FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Intent detection and somatosensory feedback

#09: Regression; incrementality and interactive machine learning

Claudio CASTELLINI, Sabine THÜRAUF

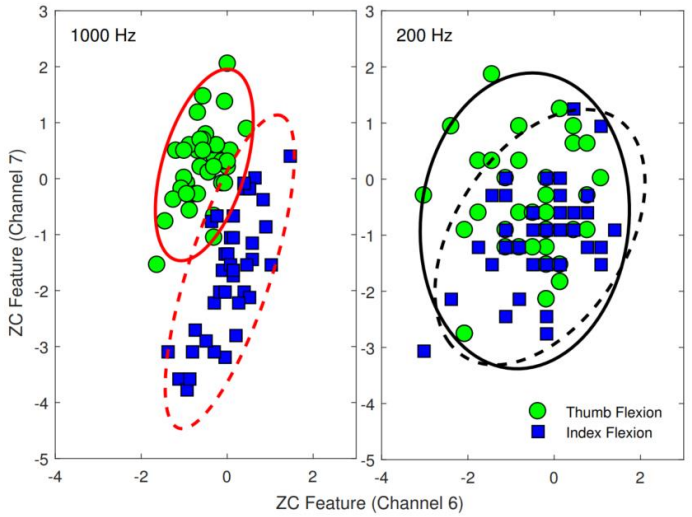


Figure 2. Differences in EMG patterns between using: (**left**) a 1000 Hz sampling rate; and (**right**) a 200 Hz sampling rate. ZC features are extracted from two different EMG channels (6 and 7) during thumb flexion (green circle markers and solid lines) and index flexion (blue square markers and dashed lines). Samples are from Subject 1 of Database 3.

EMG patterns related to two actions. Reproduced from Angkoon Phinyomark, Rami N. Khushaba and Erik Scheme, *Feature Extraction and Selection for Myoelectric Control Based on Wearable EMG Sensors*, MDPI Sensors 2018, 18, 1615

The *rubber hand illusion.* See Botvinick M, Cohen J., *Rubber hands 'feel' touch that eyes see.* Nature. 1998 Feb 19;391(6669):756. doi: 10.1038/35784. PMID: 9486643.

Lecture #09:

Regression; incrementality and interactive machine learning

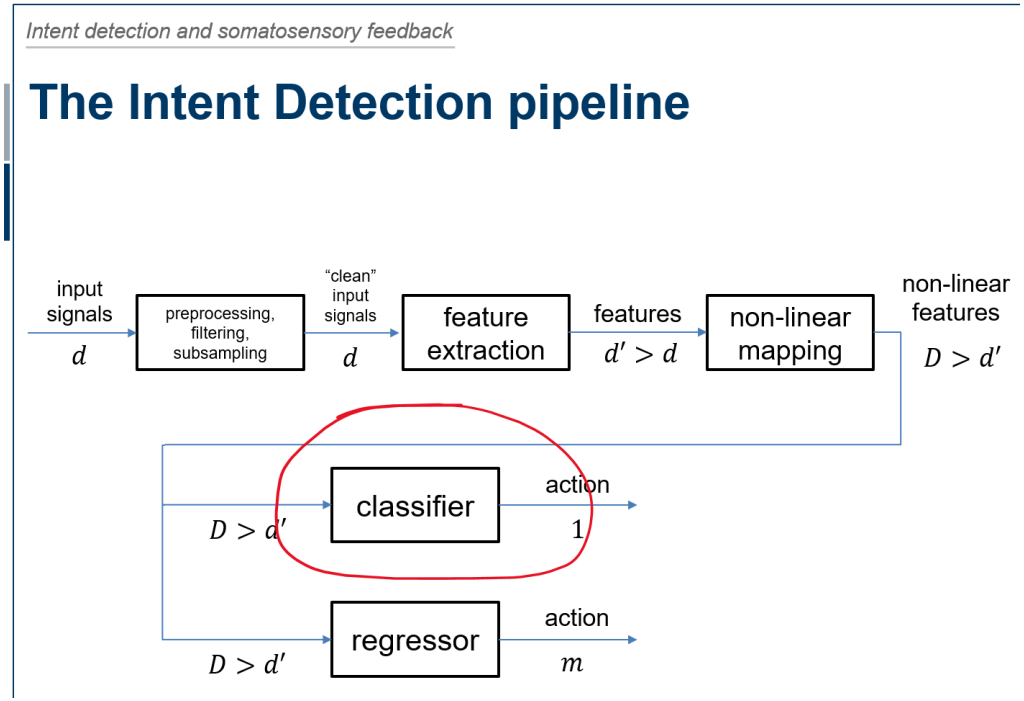
- regression
- again, on the I.D. pipeline:
  - de facto deconstructing the pipeline!
  - how to build a sensible (non-linear) mapping, ϕ…
  - …which in the end means to build the *ƒ tout court*

The Intent Detection pipeline

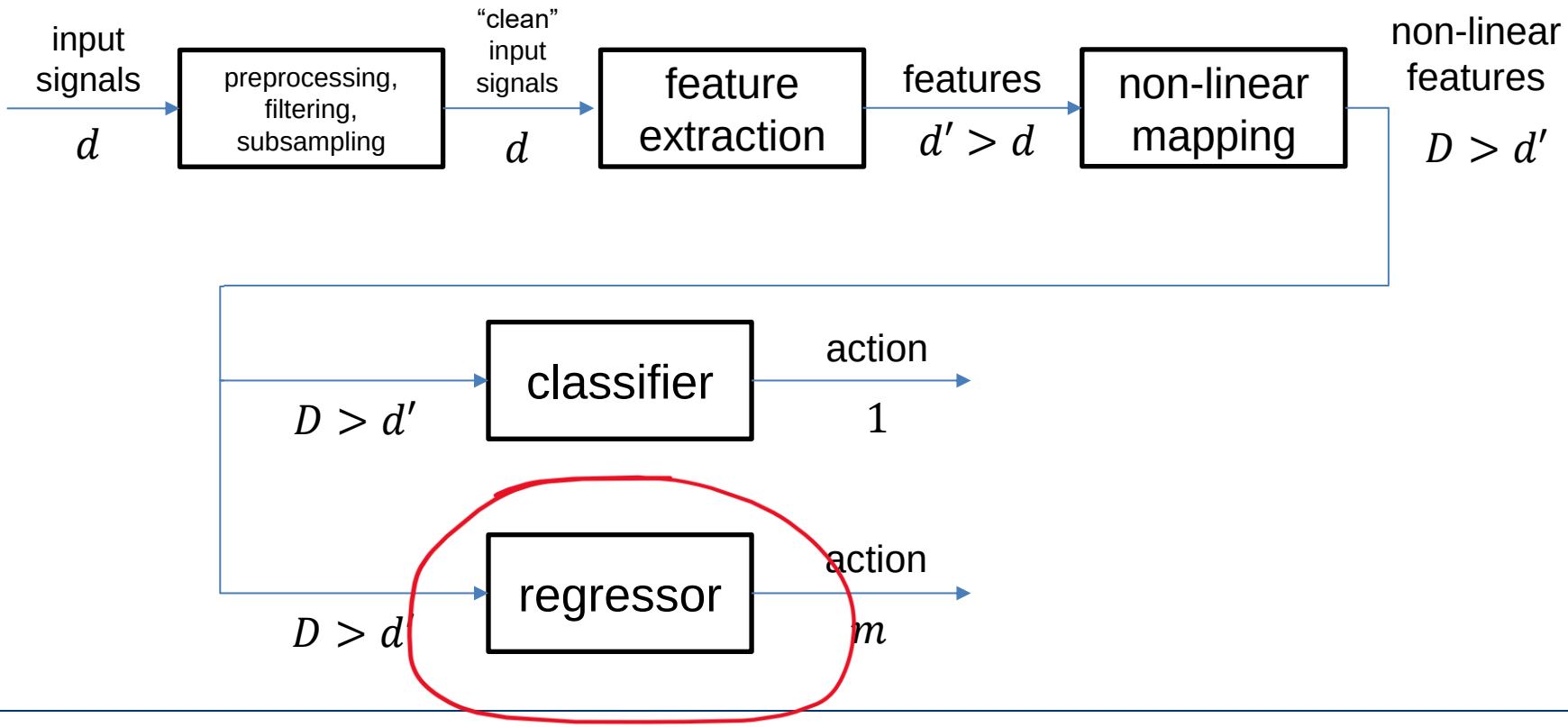
- recall lecture #08 and #09

Intent detection and somatosensory feedback

The Intent Detection pipeline



The Intent Detection pipeline



Regression

- mapping D-dimensional signals to an action.

$$y = f(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^D, y \in \mathbb{R}$$

- …and this is proper of all regression procedures.
- regression: a function mapping an input space to a real number
- alternative interpretation: there is none. This really is a function in $\mathbb{R}^D$:
  - think a curve if $D = 1$,
  - a surface if $D = 2$,
  - a bunch of coloured clouds if $D = 3$,
  - …

Regression

- all we said about classification carries on in the case of regression:
  - the need to properly choose your hypothesis space $\mathcal{H}$
  - the need to find a balance between under- and overfitting
  - minimising a cost functional $f^* = \arg\min_f [\mathcal{L}(f, X, \boldsymbol{y}) + \mathcal{R}(f)]$
- seems like a more „natural“, more apt solution than classification: why regression better?
  - naturally yields simultaneous control over many DoAs / motors
  - enables control on each single motor, proportionally
  - enables user control over an infinite manifold of activation configuration (instead of over a few predetermined actions, on-off)
- nevertheless, so far not very popular in the scientific community.

Ridge Regression

- recall lecture #09, again:
- here‘s a kind of regression which can be made *really* incremental!

ML, specifically for I.D.

- …but really, a (de-)incremental ML system is one which
  - given a new (observation,target_value) pair $(\boldsymbol{x}', y')$ (or a whole set of them)
  - can update its model without rebuilding it from scratch

$$f' \triangleq \mathcal{U}(f, \boldsymbol{x}', y')$$

- advantages:
  - no need to store any such pair!
  - potentially faster than the „simpler“ idea (storing and rebuilding)
- disadvantages:
  - how to „downdate“, i.e., how to forget past knowledge?
  - no chance to consider the quality of past data

Ridge Regression

- recall the linear classifier: let f be linear, $y = \boldsymbol{w}^T \boldsymbol{x}$ with $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^D$, where,
- given your dataset $S = (X, \boldsymbol{y})$,
- the optimal $\boldsymbol{w}$ is given by $\boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$
- and you predict your new class as $y = \text{sign}(\boldsymbol{w}^{*T} \boldsymbol{x})$

- this is also a regressor, you just remove the sign function from the predictor: $y = \boldsymbol{w}^{*T} \boldsymbol{x}$.

- this is called „regularised least-squares regression“ or „Ridge Regression“.
- Incremental learning involves updating the model parameters as new data points arrive, without retraining the model from scratch.

https://en.wikipedia.org/wiki/Sherman%E2%80%93Morrison_formula
https://en.wikipedia.org/wiki/Woodbury_matrix_identity

Ridge Regression

- The slide discusses making ridge regression "truly incremental". This means updating the model parameters as new data comes in, without needing to retrain from scratch.

- it turns out that RR (both as a classifier and as a regressor) can be made „truly“ incremental
  - by using a „rank-1 update“ of the Moore-Penrose pseudo-inverse
  - contained in the expression of the optimal $\boldsymbol{w}$, $\boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$

- let $\boldsymbol{w} = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y} \stackrel{\text{def}}{=} A\boldsymbol{b}$, then
- use the *Sherman-Morrison formula* to update A and $\boldsymbol{b}$
- and obtain the following algorithm:

  - start with $A = \boldsymbol{I_D}$ and $\boldsymbol{b} = \boldsymbol{0}$, then
  - for each new pair $(\boldsymbol{x}', y')$, let

    $$A' = A - \frac{A\boldsymbol{x}'\boldsymbol{x}'^T A}{1 + \boldsymbol{x}'^T A \boldsymbol{x}'} \quad \text{and} \quad \boldsymbol{b}' = \boldsymbol{b} + \boldsymbol{x}' y'$$

  - and let $\boldsymbol{w}' = A'\boldsymbol{b}'$ and $y = \boldsymbol{w}'^T \boldsymbol{x}$.

Ridge Regression

- is it advantageous? see Lecture #09: yes and no.
- the complexity of evaluating the Moore-Penrose pseudo-inverse from scratch („fake“ incremental) is

$$\mathcal{O}(D^3 + D^2 n)$$

- whereas the complexity of performing a Sherman-Morrison update („truly“ incremental) is

$$\mathcal{O}(D^2)$$

- *to be repeated for each new sample*. Given our n samples, the overall complexity is $\mathcal{O}(D^2 n)$…
- …as opposed to $\mathcal{O}(D^3 + D^2 n)$

Carles Igual, Luis A. Pardo Jr., Janne M. Hahne, and Jorge Igual. 2019. *Myoelectric Control for Upper Limb Prostheses* Electronics 8, 11.

Regression

- seems like a more „natural“, more apt solution than classification:
  - naturally yields simultaneous control over many DoAs / motors
  - enables control on each single motor, proportionally
  - enables user control over an infinite manifold of activation configuration (instead of over a few predetermined actions, on-off)
- nevertheless, so far not very popular in the scientific community.

3.2. Regression

Regression models do not classify input signals in a discrete set of classes, but approximate continuous multivariate outputs. Classifiers have the disadvantage that only a finite number of pre-trained patterns can be learned. Regressors do not have that handicap and give the user more freedom. A continuous mapping of the output allows a complete control and lots of combinations of values. The user can perform any movement generated by the controlled DoFs activation even if it has not been trained.

Classification has other limitations. The need of re-training for the non-stationary EMGs is one of them. As mentioned before, small changes in the EMG signals, e.g., fatigue, electrode shifting or sweat, are not well handled by classifiers. The user is more able to adapt to these effects in regression [91]. Small changes in the input signal can generate small variations in the prediction, which could lead to a missclassification. A small variation in the prediction on a regressor is better handled due to its continuity. There are no abrupt changes as class-boundaries, so the user can directly react and better compensate for potential errors in the estimation.

Carles Igual, Luis A. Pardo Jr., Janne M. Hahne, and Jorge Igual. 2019. *Myoelectric Control for Upper Limb Prostheses* Electronics 8, 11.

Regression

- seems like a more „natural“, more apt solution than classification.

In summary, regression models include the control for all DoFs simultaneously, independent and proportional, generating a smoother and more natural behavior of the prosthesis [35]. Therefore, a large amount of different motions could be used for prosthesis control. But for now, only two to three DoFs can be reliably controlled [14,46,47,92–98]. Regression allows the user to skip the separate and sequential control of different DoFs that classification proposes.

Hahne et al. [47] compared different linear and non linear regression techniques for two DoFs control. These techniques include linear regression (LR), mixture of linear experts (ME), multilayer-perceptron, and kernel ridge regression (KRR). Results have shown that KRR outperformed the other regressors. But with a basic linearization in the feature space, simpler regressors as ME or LR were able to perform as well as KRR, showing that simple linear models with the correct features are perfectly suitable for prosthesis control, increasing the efficiency of the model. It was also shown in the study how regressors were able to generalize for DoF combinations, where no training data was provided, proving their robustness against unknown situations for the model.

This more natural control was taken to a realistic manipulation scenario by Strazzulla et al. [99]. They were able to control two robotic arms that had ten independent DoFs between both, using a linear regressor for each DoF. With them they controlled the torque and force for each of the motors. The learning models were based on incremental ridge regression with random Fourier features. They achieved a completion rate of 95% of single-handed tasks and 84% of bimanual tasks.

Another very common regression approach is the support vector machine regression-based (rSVM). Ameri et al. [15] compared this regressor to a new regression convolutional network (rCNN). Regressors showed advantage over previous CNN classification studies facing independent simultaneous control of motions. Furthermore, the ability of the rCNN to extract underlying motor information in the EMG with no need for feature selection was presented as an advantage over rSVM and an option to solve robustness issues.

Janne M. Hahne, Meike A. Schweisfurth, Mario Koppe and Dario Farina, *Simultaneous control of multiple functions of bionic hand prostheses: Performance and robustness in end users,* Sci. Robotics 3, eaat3630

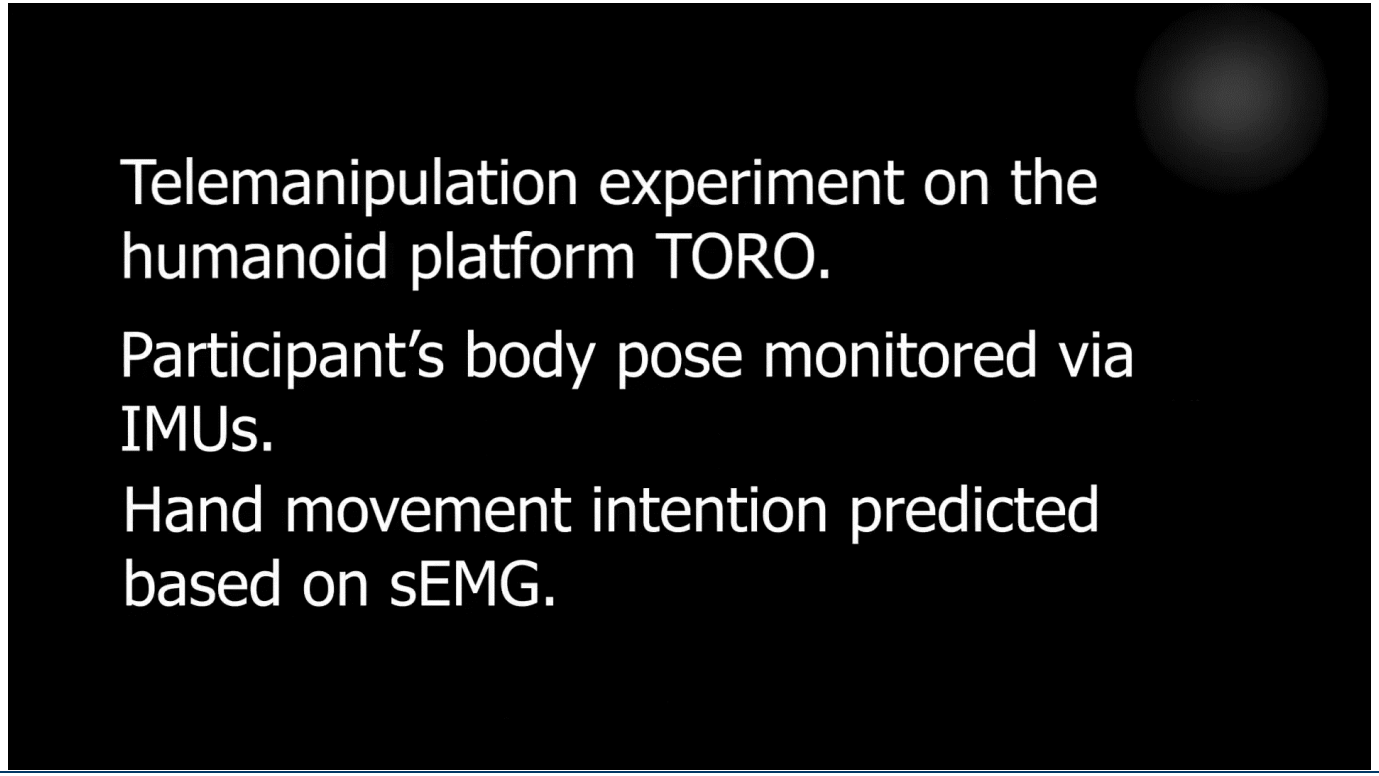
Regression

- seems like a more „natural“, more apt solution than classification. Co-contraction and regression(better results)

Connan, Mathilde / Sierotowicz, Marek / Henze, Bernd / Porges, Oliver / Albu-Schäffer, Alin / Roa Garzon, Máximo Alejandro / Castellini, Claudio, Learning to teleoperate an upper-limb assistive humanoid robot for bimanual daily-living tasks, 2021

Regression

- seems like a more „natural“, more apt solution than classification;
- possibly, it better fosters embodiment („more natural“ control)



The Intent Detection pipeline

- now let us try and bring together some of the things we saw in the past lectures.
- think of the „intent detection pipeline“: how fuzzy are the boundaries between „preprocessing / filtering“, „feature extraction“ and „non-linear mapping“?
- the answer is: they are totally undefined.
  - actually, the whole pipeline till the classifier / regressor could be called ϕ !
  - …and, what is this ϕ then?

The Intent Detection pipeline

- the concept is overtly simple:
  - you start from signals (data)
  - then you somehow transform those data so that they „better represent“ your outcomes (labels, target values)
  - and lastly you feed the transformed data to a classifier or a regressor
- „better represent“: they encode *salient characteristics* of the outcomes.
- how do you build a proper ϕ then? you use your imagination.
- or… is there an alternative way?
- fed raw input signals - > ϕ -> Extract right features from the signals

The Intent Detection pipeline

- well, there is at least three of them – three ways to *try and mechanise the generation of* ϕ
  - in other words: to *automatically* extract the „right“ features from your raw signals

1. use a deep-learning approach
2. use the kernel trick
3. use a finite approximation of the kernel trick

The Intent Detection pipeline

- well, there is at least three of them – three ways to *try and mechanise the generation of* ϕ
  - in other words: to *automatically* extract the „right“ features from your raw signals

1. use a deep-learning approach
2. use the kernel trick
3. use a finite approximation of the kernel trick

Deep learning

- figure out a sensible neural network structure
  - input is $N \times M$ images? then let the input be exactly an $N \times M$ layer
  - further layers adapt to the „desired number of features“
- create such a huge blank system
  - with hundreds of thousands of parameters to be tuned
- feed it as many labelled samples as you can
  - use standard techniques (e.g., backprop) to build a model:
  - either „learn“ salient features, or
  - discriminate among them

- in intent detection, this approach is not (yet) very popular, mainly since
  - there usually is not that many data available
  - you anyway want to gather as few as possible of them! (avoid stressing the user)
  - there are publicly available databases, but they are of scarce usefulness (users are too diverse)
  - there usually are few sensors, i.e., sEMG data mostly are not images

Deep learning

- example: large meta-analysis

Existing EMG pattern recognition approaches can be broadly divided into two categories: (1) methods based on *feature engineering* and (2) methods based on *feature learning*. "Feature Engineering" and feature extraction have been key parts of conventional machine learning algorithms. In EMG analysis, short time windows of the raw EMG signal are extracted and augmented by extracting time and frequency features aimed at improving information quality and density. Many studies have shown that the quality and quantity of the hand-crafted features have great influence on the performance of EMG pattern recognition [18–21].

Conversely, in "feature learning", explicit transformation of the raw EMG signals is not required as features are automatically created by the machine learning algorithms as they learn. The use of "deep learning" therefore shifts the focus from manual (human-based) feature engineering to automated feature engineering or learning. Although neural networks have been used in EMG research for several decades, deep learning techniques have recently been applied to EMG pattern recognition. This is, at least in part, due to the lack of sufficient EMG data availability to train these deep neural networks in the earlier years of the field. With the advent of shared bigger EMG data sets and recent advances in techniques for addressing overfitting problems, most emerging deep learning architectures and methods have now been employed in EMG pattern recognition systems (e.g., [14,23,24]). In some cases, both feature engineering and learning are combined by inputing pre-processed data or pre-extracted features to a deep learning algorithm with some benefits having been shown (e.g., references [11,23,24]).

Phinyomark, A.; Scheme, E.
EMG Pattern Recognition in the Era of Big Data and Deep Learning.
Big Data Cogn. Comput. **2018**, *2*, 21.

Deep learning

- example: large meta-analysis

Table 5. *Cont.*

Ref.	Deep Leaning Model	Deep Learning Software	Input Data (Window Size/Overlap)	Application	Test Conditions	Dataset (Number of Subjects)	Results
[11]	CNN	Theano [e]	Time-frequency features (260 ms/25 ms)	Motion recognition	Inter-subject, Between-day	Côté-Allard et al. (36), Ninapro 5	Côté-Allard et al.: 98.31% (for 7 motion classes), Ninapro 5: 65.57% (for 18 motion classes)
[113]	CNN	MatConvNet [c]	Time-frequency features (200 ms/100 ms)	Motion recognition	Amputation	Ninapro 2,3	CNN > SVM
[114]	CNN	MatConvNet [c]	Raw EMG (150 ms)	Motion recognition	Amputation	Ninapro 1,2,3	Ninapro 1,2: RF > CNN Ninapro 3: SVM > CNN
[118]	CNN	n/a	Raw EMG (200 ms)	Motion recognition	Inter-subject	Ninapro 1	CNN > SVM
[102]	CNN	Keras [f] + TensorFlow [g]	Raw EMG (150 ms/5 ms)	Motion recognition	Compact architecture	Local data set (10) 8 + 5 EMG channels	CNN > SVM
[23]	RNN + CNN	n/a	Time-frequency features (50 ms/30 ms)	Joint angle estimation	Regression	Local data set (8) 5 EMG channels	RNN + CNN > CNN, SVR
[115]	RNN	CNTK [h]	Time domain features (200 ms/150 ms)	Motion recognition	Amputation	Ninapro 7	RNN > RNN + CNN > CNN

Note that UPN, unsupervised pre-trained networks; DBN, deep belief network; SM-DBN, split-and-merge deep belief network; SAE, stacked auto-encoder; CNN, convolutional neural network; RNN, recurrent neural network; SVM, support vector machine; SVR, support vector regression; LDA, linear discriminant analysis; GMM, Gaussian mixture model; KNN, k-nearest neighbors; MLP, multi-layer perceptron neural network; RF, random forests; DWT, discrete wavelet transform; CS, compressive sensing; PCA, principal component analysis. [a] http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html; [b] https://github.com/rasmusbergpalm/DeepLearnToolbox; [c] http://www.vlfeat.org/matconvnet/; [d] https://mxnet.apache.org/; [e] http://deeplearning.net/software/theano/; [f] https://keras.io/; [g] https://www.tensorflow.org/; [h] https://www.microsoft.com/en-us/cognitive-toolkit/.

Phinyomark, A.; Scheme, E.
EMG Pattern Recognition in the Era of Big Data and Deep Learning.
Big Data Cogn. Comput. **2018**, *2*, 21.

Deep learning

- example: comparing and combining „learned“ and „handcrafted“ features- ADANN is an advanced algorithm designed to improve the generalization of deep learning models for EMG-based gesture recognition by employing domain adaptation and adversarial learning techniques. This allows the model to perform better across different individuals, overcoming the variability in EMG signals.

Existing research on myoelectric control systems primarily focuses on extracting discriminative characteristics of the electromyographic (EMG) signal by designing handcrafted features. Recently, however, deep learning techniques have been applied to the challenging task of EMG-based gesture recognition. The adoption of these techniques slowly shifts the focus from feature engineering to feature learning. Nevertheless, the black-box nature of deep learning makes it hard to understand the type of information learned by the network and how it relates to handcrafted features. Additionally, due to the high variability in EMG recordings between participants, deep features tend to generalize poorly across subjects using standard training methods. Consequently, this work introduces a new multi-domain learning algorithm, named ADANN (Adaptive Domain Adversarial Neural Network), which significantly enhances ($p = 0.00004$) inter-subject classification accuracy by an average of 19.40% compared to standard training. Using ADANN-generated features, this work provides the first topological data analysis of EMG-based gesture recognition for the characterization of the information encoded within a deep network, using handcrafted features as landmarks. This analysis reveals that handcrafted features and the learned features (in the earlier layers) both try to discriminate between all gestures, but do not encode the same information to do so. In the later layers, the learned features are inclined to instead adopt

Deep learning

- example: comparing and combining „learned“ and „handcrafted“ features

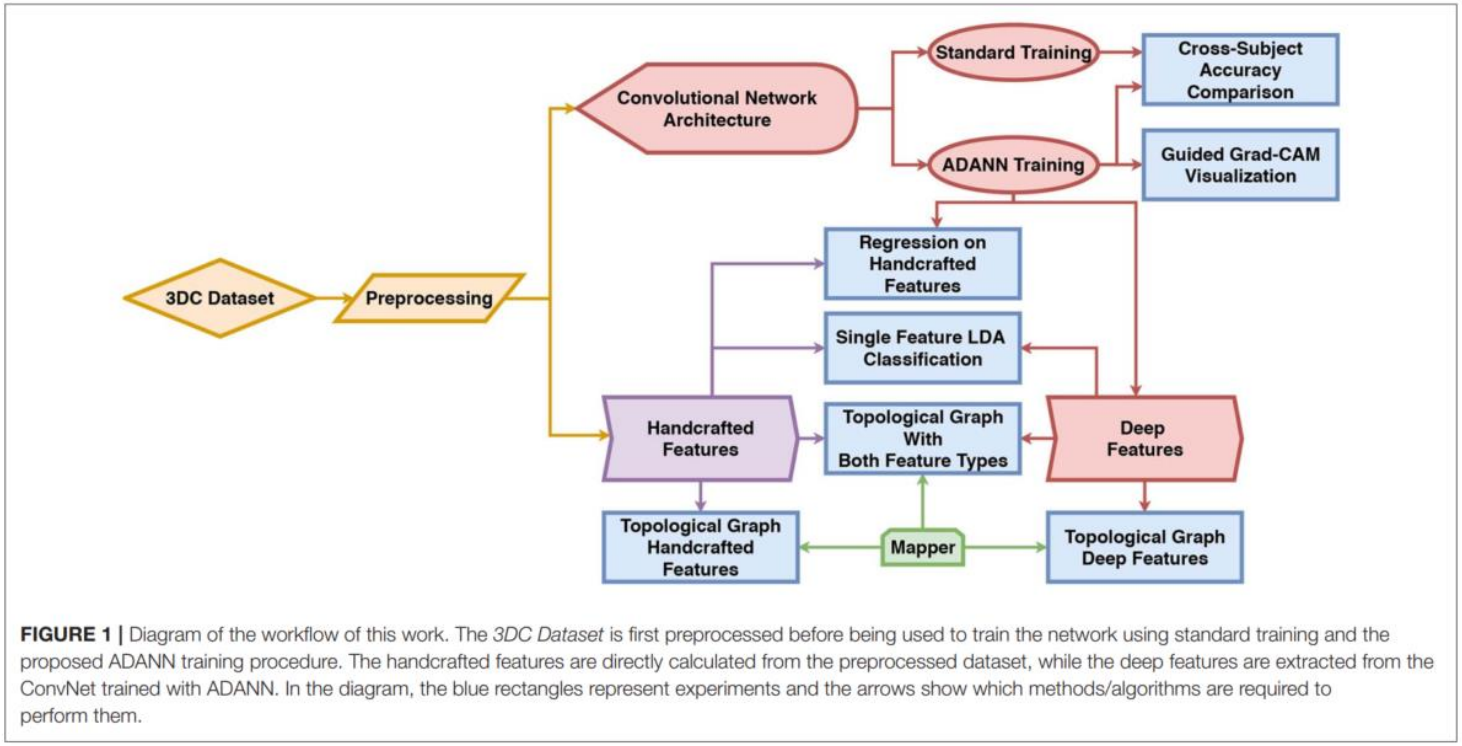


FIGURE 1 | Diagram of the workflow of this work. The *3DC Dataset* is first preprocessed before being used to train the network using standard training and the proposed ADANN training procedure. The handcrafted features are directly calculated from the preprocessed dataset, while the deep features are extracted from the ConvNet trained with ADANN. In the diagram, the blue rectangles represent experiments and the arrows show which methods/algorithms are required to perform them.

Côté-Allard U, Campbell E, Phinyomark A, Laviolette F, Gosselin B and Scheme E (2020) Interpreting Deep Learning Features for Myoelectric Control: A Comparison With Handcrafted Features. *Front. Bioeng. Biotechnol.* 8:158.

Deep learning

- example: comparing and combining „learned“ and „handcrafted“ features

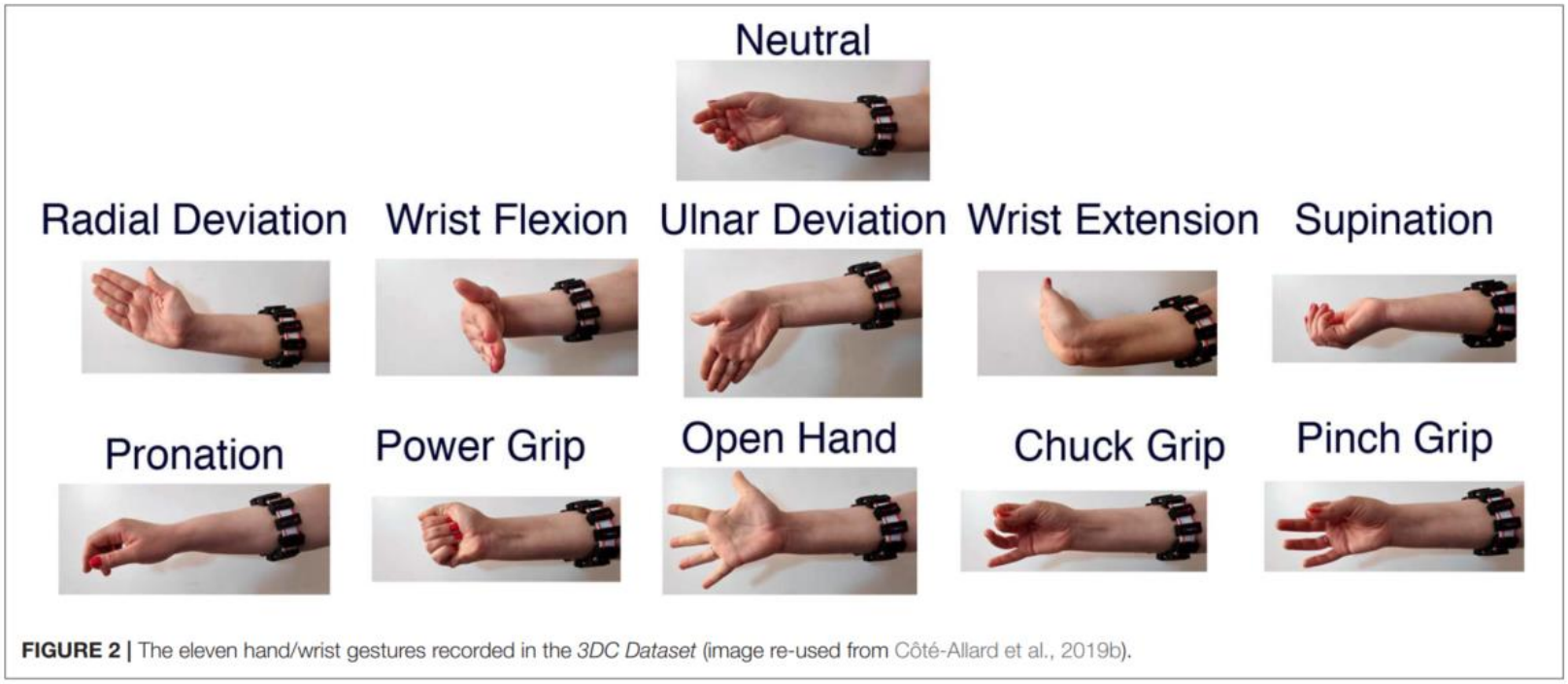


FIGURE 2 | The eleven hand/wrist gestures recorded in the *3DC Dataset* (image re-used from Côté-Allard et al., 2019b).

Côté-Allard U, Campbell E, Phinyomark A, Laviolette F, Gosselin B and Scheme E (2020)
Interpreting Deep Learning Features for Myoelectric Control: A Comparison With Handcrafted Features.
Front. Bioeng. Biotechnol. 8:158.

Deep learning

- example: comparing and combining „learned“ and „handcrafted“ features

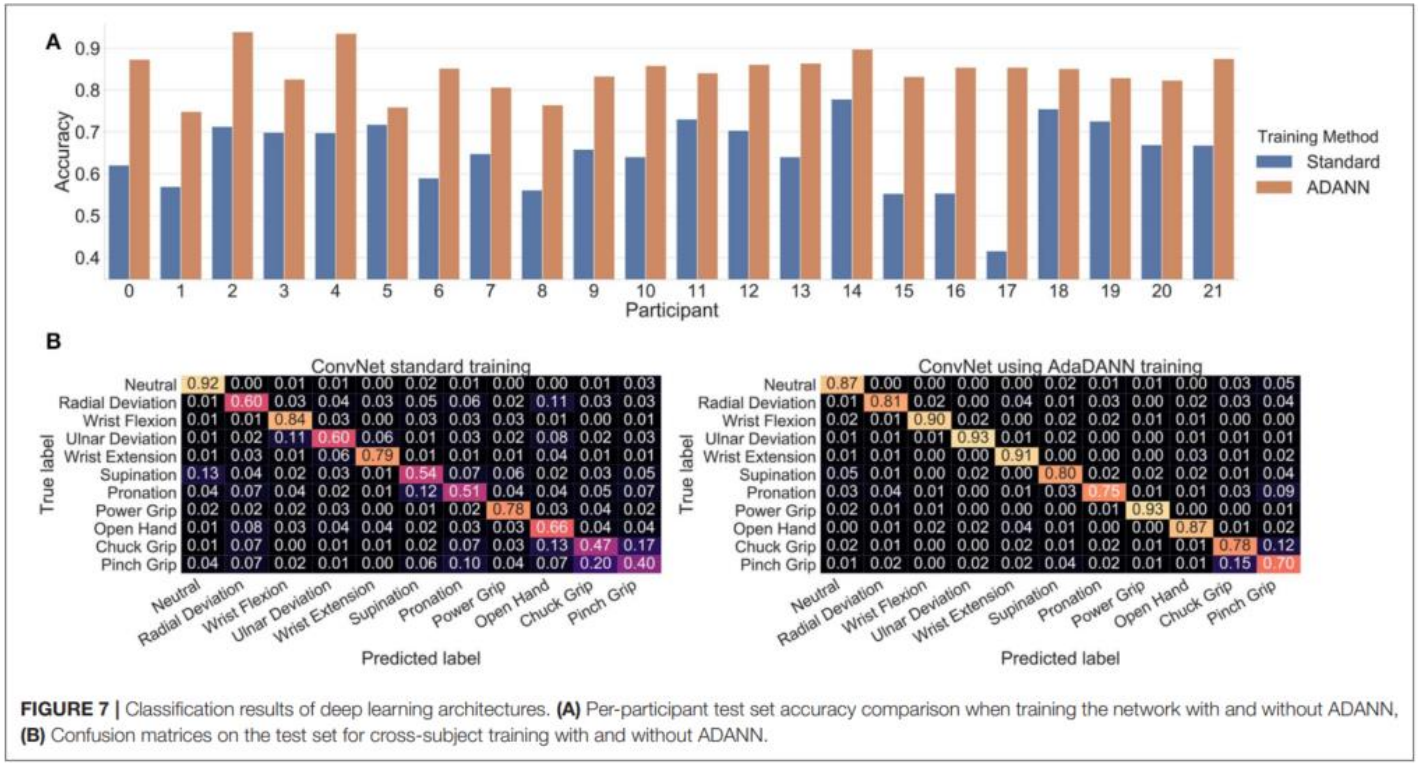


FIGURE 7 | Classification results of deep learning architectures. **(A)** Per-participant test set accuracy comparison when training the network with and without ADANN, **(B)** Confusion matrices on the test set for cross-subject training with and without ADANN.

Côté-Allard U, Campbell E, Phinyomark A, Laviolette F, Gosselin B and Scheme E (2020) Interpreting Deep Learning Features for Myoelectric Control: A Comparison With Handcrafted Features. *Front. Bioeng. Biotechnol.* 8:158.

The Intent Detection pipeline

- well, there is at least three of them – three ways to *try and mechanise the generation of* ϕ
  - in other words: to *automatically* extract the „right“ features from your raw signals

1. use a deep-learning approach
2. use the kernel trick
3. use a finite approximation of the kernel trick

Using the kernel trick

- instead of „keeping it simple“ (possibly *too simple*…), that is linear, $y = \boldsymbol{w}^T \boldsymbol{x}$
  - or $y = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$ in the case of classification

- many ML approaches try and find their approximant by using *kernel functions*,

$$y = f(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \, K(\boldsymbol{x}_i, \boldsymbol{x})$$

- where K is a real-valued binary function of your input space. in the ideal case,
  - $K(\boldsymbol{a}, \boldsymbol{b}) = 1$ iff $\boldsymbol{a} = \boldsymbol{b}$
  - $K(\boldsymbol{a}, \boldsymbol{b}) \to 0$ iff $\|\boldsymbol{a} - \boldsymbol{b}\| \to \infty$

- that is, K is an affinity / similarity function,
- and the approximant is a weighted sum of „how similar $\boldsymbol{x}$ is to the $\boldsymbol{x}$s I already know“.

Ntampaka, M., Trac, H., Sutherland, D., Battaglia, N., Póczos, B. and Schneider, J., *A Machine Learning Approach For Dynamical Mass Measurements Of Galaxy Clusters*, The Astrophysical Journal, 2015.

Using the kernel trick

- there are conditions on K in order to be a „proper“ kernel function (Mercer‘s conditions),
- but its main property is that it implicitly defines a function ϕ

$$K(\boldsymbol{a}, \boldsymbol{b}) = \phi(\boldsymbol{a})^T \phi(\boldsymbol{b})$$

- mapping observations onto a D-dimensional „feature space“!

- simple example: the quadratic kernel
- $K(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}^T \boldsymbol{b} + c)^2$

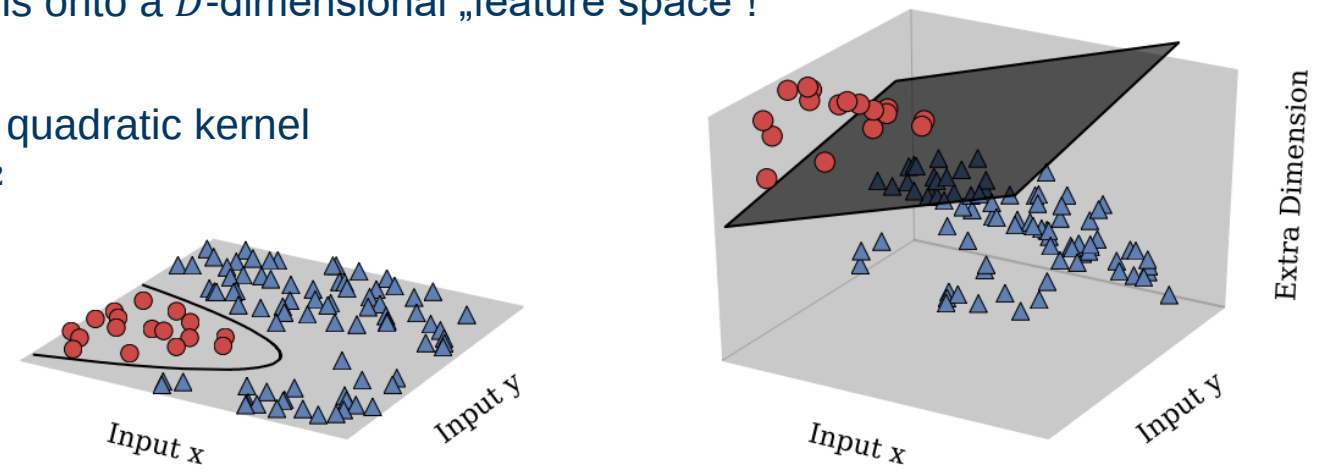


FIG. 5.— In their basic form, Support Vector Machine classifiers search for a hyperplane that separates two classes, here shown as red circles and blue triangles. In the two-feature input space (left panel), the two classes are not separable by a hyperplane. But if the inputs are re-cast into an appropriate higher-dimensional space (right panel), the red circles and blue triangles can be divided by a plane. Test examples can now be classified based on the side of the plane on which they lie; a test example lying above the plane would be classified as a red circle, and one below as a blue triangle. The “decision boundary” which divides the predicted classes (black plane, right panel) corresponds to a different shape in the original space (solid black curve, left panel). We can find the boundary in the original space without explicitly mapping to the higher-dimensional space using the “kernel trick.” Although we wish to predict a continuous-valued label rather than a binary classification, an analogous process applies.

Using the kernel trick

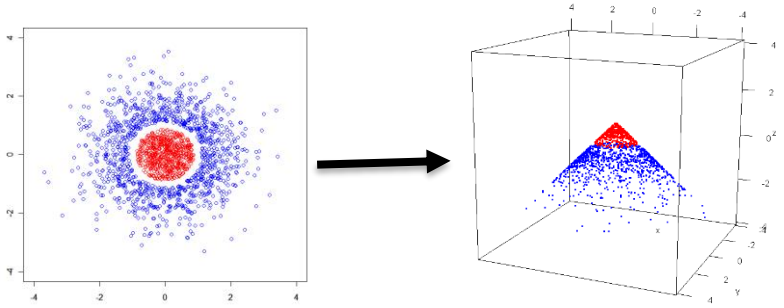
- there are conditions on K in order to be a „proper“ kernel function (Mercer‘s conditions),
- but its main property is that it implicitly defines a function ϕ

$$K(\boldsymbol{a}, \boldsymbol{b}) = \phi(\boldsymbol{a})^T \phi(\boldsymbol{b})$$

- mapping observations onto a D-dimensional „feature space“!

- simple example: our „cone-mapping“ $\phi(\boldsymbol{a}) = (a_1, a_2, -\|\boldsymbol{a}\|)$ (Lecture 8) is implicitly defined by the kernel function

$$K(\boldsymbol{a}, \boldsymbol{b}) = [a_1 \; a_2 \; -\|\boldsymbol{a}\|] \cdot \begin{bmatrix} b_1 \\ b_2 \\ -\|\boldsymbol{b}\| \end{bmatrix} = a_1 b_1 + a_2 b_2 + \|\boldsymbol{a} \cdot \boldsymbol{b}\| = \boldsymbol{a} \cdot \boldsymbol{b} + \|\boldsymbol{a} \cdot \boldsymbol{b}\|$$



Using the kernel trick

- now what is better? to use $\phi(\cdot)$ directly, or to use the kernel $K(\cdot,\cdot)$?
- if we can write down ϕ then we can choose; but there is a case in which we cannot
- the case in which D is infinite – more properly, ϕ maps observations to functions.
- it is the case of the famous Radial-Basis-Function (RBF/Gaussian) kernel,

$$K(\boldsymbol{a},\boldsymbol{b}) \triangleq \exp\left(-\frac{\|\boldsymbol{a}-\boldsymbol{b}\|^2}{2\sigma^2}\right) = \phi(\boldsymbol{a})^T\phi(\boldsymbol{b})$$

- in this case, ϕ has an infinitely-long expression, i.e., unusable in practice
  - notice, however, that if ϕ only appears in dual inner-product form, i.e., $\phi(\cdot)^T\phi(\cdot)$,
  - we can just use $K(\cdot,\cdot)$ and be happy not knowing ϕ
  - this is why it‘s called *the kernel trick:* we really use ϕ, but *we cannot write it down!*

Using the kernel trick

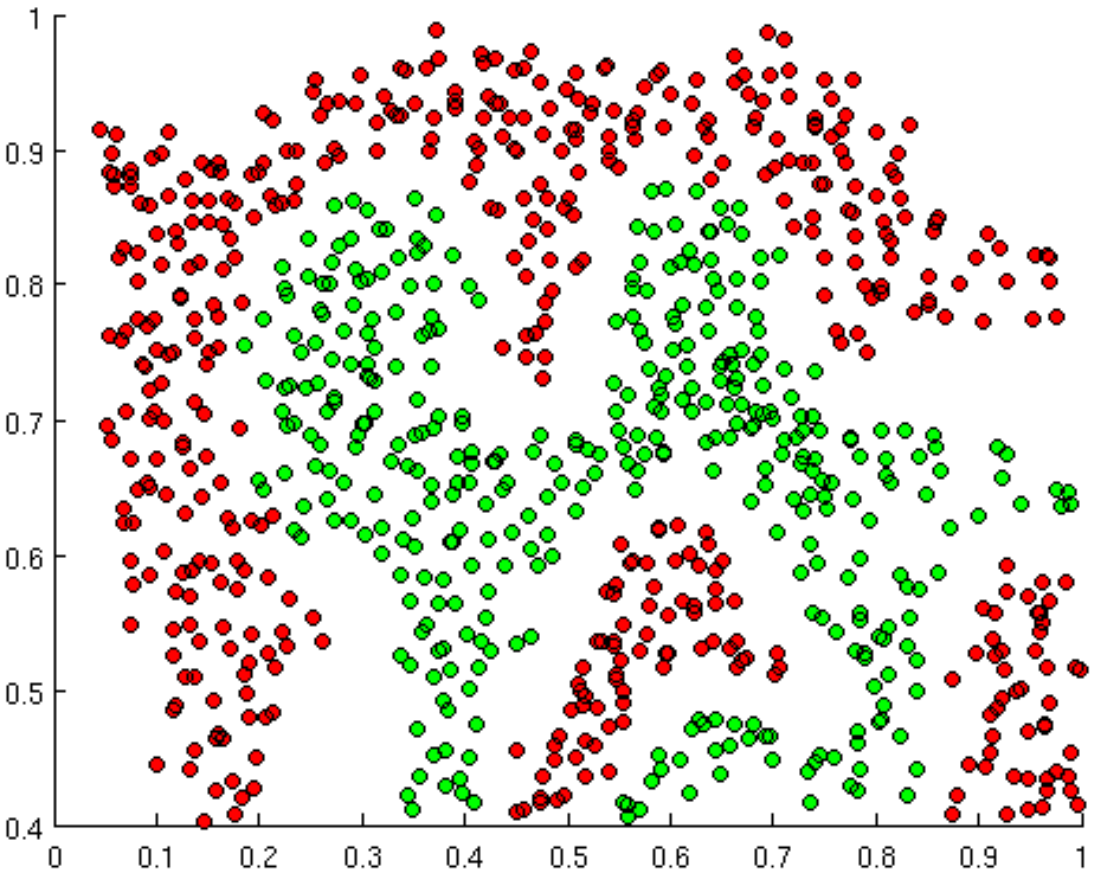
- now why would we ever be interested in using the Gaussian kernel?
- because it is a *universal approximator*, given a $\sigma > 0$:

$$y = f(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \, K(\boldsymbol{x}_i, \boldsymbol{x}) \triangleq \sum_{i=1}^{n} w_i \exp(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}\|^2}{2\sigma^2})$$

- that, is, a weighted sum of Gaussians,
  - each one centered around one known observation,
  - and gifted with arbitrarily small variance σ
  - the smaller the σ, the higher the „precision“ (and the danger to overfit…!)
- that means, it more or less is the most powerful weapon at our disposal
  - no matter how complicated the problem is, it can always solve it
- compare with, e.g., linear and polynomial kernels:
  - if your data is not linearly (polynomially) separable, these kernels won‘t work
  - „there is only such-and-such amount of complexity those kernels can bear“

Using the kernel trick

- example: a very involved, but still clearly (non-linearly) separable problem



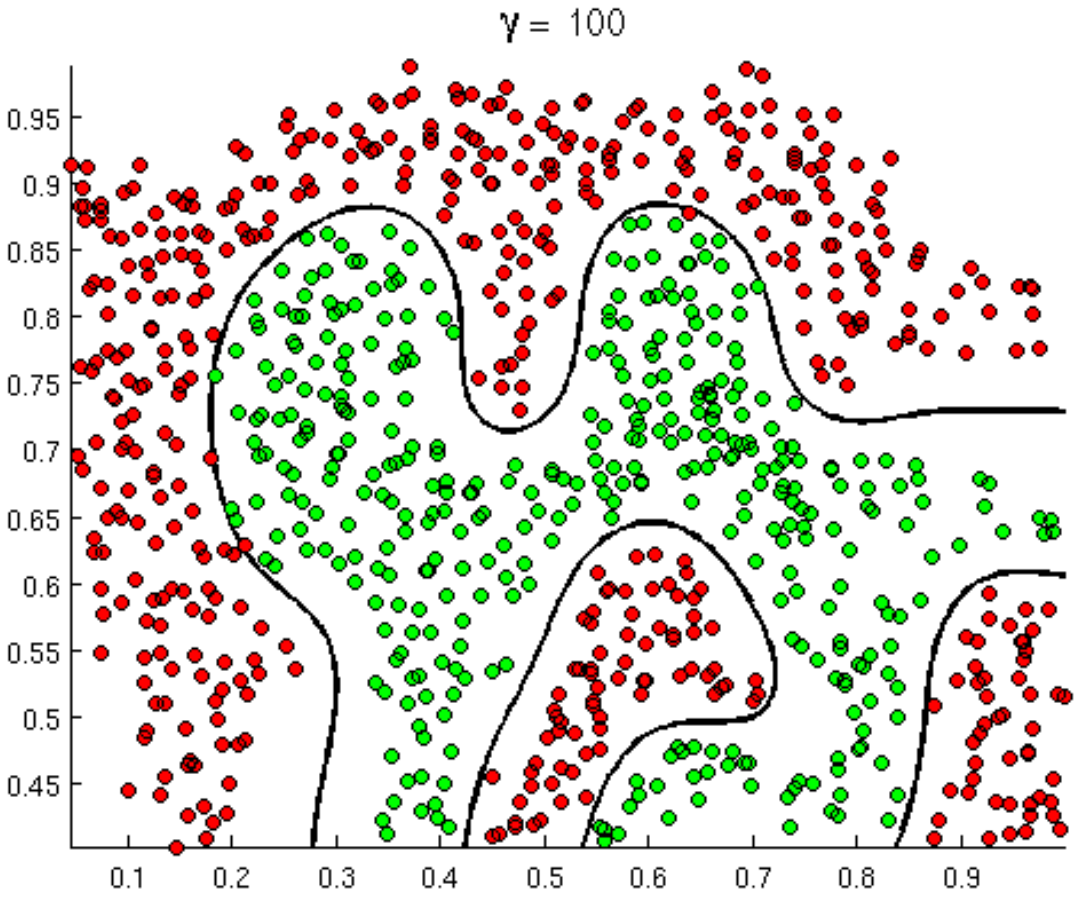
Andrew Ng, http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html

Using the kernel trick

- example: a very involved, but still clearly (non-linearly) separable problem

(notation: sometimes $\gamma \triangleq -\frac{1}{2\sigma^2}$,

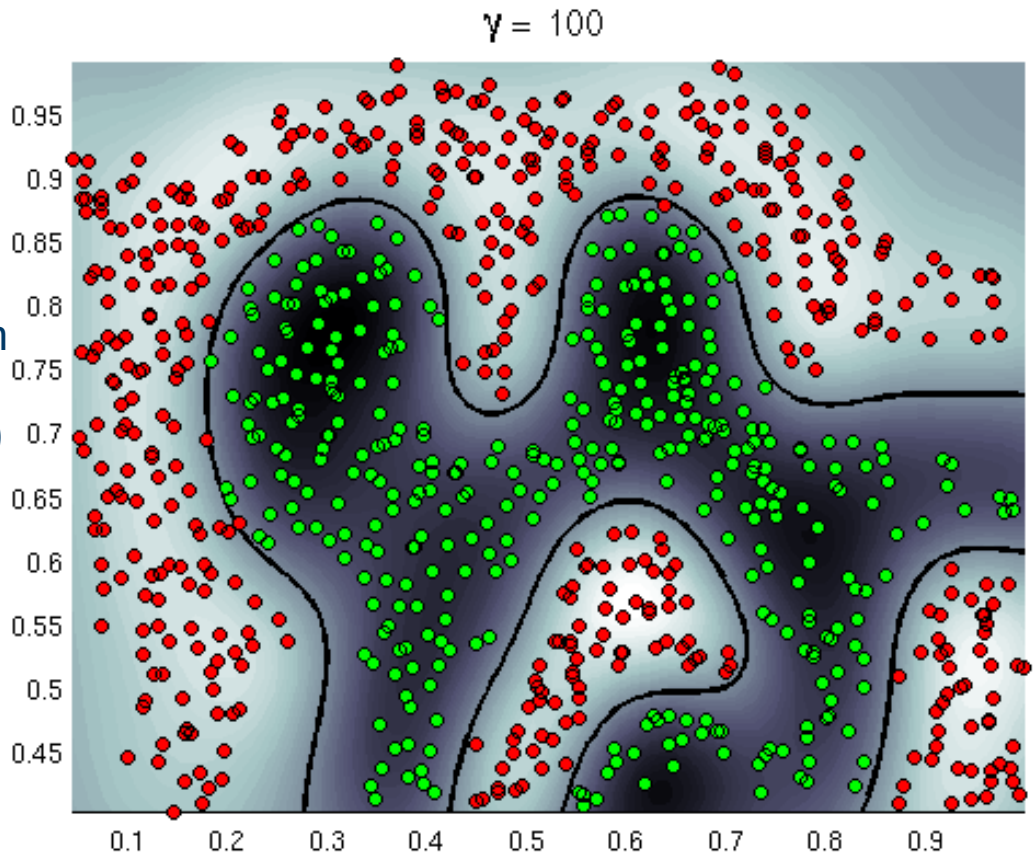
so that $y = \sum_{i=1}^{n} w_i \exp(-\gamma \|\boldsymbol{x_i} - \boldsymbol{x}\|^2)$)



Andrew Ng, http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html

Using the kernel trick

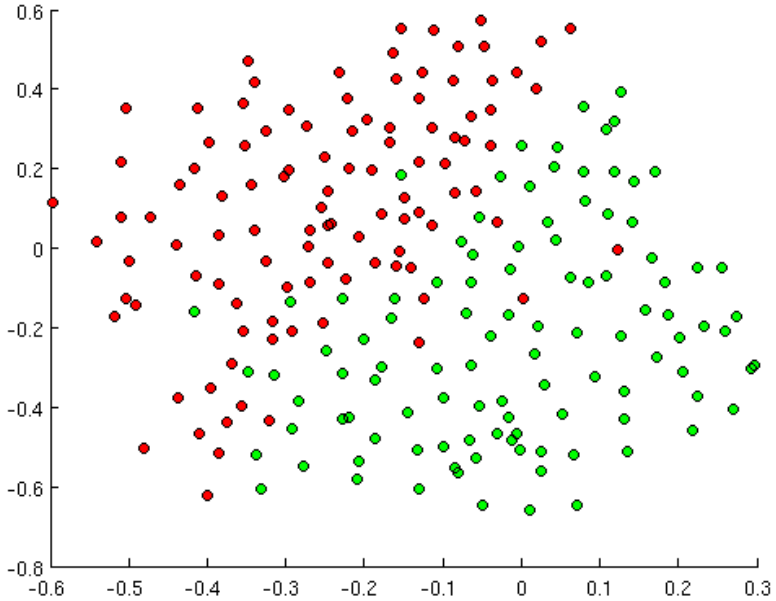
- example: a very involved, but still clearly (non-linearly) separable problem
- this crooked separating hyper-surface (a.k.a. manifold, curve) is exactly the intersection of some even-more-crooked 3D function with the plane $z = 0$
- and it was found automatically (i.e., via minimisation of a cost functional) using a SVM.



Andrew Ng, http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html

Using the kernel trick

- a definitely less lucky example:

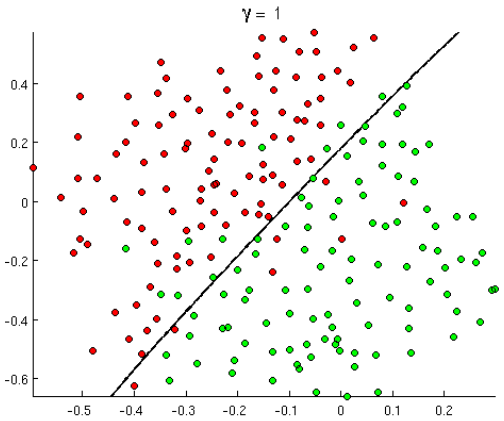


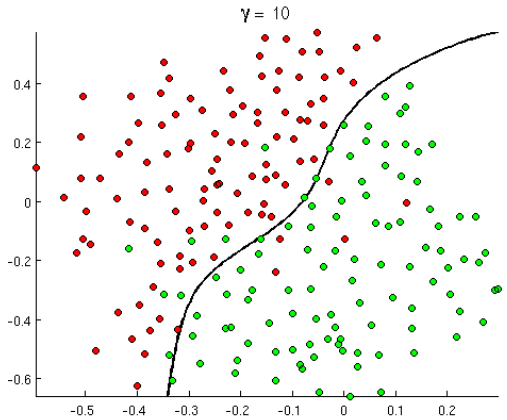
Now train your model using γ values of 1, 10, 100, and 1000 and plot the decision boundary (using no contour fill) for each model. How does the fit of the boundary change with γ?

Andrew Ng, http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html

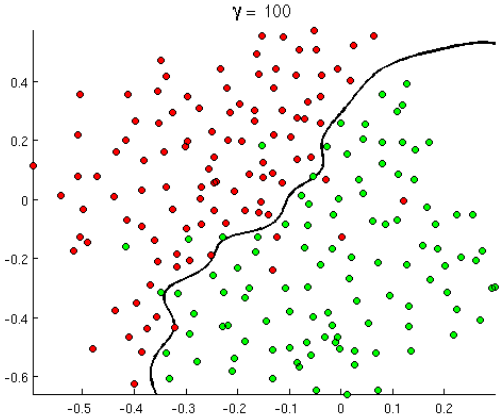
Using the kernel trick

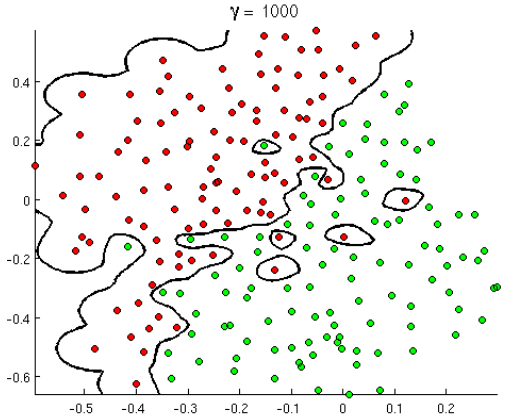
- a definitely less lucky example:





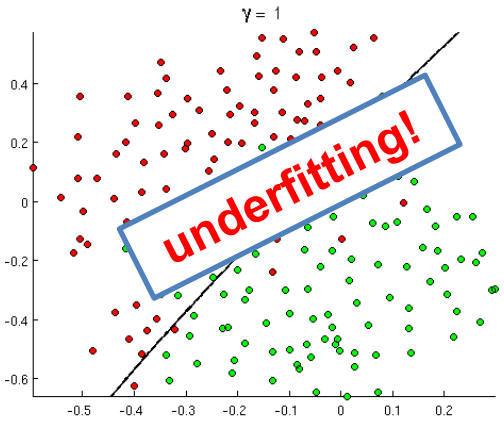
- all four solutions are produced using the same algorithm: SVM Classification with RBF kernel.

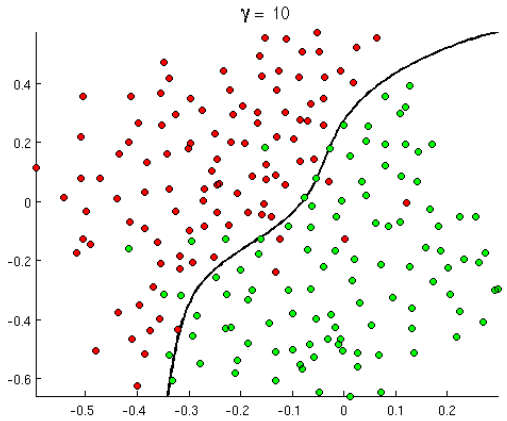




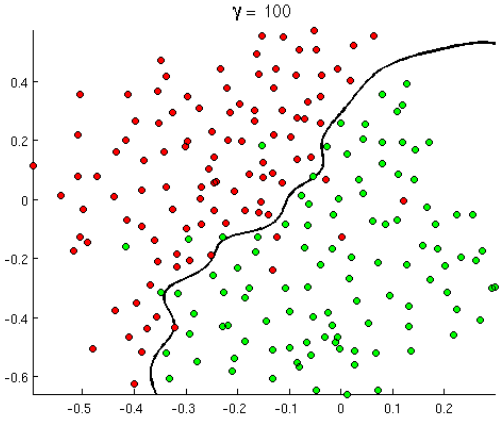
Using the kernel trick

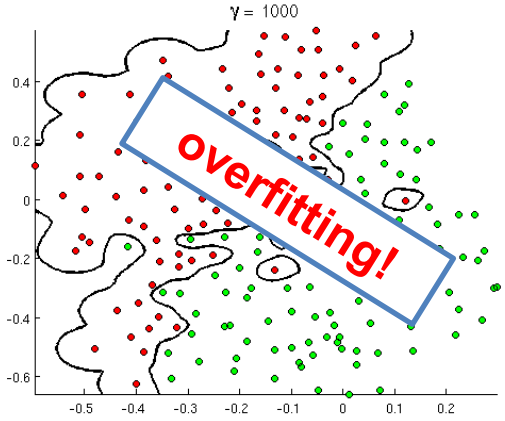
- a definitely less lucky example:





- all four solutions are produced using the same algorithm: SVM Classification with RBF kernel.
- some of them are good, some of them are bad.





Using the kernel trick

- so the kernel trick is another way of automatically finding our ϕ
- advantages:
  - universal – can solve any possible (classification/regression) problem in principle
  - models are really good if overfitting avoided (do cross-validation and grid-search on σ)
- disadvantages:
  - long model-building (training) time – usually cubic in n !
  - non-incremental – still need to rebuild the model from scratch when new knowledge is available
  - need optimisation – computationally hard if cost functional convex (SVM), otherwise prone to local minima
- can we then find something even better than this?
- yes, we can.

The Intent Detection pipeline

- well, there is at least three of them – three ways to *try and mechanise the generation of* ϕ
  - in other words: to *automatically* extract the „right“ features from your raw signals

1. use a deep-learning approach
2. use the kernel trick
3. use a finite approximation of the kernel trick

Random Fourier Features

- suppose that, instead of taking the full Gaussian kernel $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{2\sigma^2})$,
- we *approximate it* using a finite number D of its Fourier coefficients.
- in that case, what does ϕ look like? is it feasible to use it?
  and what coefficients do we really need?
- according to Bochner‘s theorem (1933):

$$K(\boldsymbol{x}_1 - \boldsymbol{x}_2) = \int_{\mathbb{R}^D} e^{-i\boldsymbol{\omega}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)} p(\boldsymbol{\omega}) d\boldsymbol{\omega}$$

- here, $e^{-i\boldsymbol{\omega}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)}$ is precisely an approximation of $K(\boldsymbol{x}_1 - \boldsymbol{x}_2)$, so it‘s the kernel we‘re after.

Random Fourier Features

- suppose that, instead of taking the full Gaussian kernel $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}{2\sigma^2})$,
- we *approximate it* using a finite number D of its Fourier coefficients.
- in that case, what does ϕ look like? is it feasible to use it?
  and what coefficients do we really need?
- according to Bochner's theorem (1933):

$$K(\boldsymbol{x}_1 - \boldsymbol{x}_2) = \int_{\mathbb{R}^d} e^{-i\boldsymbol{\omega}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)} p(\boldsymbol{\omega}) d\boldsymbol{\omega}$$

- and $e^{-i\boldsymbol{\omega}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)}$ can be rewritten as

$$e^{-i\boldsymbol{\omega}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)} = \cos(\boldsymbol{\omega}^T\boldsymbol{x}_1 + b) \cdot \cos(\boldsymbol{\omega}^T\boldsymbol{x}_2 + b) \triangleq \phi(\boldsymbol{x}_1)^T\phi(\boldsymbol{x}_2)$$

- which tells us exactly how to build ϕ !

Random Fourier Features

- so we can explicitly build a non-linear mapping $\phi(\boldsymbol{x}) = \cos(\boldsymbol{\omega}^T \boldsymbol{x} + b)$, where
  - the $\boldsymbol{\omega}$s are randomly drawn from a Gaussian distribution $\mathcal{N}(\mathbf{0}, \sigma)$ and
  - the bs are randomly drawn from a uniform distribution $\mathcal{U}(-\pi, \pi)$.

- and this is for one Fourier coefficient $\boldsymbol{\omega}$. We need to use D of them, leading to

$$\phi(\boldsymbol{x}) = \begin{bmatrix} \cos(\boldsymbol{\omega}_1^T \boldsymbol{x} + b_1) \\ \vdots \\ \cos(\boldsymbol{\omega}_D^T \boldsymbol{x} + b_D) \end{bmatrix}$$

- this ϕ
  - is a finite-(D-)dimensional approximation of the Gaussian kernel,
  - it is easy to build, computationally fast,
  - and it can be plugged into Ridge Regression like any other ϕ:

obtain $\boldsymbol{w}^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \boldsymbol{y}$

use $y = \boldsymbol{w}^{*T} \phi(\boldsymbol{x})$

Random Fourier Features

- the Matlab code implementing RFFs is ridiculously simple.

```
% computed once and for all
D = 25; % choose how many Fourier features
global omega b;
omega = randn(D,d); % omega ~ N(0,1) (Dxd)
b = 2*pi*rand(D,1)-pi; % b ~ U(-pi,pi) (Dx1)
```

```
function phi_X = phi(X)

global omega b;
n = size(X,1);
phi_X = cos(omega*X' + b*ones(1,n))';
```

Random Fourier Features

- the Matlab code implementing RFFs is ridiculously simple.

```
% -------------------------------- ridge regression
lambda = 1;
w = inv( X'*X + lambda*eye(d) ) * X' * Y;
Y_hat = w'*X';
rMSE = sqrt(mean( (Y-Y_hat').^2 )) % rMSE
```

```
% -------------------------------- ridge regression with R
% just substitute phi(X) for X
lambda = 1;
w = inv( phi(X)'*phi(X) + lambda*eye(D) ) * phi(X)' * Y;
Y_hat = w'*phi(X)';
rMSE = sqrt(mean( (Y-Y_hat').^2 ))
```

Meeting your ideal

- recall Lecture #09:

ML, specifically for I.D.

- so, which one is better? it depends on
  - speed of rebuilding the model
  - need to store past pairs vs. memory requirements
  - …
- in general, our ideal ML method for I.D.
  - is **incremental** / decremental (how? see above)
  - is numerically stable and **fast** to be rebuilt / updated
  - can tackle **non-linear** problems
  - yields a **measure of confidence** in its own prediction
  - can be either a **classifier** or a **regressor**

Summary

- today:
- regression – better or worse than classification?
- a method meeting our ideal requirements

References

- Phinyomark, A.; Scheme, E., *EMG Pattern Recognition in the Era of Big Data and Deep Learning.* Big Data Cogn. Comput. 2018, 2, 21.
- Côté-Allard U, Campbell E, Phinyomark A, Laviolette F, Gosselin B and Scheme E (2020), *Interpreting Deep Learning Features for Myoelectric Control: A Comparison With Handcrafted Features.* Front. Bioeng. Biotechnol. 8:158.
- Janne M. Hahne, Meike A. Schweisfurth, Mario Koppe and Dario Farina, Simultaneous control of multiple functions of bionic hand prostheses: Performance and robustness in end users, Sci. Robotics 3, eaat3630
- Castellini, Claudio and van der Smagt, Patrick, *Surface EMG in advanced hand prosthetics,* 2009, Biological Cybernetics
- Carles Igual, Luis A. Pardo Jr., Janne M. Hahne, and Jorge Igual. 2019. Myoelectric Control for Upper Limb Prostheses Electronics 8, 11.
- Connan, Mathilde / Sierotowicz, Marek / Henze, Bernd / Porges, Oliver / Albu-Schäffer, Alin / Roa Garzon, Máximo Alejandro / Castellini, Claudio, Learning to teleoperate an upper-limb assistive humanoid robot for bimanual daily-living tasks, 2021