \sum_{j=1}^{d} \beta_j x_j = (1, x_1, \ldots, x_d)\beta$$

- Given training data $D = \{(x_i, y_i)\}_{i=1}^n$ the *least squares* loss is[0]

$$L^{\text{ls}}(\beta) = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \|y - X\beta\|^2$$

- The optimial $\hat{\beta}$ then is $\hat{\beta} = (X^T X)^{-1} X^T y$

[0] $X \in \mathbb{R}^{n \times d+1}$ are the 1-augmented x_i stacked as row vectors and $y \in \mathbb{R}^n$ the stacked y_i

Linear in the Input

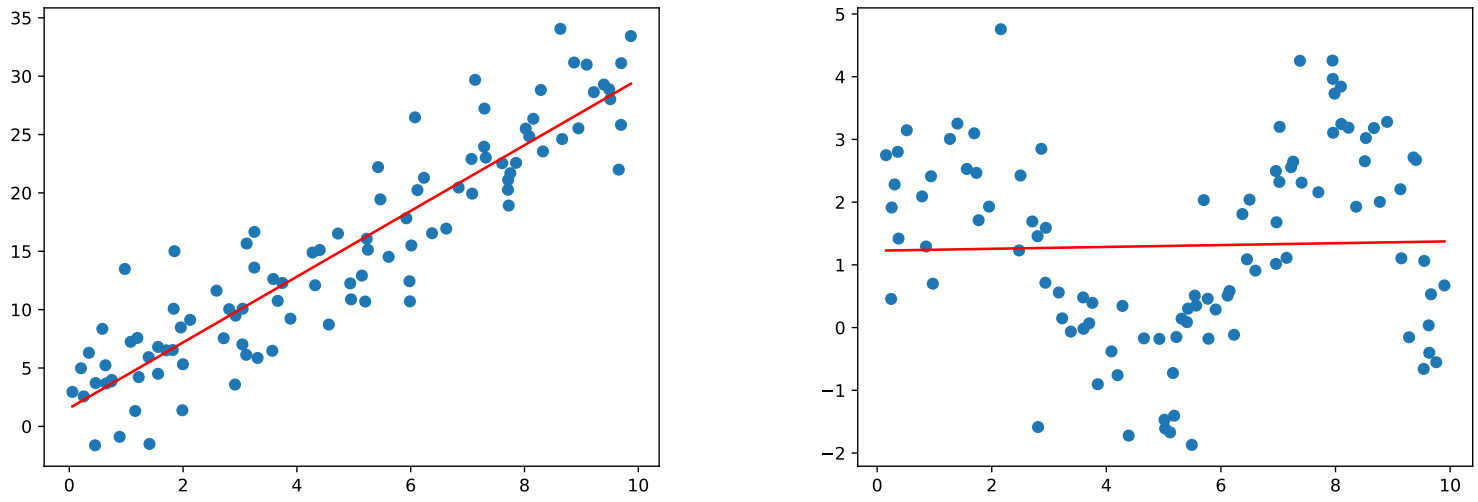

Figure: Linear Regression directly on the input only works for linear data

Introducing Features

Clearly we need a more powerful model! Instead of learning a mapping directly on the input values x we first transform them into vectors of their *characteristic properties* the so called features.

- Replace each input $x \in \mathbb{R}^d$ with a feature vector $\phi(x) \in \mathbb{R}^k$

$$f(x) = \sum_{j=1}^{k} \phi_i(x)\beta_j = \phi(x)^T \beta$$

- For a real value x simply adding x^2 using $\phi(x) = (1, x, x^2)$ allows fitting a parabola
- Note that in regression we almost always prepend a 1 to capture constant offsets
- Everything else stays the same including the optimization

Linear in Features

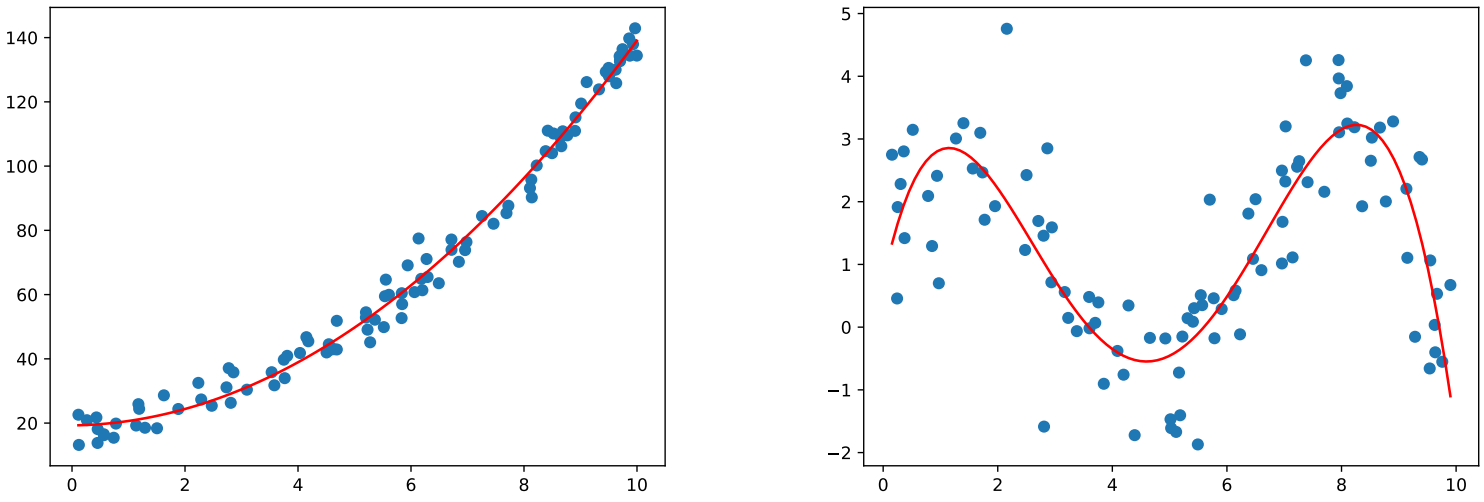

Figure: The same algorithm but parametrized on $\phi(x) = (1, x, x^2)$ instead of just $(1, x)$ allows us to perfectly fit quadratic relationships. Using $\phi(x) = (1, x, x^2, \ldots, x^5)$ we can fit even more complicated functions.

Example Features

Besides allowing us to fit data using complicated mathematical functions, features allow us to express the dependence on arbitrary other data

- The stock value of Facebook as a linear combination of the stock of related companies
  $\phi(x) = (1, x_{\text{APPL}}, x_{\text{GOOG}}, \ldots)$
- The quality of a relationship based on the number and type of emoticons in a chat
  $\phi(\text{msg}) = (1, |\heartsuit \in \text{msg}|, \ldots)$
- The price of oil as a function of keyword counts in Donald Trump's Tweets
  $\phi(\text{tweet}) = (1, |\text{bad} \in \text{tweet}| + |\text{oil} \in \text{tweet}|, \ldots)$

Features for Natural Language Processing

In the previous slides we used features like $|\textbf{bad} \in \textbf{tweet}|$ with handpicked words and dimensions.
To automate this we can encode a word or a fixed sequence of words (n-gram) with:

- **One-Hot Encoding**: For each word w reserve a dimension that is 1 iff $w \in x$ (DictVectorizer with boolean values in Scikit-Learn)
- **Bag-of-Words**: Like One-Hot-Encoding but store the number of occurences (CountVectorizer) or the tf-idf values (TfidfVectorizer) useful for longer x

Depending on the task other useful features may include

- Part-of-speech if known or computed with another model
- Capitalization, length, sub-word sequences (e.g. "-ing"), match with a dictionary...

Overfitting

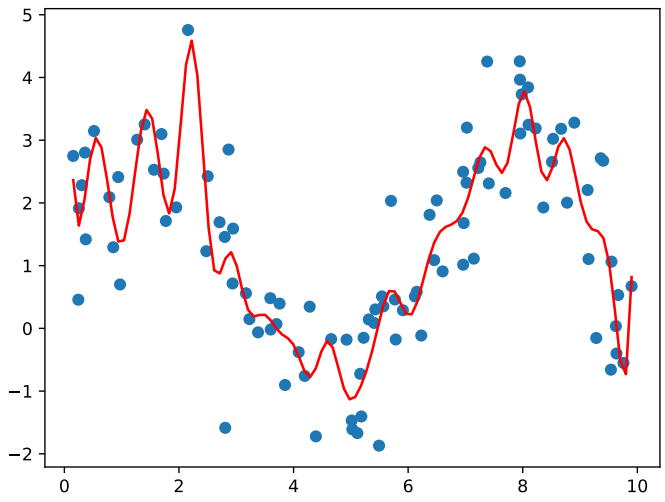

Figure: With more expressive features like Radial Basis Functions $\phi(x) = (1, e^{-\frac{1}{2}\|x-c_1\|}, \ldots)$ we may **overfit** the training data - and our model generalizes badly.

Preventing Overfitting: Regularization & Cross-Validation

To counter overfitting we use two primary techniques (besides adding more data)

- **Regularization** we modify the learning for example by changing the loss function to prevent overfitting
  - For Linear Regression we add a regularization term, penalizing large β_i with regularization factor λ[0]

$$L^{\text{ridge}}(\beta) = \sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \sum_{j=2}^{k} \beta_j^2$$

- **Cross-Validation** to estimate generalization we train only on part of the training set while testing on the rest. We shuffle roles until all data has been in both (KFold class in Scikit-Learn)

[0] The new Optimum is $\hat{\beta} = (X^T X + \lambda I)^{-1} X^T y$ where I is the identity matrix with $I_{1,1}$ set to 0

Regularization

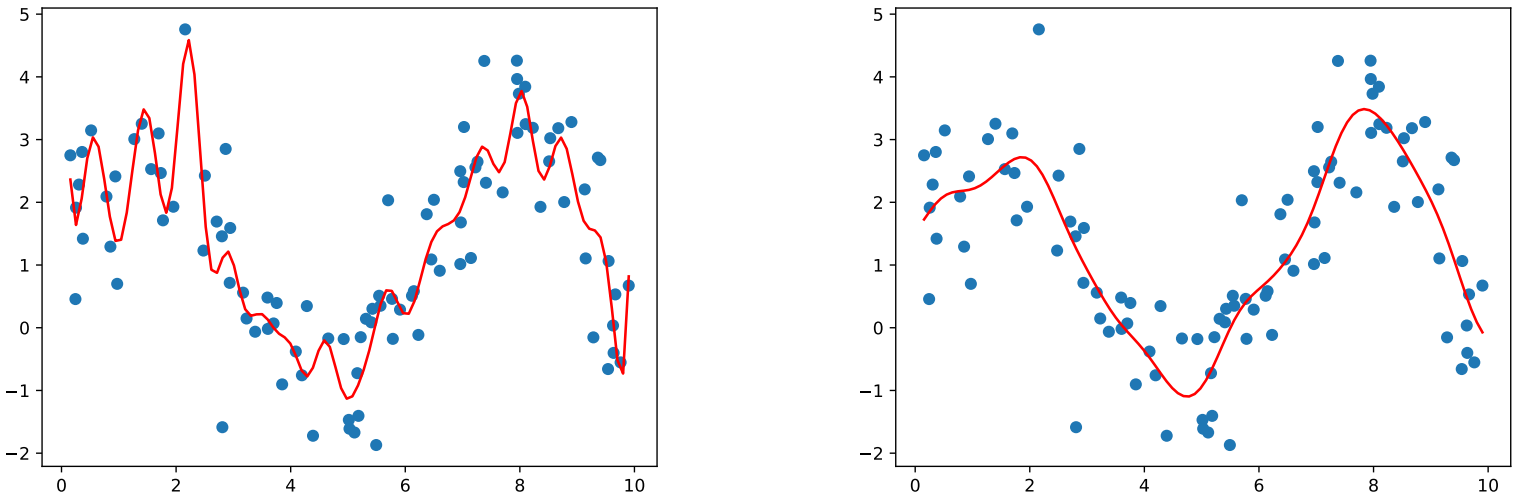

Figure: Using the same Radial Basis Functions and data as before but with Linear Ridge Regression and $\lambda = 0.2$

Classification: Discriminative Function

Often our target variable y is a discrete value from a set of classes Y (part-of-speech, sentiment, topic etc).

- Instead of dealing with a discrete-valued function $F : \mathbb{R}^d \rightarrow Y$ we can learn a **discriminative function**

$$f : \mathbb{R}^d \times Y \rightarrow \mathbb{R}$$

such that

$$\hat{y} = \underset{y}{\text{argmax}}\, f(x, y)$$

- $f(x, y)$ is high if y is the correct class for x and low otherwise

Classification: Separating Hyperplanes

Assume for simplicity we only have two classes $y \in \{0, 1\}$

- Now we can use a simple threshold approach for $f(x, y)$

$$f(x, 1) = \phi(x)^T \beta$$

$$\hat{y} = \operatorname*{argmax}_y f(x, y) = \begin{cases} 0, & f(x, 1) \leq 0 \\ 1, & f(x, 1) > 0 \end{cases}$$

- In feature space β then defines a hyperplane separating the two classes $(1, x_1, x_2, \ldots, x_d)\beta = 0$
- Note that even with this approach we can simulate more classes using multiple classifiers

Classification: Separatable in Higher Dimensions

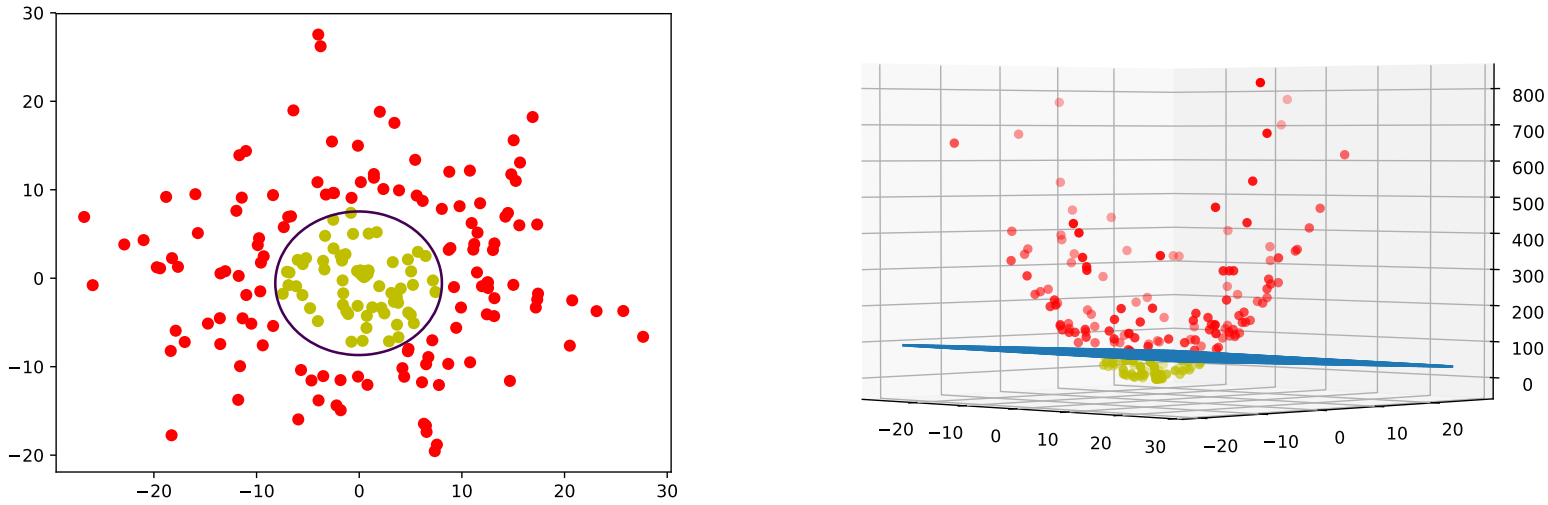

Figure: Two classes of points $(x_1, x_2) \subset \mathbb{R}^2$ become linearly separable when adding $x_1^2 + x_2^2$ as a feature/dimension

Classification: Finding a Separating Hyperplane

It remains to find a suitable separating hyperplane and thus a discriminative function

- **Perceptron**: Choose initial β. Classify each point x and if misclassified nudge β in the right direction by adding/subtracting $\phi(x)$.
- **Support Vector Machines**: Optimize for a maximal margin between the hyperplane and closest points (support vectors)
- **Logistic Regression**: Turn discriminative function into conditional class probabilities and maximize likelihood

Classification: Logistic Regression

Again we look only at the case of two classes $y \in \{0, 1\}$

- Note, that the discriminative function

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, f(x, y) = \begin{cases} 0, & f(x, 1) \leq 0 \\ 1, & f(x, 1) > 0 \end{cases}$$

  can also be interpreted as two functions $f(x, 0)$ and $f(x, 1)$ where $f(x, 0)$ is constant 0

- Now we can write *conditional class probabilities* as

$$p(y = 1|x) = \frac{e^{f(x,1)}}{e^{f(x,0)=0} + e^{f(x,1)}} = \sigma(f(x, 1))$$

- With this and data $D = \{(x_i, y_i)\}_{i=1}^n$ we can define and minimize the neg-log-likelihood[0]

$$L^{\text{logistic}}(\beta) = -\sum_{i=1}^{n} \log p(y_i|x_i) + \lambda\|\beta\|^2$$

[0] See Appendix for how to derive the optimal parameters

Classification: Logistic Regression

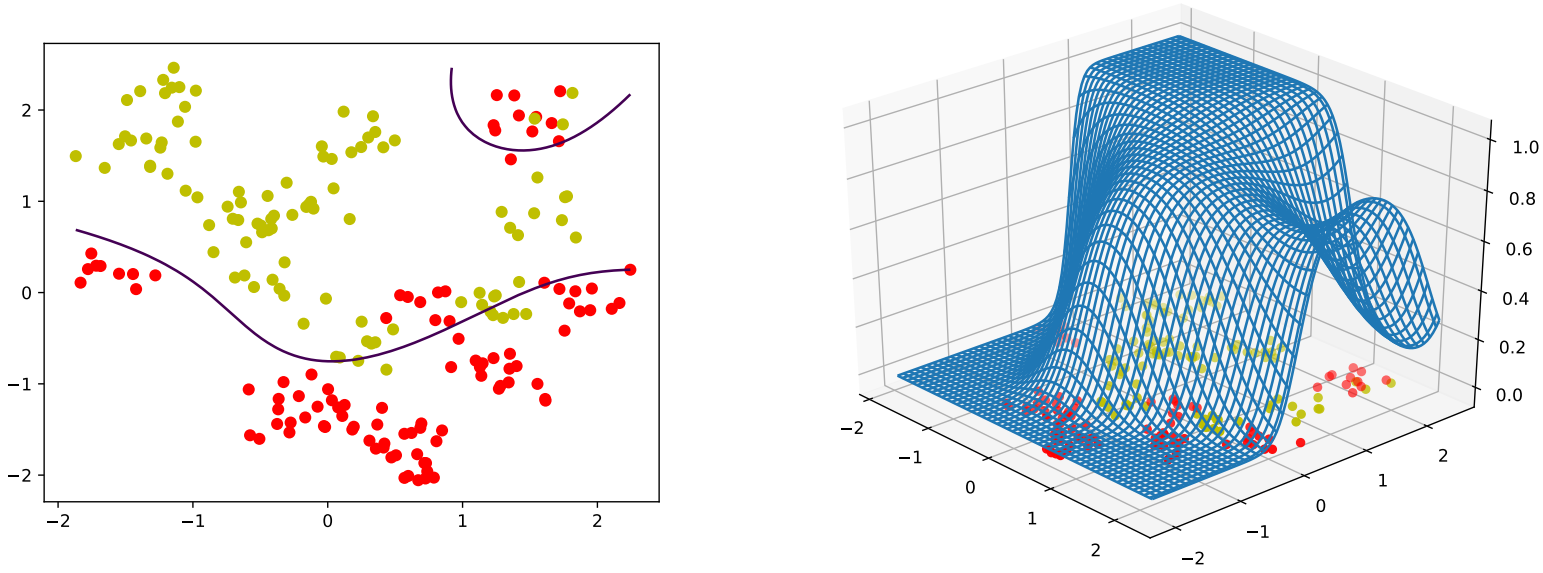

Figure: Using cubic features on $x \in \mathbb{R}^2$ input data. On the left we can see the $P(y = 1|x) > 0.5$ decision boundary in the input space, while the right plot shows the 3D structure of class probabilities $\sigma(f(x, 1))$.

Classification: Other Classical Methods

Besides these linear methods other classical machine learning approaches include

- **Naive Bayes**: Directly apply Bayes' theorem by assuming independence of features
- **Kernel Methods**: Generalize the aforementioned linear models using a similarity measure without needing explicit features
- **Local & Lazy Learning**: Create only a local model around a query point
- **Bootstrapping**: Use multiple models of same type on randomized versions of the input data
- **Boosting**: As bootstrapping but be more clever in selecting data for classifiers
- **Decision Trees**: Build a tree from regions of constant prediction, often used with Boosting

References

This presentation + code for the demos and plots is available at:
`https://ad-git.informatik.uni-freiburg.de/ad/nlpml`
It is based mainly on the following materials:

- Introduction to Machine Learning
  Prof. Marc Toussaint

  `https://ipvs.informatik.uni-stuttgart.de/mlr/marc/teaching/Lecture-MachineLearning.pdf`

- The Elements of Statistical Learning[1]
  Trevor Hastie, Robert Tibshirani, Jerome Friedman

  `https://web.stanford.edu/~hastie/ElemStatLearn/`

Appendix Logistic Regression: Optimal Parameters I

Unlike Linear Regression, Logistic Regression has no closed-form solution

- In the two class case the neg-log-likelihood with data $D = \{(x_i, y_i)\}_{i=1}^n$ is

$$L^{\text{logistic}} = -\sum_{i=1}^{n} \left[y_i \log p(1|x_i) + (1-y_i) \log(1-p(1|x_i)) \right] + \lambda \|\beta\|^2$$

- Its gradient with $p_i := p(y = 1|x_i)$ then is

$$\frac{\partial L^{\text{logistic}}}{\partial \beta} = \sum_{i=1}^{n} (p_i - y_i)\phi(x_i) + 2\lambda I\beta = X^T(p - y) + 2\lambda I\beta$$

Appendix Logistic Regression: Optimal Parameters II

Now we could simply do a gradient descent which can also be done one example at a time and scales very well. Or we can compute the Hessian matrix and use Newton's algorithm to find the minimum.

- The Hessian can be computed as

$$H = \frac{\partial^2 L^{\text{logistic}}}{\partial^2 \beta} = X^T W X + 2\lambda I$$

  where W is the diagonal matrix with
  $W_{i,i} = p(y = 1|x_i)(1 - p(y = 1|x_i))$
- Applying Newton's algorithm then means iterating

$$\beta \leftarrow \beta - H^{-1} \frac{\partial L^{\text{logistic}}}{\partial \beta}^T$$