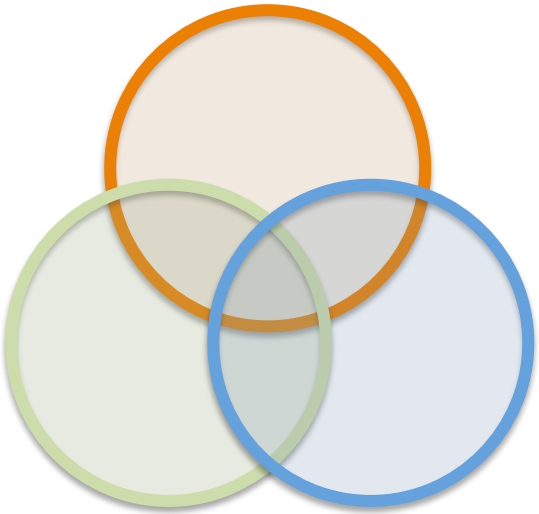
Machine Learning for the Quantified Self



Chapter 7

Predictive Modeling Without Notion of Time

Predictive modeling without notion of time

- Ok, let us start learning
- We will consider learning algorithms that do not take time into account explicitly
  - Of course we do have the temporal features we identified before
- We first need to consider the learning setup
  - Do we want to learn individual models?
  - Do we want to learn to predict well for the future (only for temporal data), or for unseen individuals?

Learning setup

- Here are the options we have (validation set done in the same way):

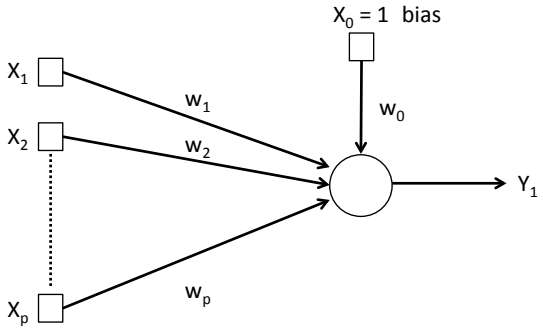
	Data	
Level	temporal	non-temporal
individual	$\mathbf{X_{train,qs_i}} = x_{1,qs_i}, \ldots, x_{n_{train},qs_i}$ $\mathbf{X_{test,qs_i}} = x_{n_{train}+1,qs_i}, \ldots, x_{N_{qs_i},qs_i}$	$\mathbf{X_{train,qs_i}} \subset \mathbf{X_{qs_i}}$ where $\lvert\mathbf{X_{train,qs_i}}\rvert = n_{train} \wedge \mathbf{X_{train,qs_i}} \cap \mathbf{X_{test,qs_i}} = \emptyset$ $\mathbf{X_{test,qs_i}} \subset \mathbf{X_{qs_i}}$ where $\lvert\mathbf{X_{test,qs_i}}\rvert = (N_{qs_i} - n_{train}) \wedge \mathbf{X_{train,qs_i}} \cap \mathbf{X_{test,qs_i}} = \emptyset$
population - unknown users	$\mathbf{X_{train}} \subset \{\mathbf{X_{qs_1}}, \ldots, \mathbf{X_{qs_n}}\}$ where $\lvert\mathbf{X_{train}}\rvert = n_{train} \wedge \mathbf{X_{train}} \cap \mathbf{X_{test}} = \emptyset$ $\mathbf{X_{test}} \subset \{\mathbf{X_{qs_1}}, \ldots, \mathbf{X_{qs_n}}\}$ where $\lvert\mathbf{X_{test}}\rvert = (n - n_{train}) \wedge \mathbf{X_{train}} \cap \mathbf{X_{test}} = \emptyset$	
population - unseen data of known users	$\mathbf{X_{train}} = \{\mathbf{X_{train,qs_1}}, \ldots, \mathbf{X_{train,qs_n}}\}$ (Note that we refer to the definition of the training sets specified in the row for the individual level here) $\mathbf{X_{test}} = \{\mathbf{X_{test,qs_1}}, \ldots, \mathbf{X_{test,qs_n}}\}$	

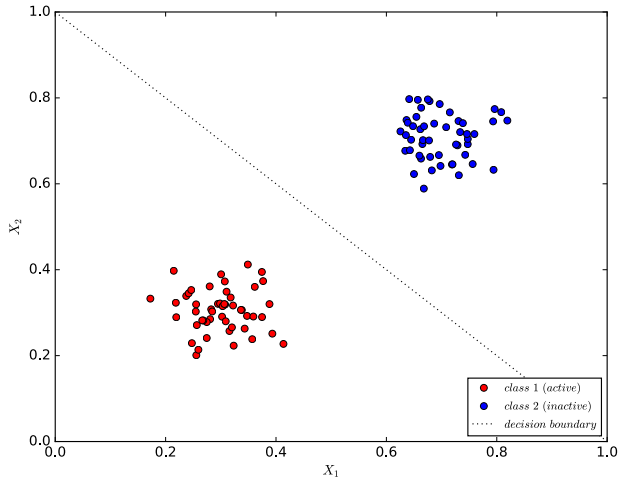
Learning algorithms

- Assume you know the basic learning algorithms
- Quick recap, two slides per learning algorithm
  - Neural Networks
  - Support Vector Machines
  - K-Nearest Neighbor
  - Decision trees
  - Naïve Bayes
  - Ensembles

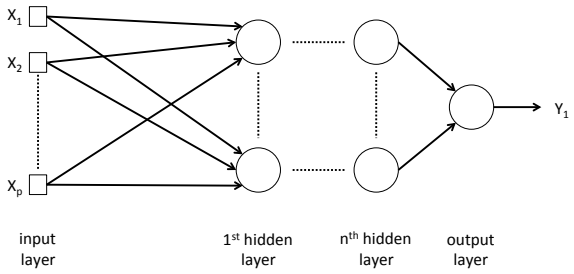
Neural Networks (1)

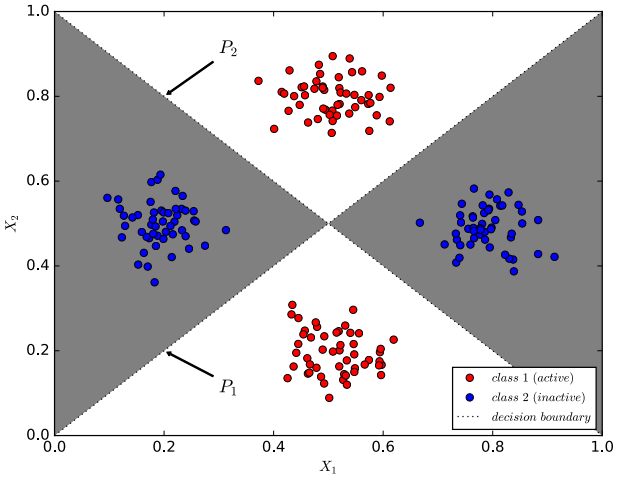
- Perceptron





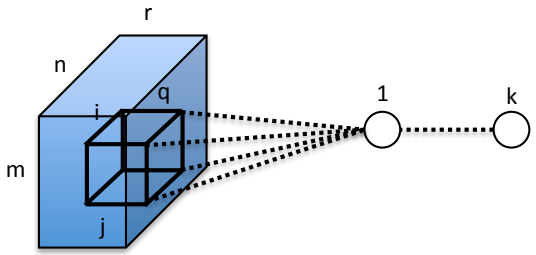
- Multi-Layer Perceptron



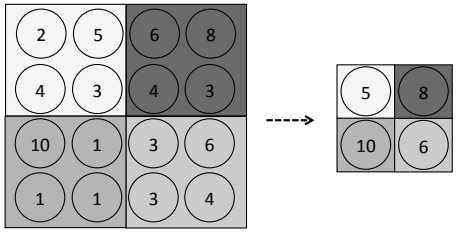


Neural Networks (2)

- Convolutional neural networks
  - Extract features using:
    - Convolutional layers



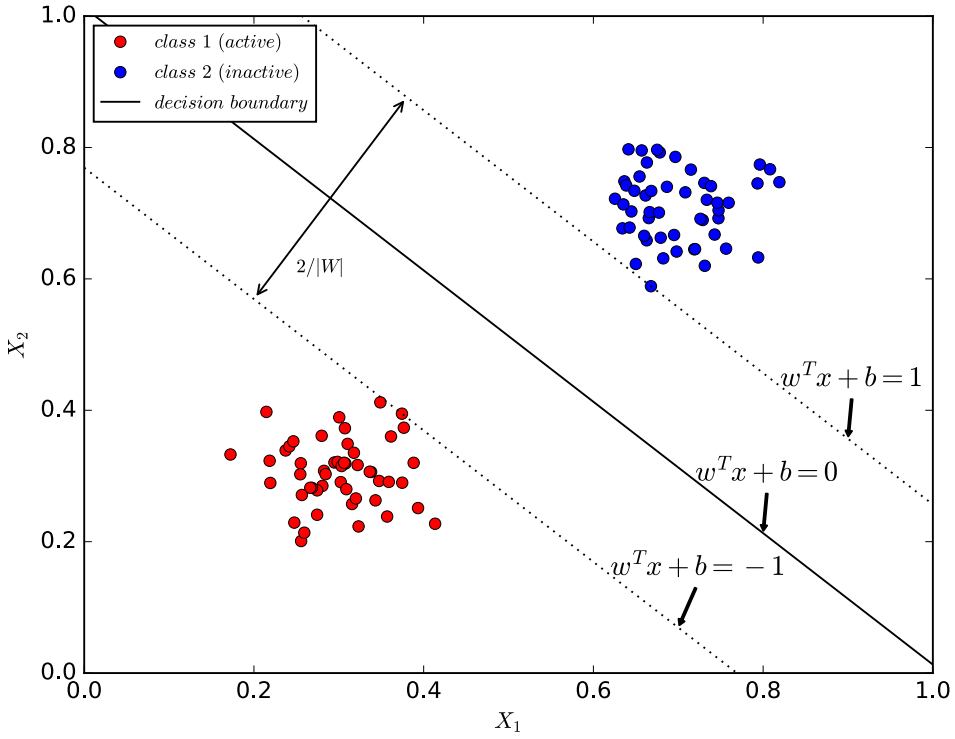
    - Pooling layers



  - Followed by a “regular” neural network

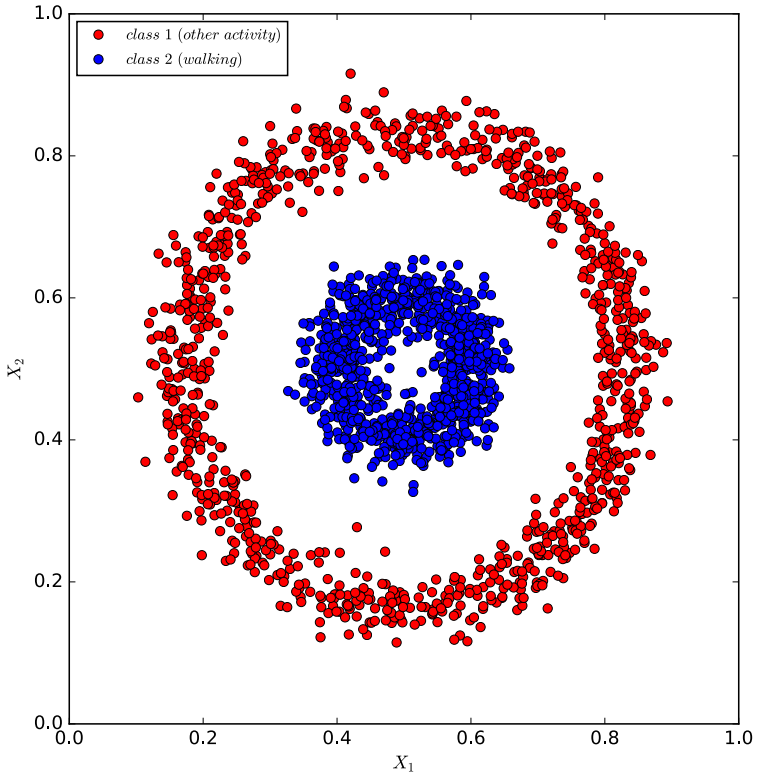
Support Vector Machines (1)

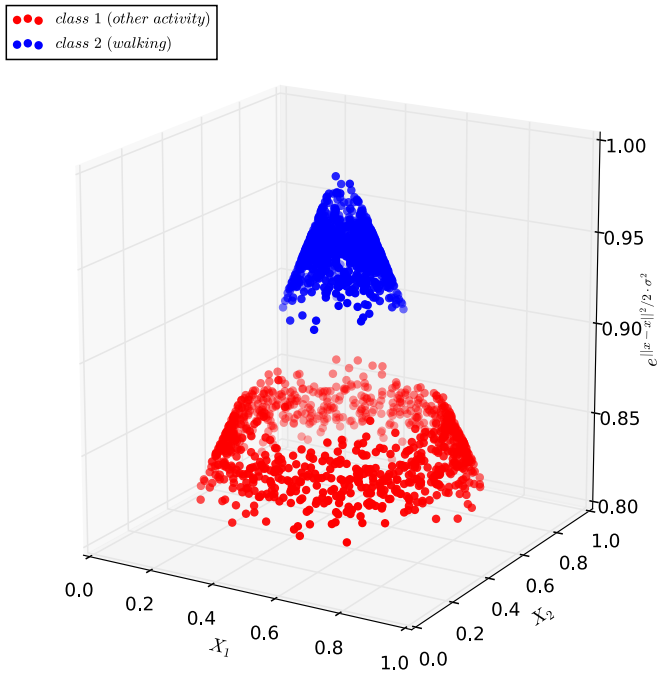
- Find hyperplane that maximizes separation between classes



Support Vector Machines (2)

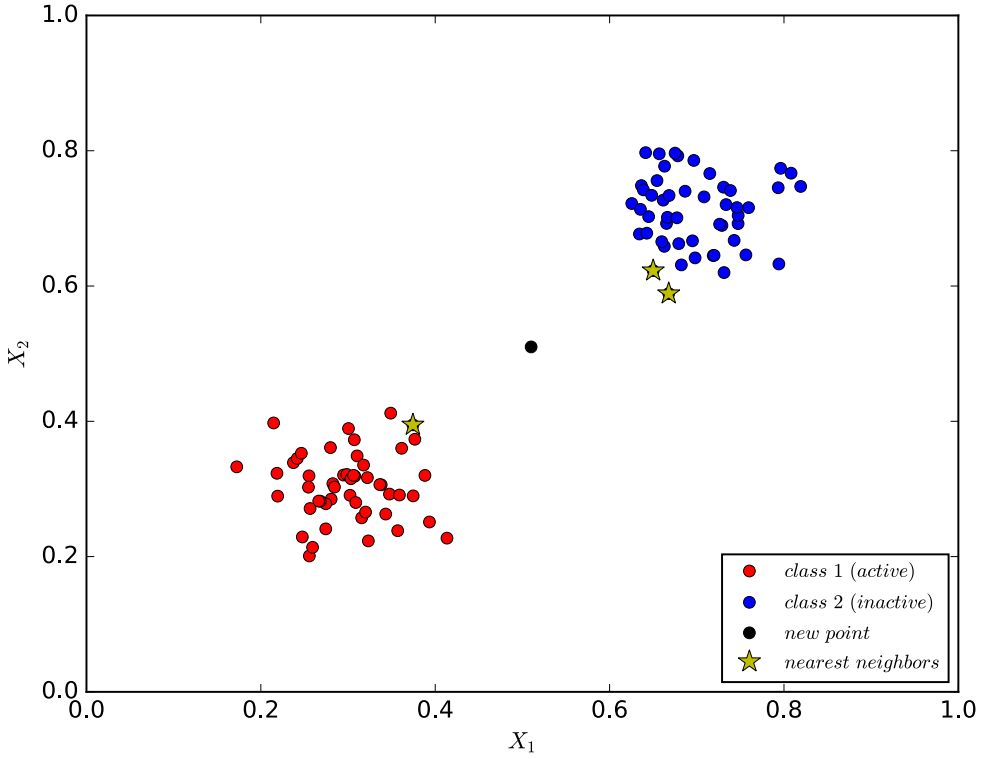
- Kernel functions





K-Nearest Neighbor (1)

- Find k closest examples
  - use distance functions discussed before

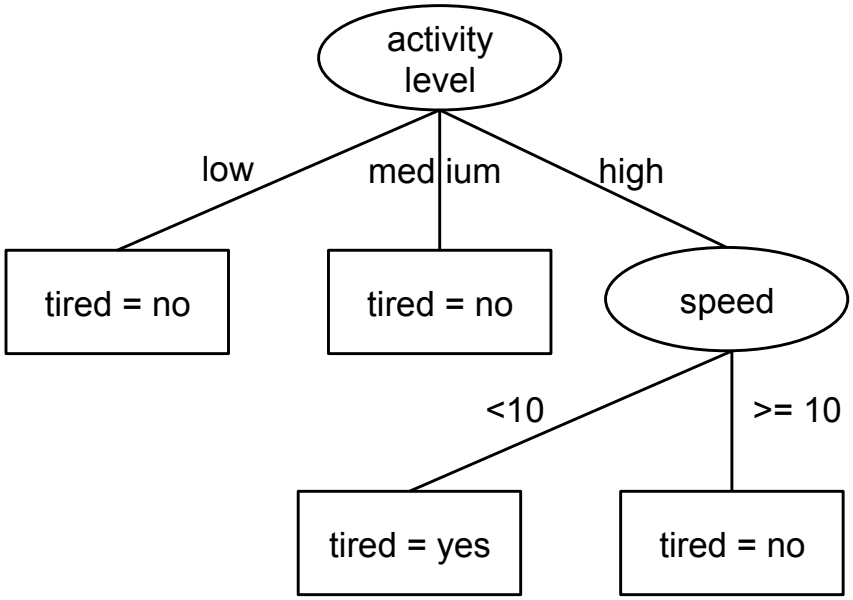


K-Nearest Neighbor (2)

- Assign classes based on some function, for example:
  - Majority class
  - Average value

Decision tree (1)

- Create a tree to decide on target value



Decision tree (2)

- Use criterion to decide on which attributes are most important (i.e. should be selected first)
  - Categorical target:

$$entropy(\mathbf{G}) = \sum_{v \in \mathcal{G}} -p_v log_2 p_v \text{ where } p_v = \frac{|\{g_i \in \mathbf{G} | g_i = v\}|}{|\mathbf{G}|}$$

$$gain(X_i, \mathbf{X}, \mathbf{G}) = entropy(\mathbf{G}) - \sum_{v \in \mathcal{X}_i} \frac{|\{x_j \in \mathbf{X} | x_j^i = v\}|}{|\mathbf{X}|} \cdot entropy(\{g_i \in \mathbf{G} | x_j^i = v\})$$

  - Numerical target:

$$sd\_red(X_i, \mathbf{X}, \mathbf{Y}) = \sigma_Y - \sum_{v \in \mathcal{X}_i} \frac{|\{y_i \in \mathbf{Y} | x_j^i = v\}|}{|\mathbf{Y}|} \sigma_{\{y_i \in \mathbf{Y} | x_j^i = v\}}$$

Naïve Bayes

- We use Bayes formula to express the probability of a target value *g* given our observations *x*:

$$p(G = g|x) = \frac{p(x|G = g)p(G = g)}{p(x)}$$

Ensembles (1)

- Bagging
  - Create multiple models on samples of the data

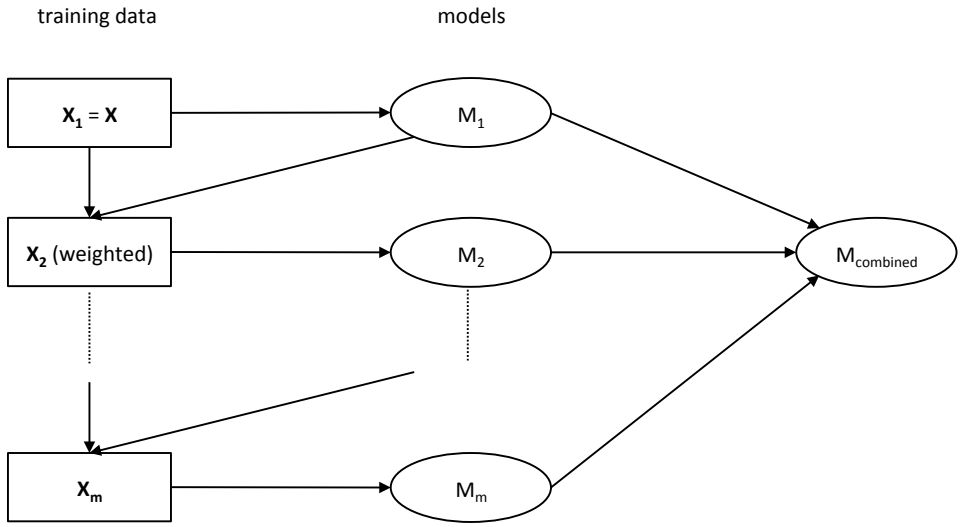
Algorithm 14: Bagging

models = []
for *i in 1,. . . ,m* **do**
  Take a sample $\mathbf{X_i} \subset \mathbf{X}$
  Build a model M_i on training data $\mathbf{X_i}$ using some learning algorithm
  Add M_i to models
end

  - Combine output of models using some aggregation function
  - Tackles *variance* problem

Ensembles (2)

- Boosting
  - Build models sequentially and focus models on where we made mistakes before



  - Focuses on *bias* problem

Practical considerations

- Ok, let us start to apply the learning algorithms!
  - We need to be careful not to overfit (think of the first set of slides)
  - We can select features using forward and backward selection
  - We can consider regularization (punishing more complex models)

Forward selection

- We iteratively add the most predictive feature:

Algorithm 17: Forward selection

```
selected_attributes = {}
performances = []
for k = 1, ..., p do
    best_attribute = ""
    best_performance = ∞
    available_attributes = X \ selected_attributes
    for l ∈ 1, ..., |available_attributes| do
        temp_attributes = selected_attributes ∪ available_attributes_l
        performance = learn_model(temp_attributes, X)
        if performance < best_performance then
            best_attribute = available_attributes_l
            best_performance = performance
        end
    end
    performances[k] = best_performance
    selected_attributes = selected_attributes ∪ best_attribute
end
return performances
```

Backward selection

- We iteratively remove the least predictive feature:

Algorithm 18: Backward selection

```
selected_attributes = X
performances = []
performances[p] = learn_model(selected_attributes, X)
for k = p − 1, . . . , 1 do
    worst_attribute = ""
    best_performance = ∞
    available_attributes = selected_attributes
    for l ∈ 1, . . . , |available_attributes| do
        temp_attributes = selected_attributes \ available_attributes_l
        performance = learn_model(temp_attributes, X)
        if performance < best_performance then
            worst_attribute = available_attributes_l
            best_performance = performance
        end
    end
    performances[k] = best_performance
    selected_attributes = selected_attributes \ worst_attribute
end
return performances
```

Regularization

- We add a term to the error function to punish for more complex models
  - For neural networks:

$$E = \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 + \frac{\lambda}{2} ||w||$$

CrowdSignals

- Remember, we have two tasks for our CrowdSignals dataset:
  1. Predict label (activity)
  2. Predict heart rate
- We consider the first with a non-temporal setup and the second with a temporal setup

Predicting the label

- We currently have a set of binary attributes for each label value
  - We need to transform these to a single categorical attribute
  - How do we do this? What if we have multiple "1"'s per instance?
  - We only use instances where we have a single activity marked with a 1
  - Results in 1837 instances (out of 2895)

Splitting the data

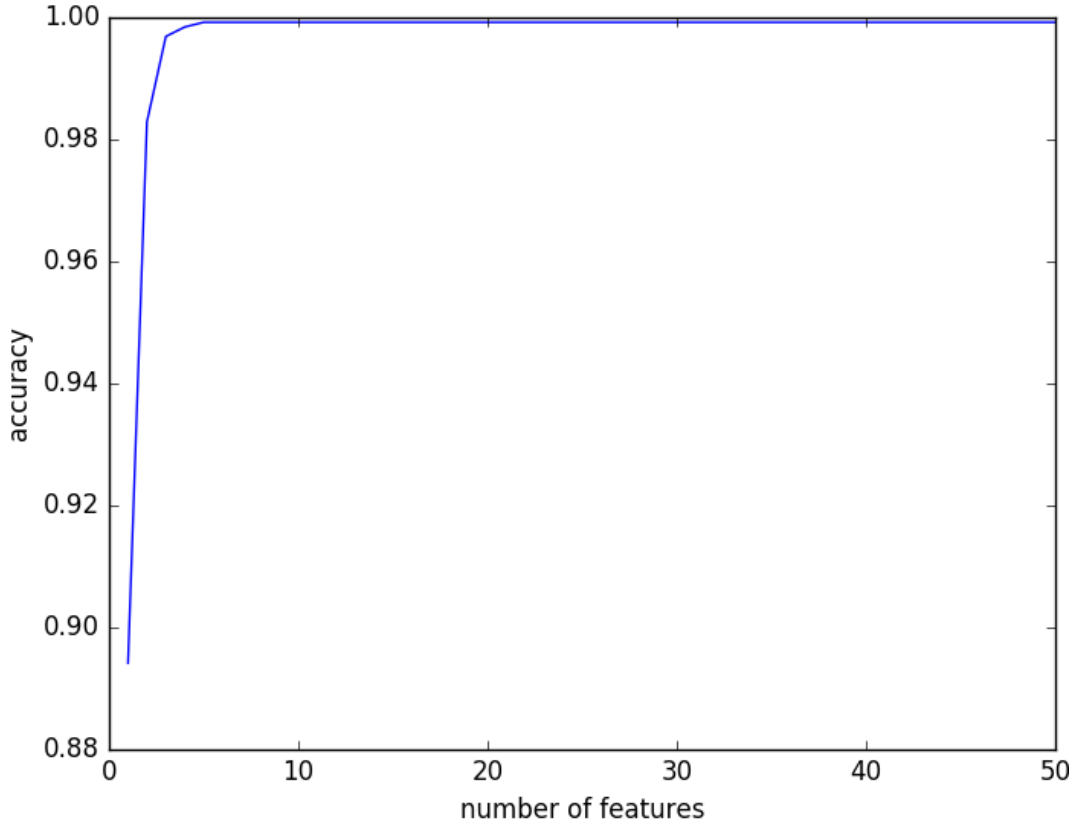
- How do we split the data into training and test?
  - We assign 70% of the instances to the training set and 30% to the test set
  - We do so in a *stratified* way
  - Once again, we ignore the time component

Feature selection (1)

- We have 517 features if we use all the ones we have identified
  - Initial features (21)
  - PCA (7)
  - Time domain features, mean + standard deviation (2 x 28 features = 56)
  - Frequency domain (18 frequencies, 24 initial features = 432)
  - Cluster (1)
- Should we select some features?

Feature selection (2)

- Forward selection:



Feature selection (3)

- We end up with different options which we will try:

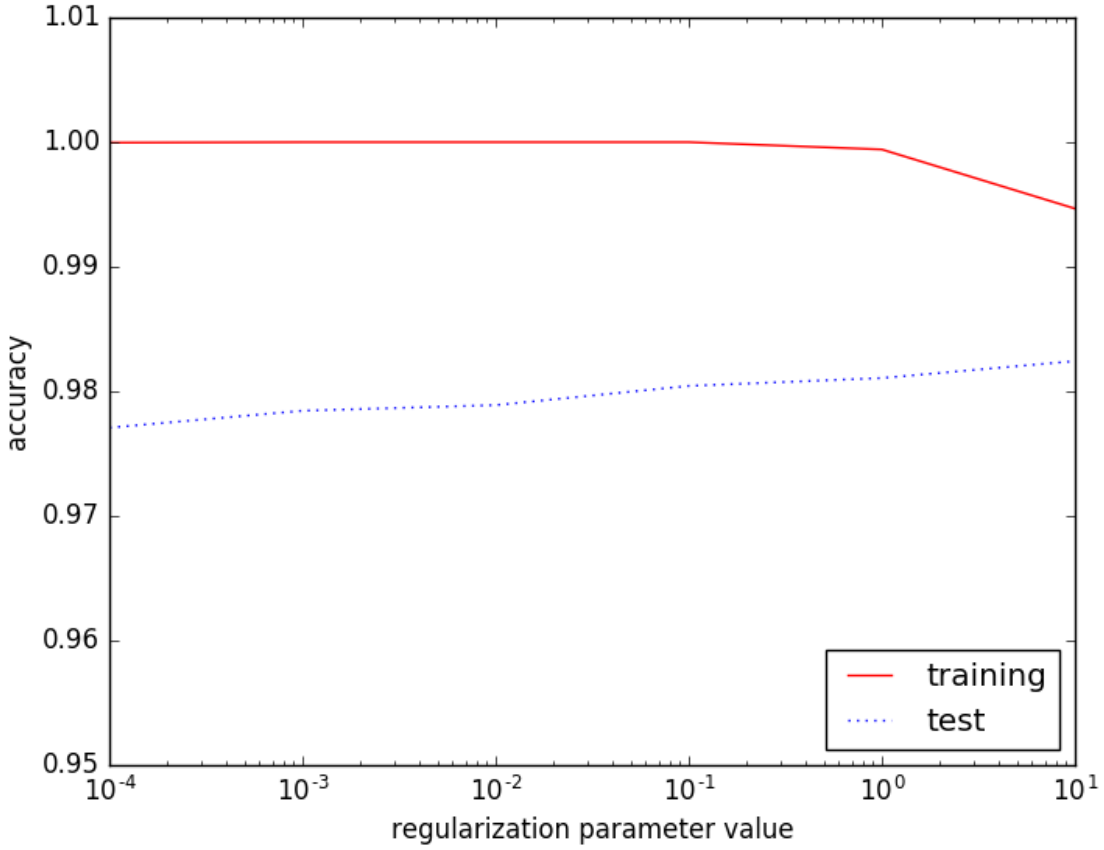
Dataset name	Basic features	PCA	Time-and fre-quency based features	Clusters	#features
Initial set (cleaned)	21				21
Chapter 3	21	7			28
Chapter 4	21	7	56 + 432		516
Chapter 5	21	7	56 + 432	1	517
Selected features			2 + 8		10

Influence of parameter settings (1)

- How does model complexity influence our performance in this case?
  - Remember the theoretical results we have seen
  - We can tailor the parameters of the models
  - We can change the weight of the regularization terms

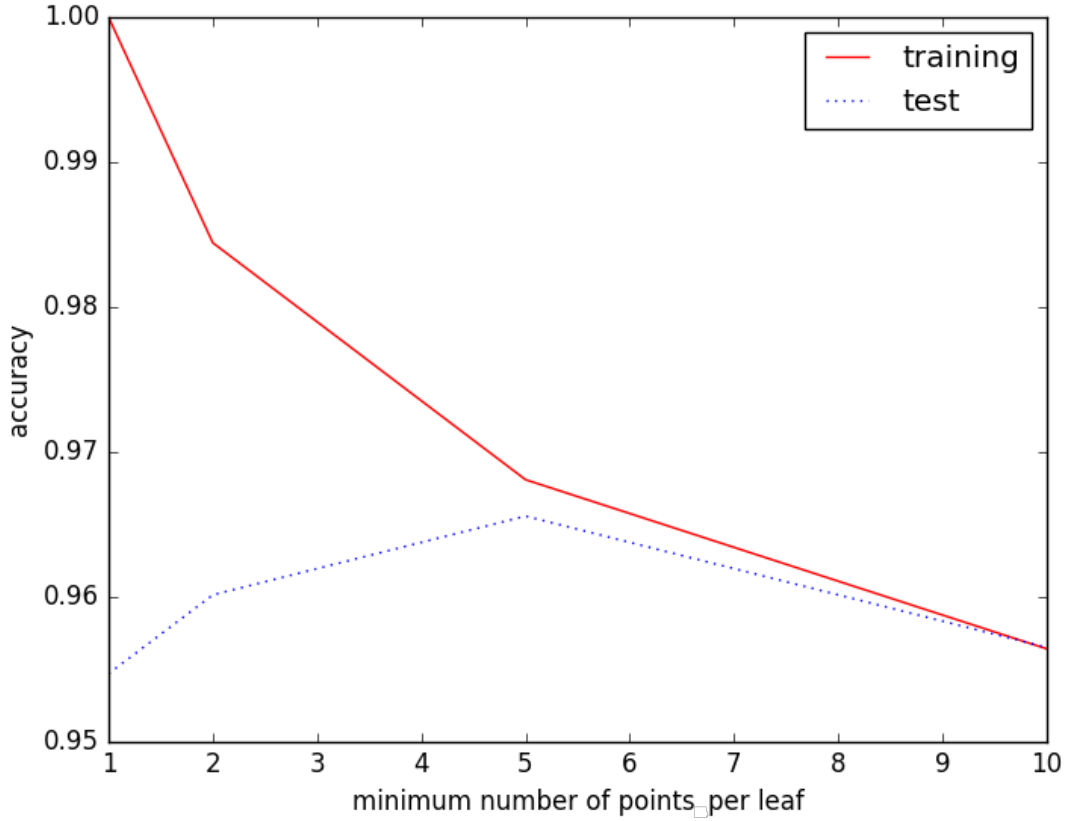
Influence of parameter settings (2)

- Neural network and regularization term



Influence of parameter settings (3)

- Decision tree and points per leaf:

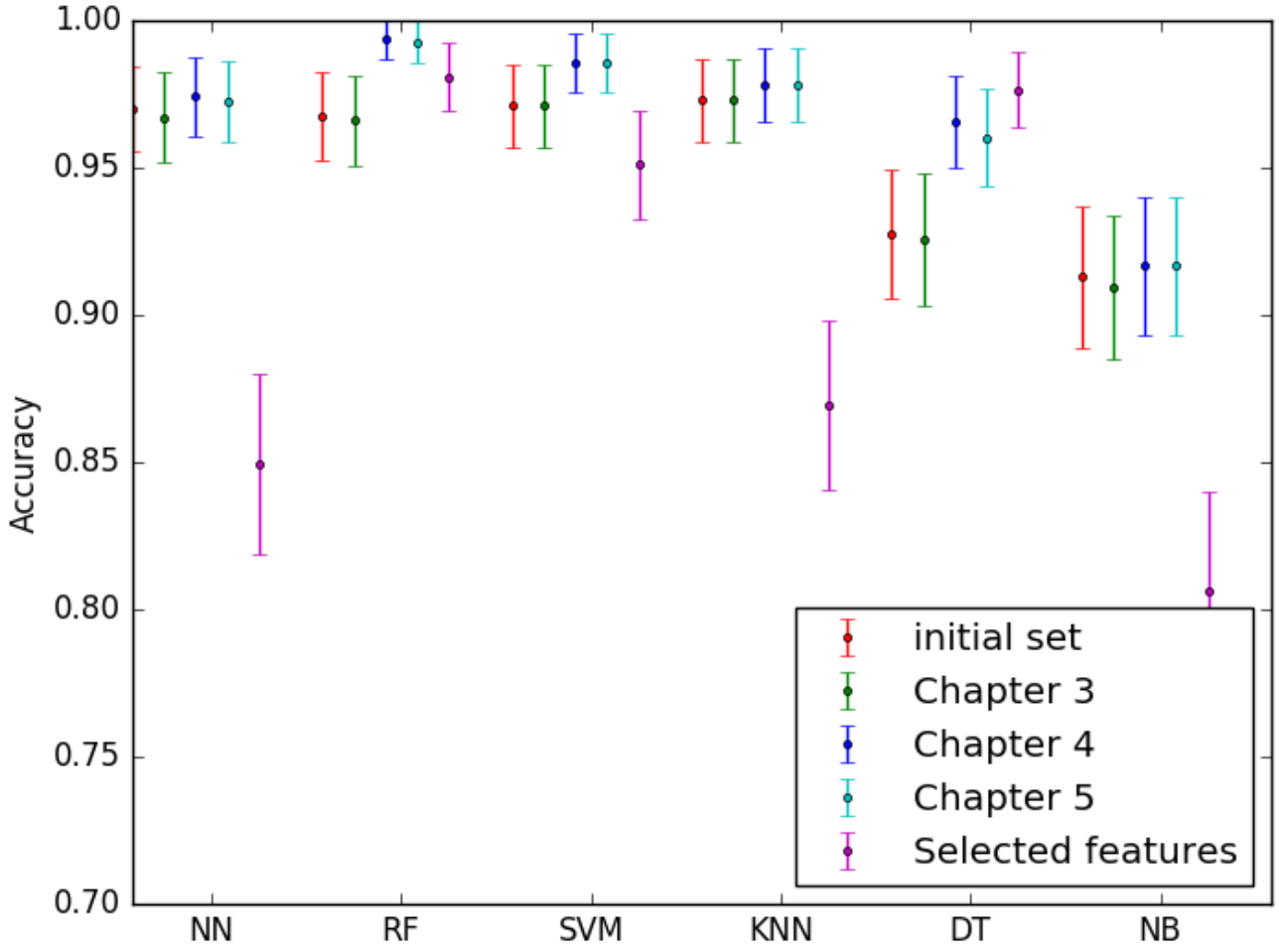


Influence of parameter settings (4)

- We try different parameter settings
  - study the generalizability based on a cross validation on the training set

Algorithm	Variant description	Parameters varied
Neural Network (NN)	Multi-Layer Perceptron with 1 or 2 hidden layers, a logistic activation function and one output node per class	hidden layer composition: {single layer with 5, 10, 25, or 100 neurons, two layer with 100 and 5 neurons or 100 and 10} maximum iterations: $\{1000, 2000\}$
Support Vector Machines (SVM)	SVM with kernel function with one SVM model per class	maximum iterations: $\{1000, 2000\}$ C: $\{1, 10, 100\}$ tolerance = $\{0.001, 0.0001\}$ kernel function: $\{rbf, polynomial\}$
K-Nearest Neighbor (KNN)	KNN model using simple Euclidean distance	k: $\{1, 2, 5, 10\}$
Decision Tree (DT)	Decision tree algorithm following the CART approach	minimum samples per leaf: $\{2, 10, 50, 100, 200\}$ splitting criterion: $\{gini, entropy\}$
Naive Bayes (NB)	Basic Naive Bayes approach	-
Random Forest (RF)	Basic Random forest approach	minimum samples per leaf: $\{2, 10, 50, 100, 200\}$ number of trees: $\{10, 50, 100\}$ splitting criterion: $\{gini, entropy\}$

Results test set (1)



Results test set (2)

- How did we computer the 95% confidence intervals?
  - *a* is the accuracy
  - *n* the number of samples in the test set

$$sd = \sqrt{a(1-a)/n}$$

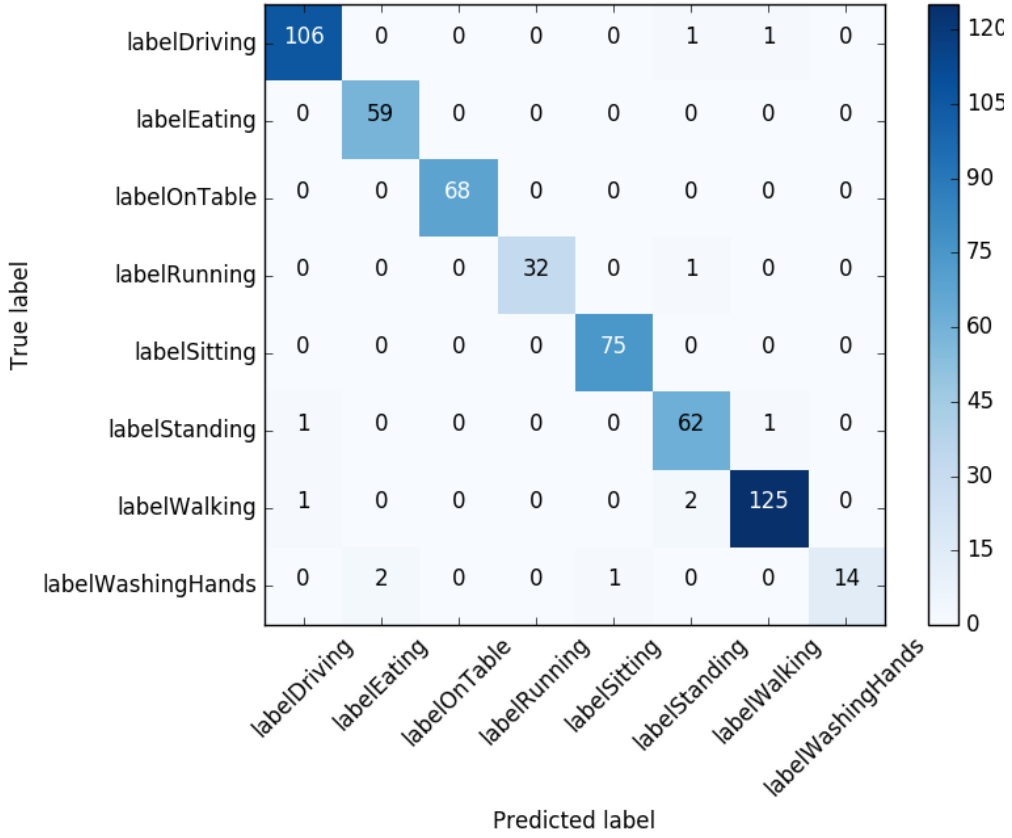
Results test set (3)

- Important features (random forest):

Feature	Importance
gyr_phone_x_temp_std_ws_120	0.2855
press_phone_pressure_temp_mean_ws_120	0.2499
acc_phone_y_freq_0.0_Hz_ws_40	0.2433
mag_watch_y_pse	0.0873
mag_phone_z_max_freq	0.0341
gyr_watch_y_freq_weighted	0.0284
gyr_phone_y_freq_1.0_Hz_ws_40	0.0255
acc_phone_x_freq_1.9_Hz_ws_40	0.0227
acc_watch_y_freq_0.5_Hz_ws_40	0.0164
mag_watch_z_freq_0.9_Hz_ws_40	0.0068

Results test set (4)

- Confusion matrix:

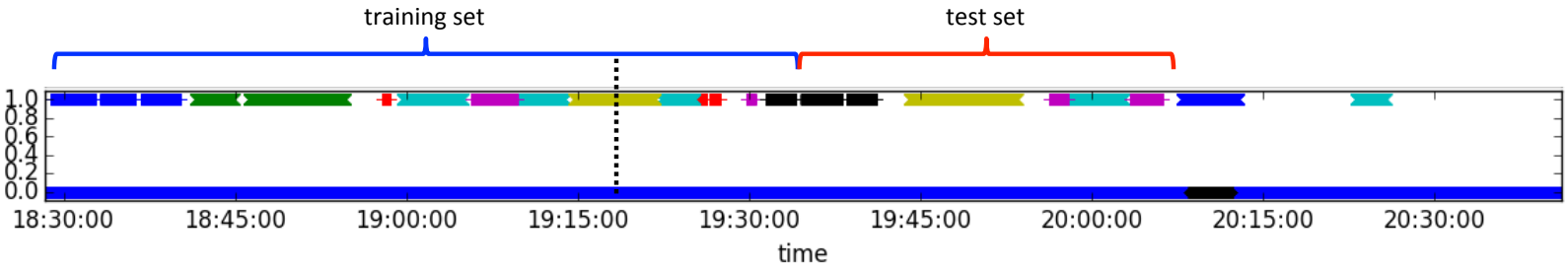


Results test set (5)

- Why are the results this good?
  - Dataset has limited variation
  - Spans only short time period
  - …

Predicting the heart rate

- We take it as a temporal problem
  - Want to learn to predict future time points well
- How do we split the training and test set here?



- Why this way?

Features

Dataset name	Basic features	PCA	Temporal features	Clusters	#features
Initial set (cleaned)	21				21
Chapter 3	28	7			35
Chapter 4	28	7	71 + 432		538
Chapter 5	28	7	71 + 432	1	539
Selected features	1	1	6 + 2		10

Algorithms we tried

Algorithm	Variant description	Parameters varied
Neural Network (NN)	Mutli-Layer Perceptron with 1 or 2 hidden layers and an identity activation function and one output node per class	hidden layer composition: {single layer with 5, 10, 25, or 100 neurons, two layer with 100 and 5 neurons or 100 and 10} maximum iterations: $\{1000, 2000\}$
Support Vector Regression (SVR)	Linear SVR (no kernel function)	maximum iterations: $\{1000, 2000\}$ C: $\{1, 10, 100\}$ tolerance = $\{0.001, 0.0001\}$
K-Nearest Neighbor (KNN)	KNN model using simple euclidean distance	k: $\{1, 2, 5, 10\}$
Decision Tree (DT)	Decision tree algorithm following the CART approach with the mean squared error as a splitting criterion	minimum samples per leaf: $\{50, 100, 200\}$
Random Forest (RF)	Basic Random forest approach with the mean squared error as a splitting criterion	minimum samples per leaf: $\{50, 100, 200\}$ number of trees: $\{10, 50, 100\}$

Results (1)

Approach	NN		RF		SVM		KNN		DT	
Features	training	test	training	test	training	test	training	test	training	test
initial set	725.3 *(1009.3)*	1584.6 *(1417.6)*	582.5 *(857.1)*	1548.0 *(1346.5)*	1155.8 *(1216.8)*	1303.3 *(1067.2)*	186.8 *(574.1)*	2309.7 *(2027.9)*	542.8 *(855.8)*	1496.7 *(1268.4)*
Chapter 3	716.3 *(971.6)*	1510.0 *(1333.6)*	476.0 *(800.9)*	1750.4 *(1500.1)*	1600.2 *(4785.8)*	1465.2 *(1609.4)*	186.8 *(574.1)*	2309.7 *(2027.9)*	537.2 *(850.8)*	1501.4 *(1282.2)*
Chapter 4	523.6 *(846.2)*	1968.0 *(2801.3)*	15.1 *(41.6)*	1310.3 *(1129.2)*	1516.1 *(2783.7)*	1967.4 *(3950.7)*	134.2 *(473.6)*	2259.0 *(1848.2)*	368.2 *(643.8)*	1317.0 *(826.8)*
Chapter 5	557.7 *(861.5)*	2014.7 *(2114.9)*	7.0 *(21.1)*	1384.8 *(1292.9)*	1211.8 *(2051.6)*	1997.2 *(3243.5)*	134.2 *(473.6)*	2259.0 *(1848.2)*	368.2 *(643.8)*	1317.0 *(826.8)*
Selected features	768.9 *(1369.2)*	2825.3 *(2855.4)*	620.7 *(1109.3)*	2614.8 *(2613.4)*	808.5 *(1909.1)*	2598.2 *(2944.2)*	286.1 *(719.9)*	2571.1 *(2896.0)*	538.6 *(1039.8)*	2740.0 *(2878.4)*

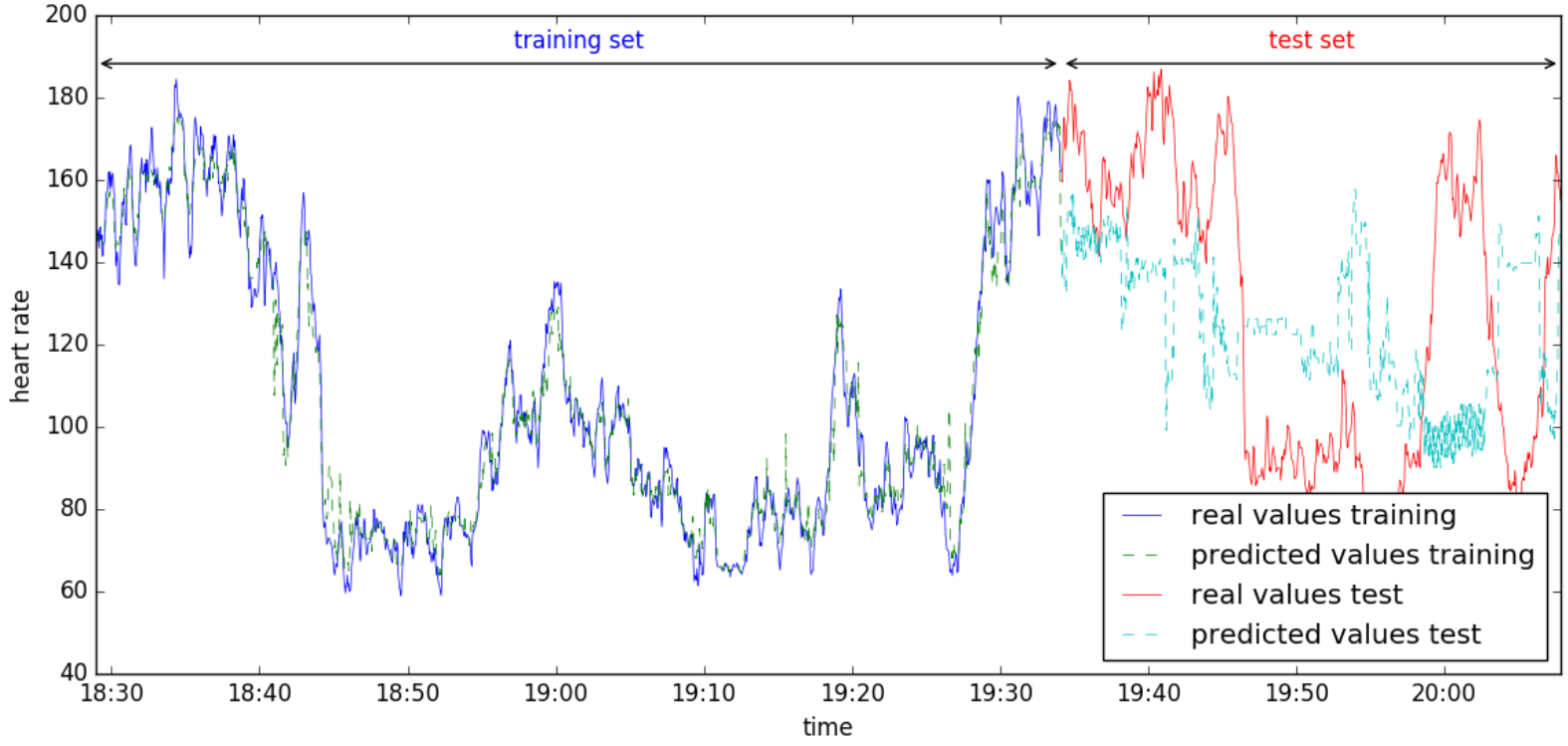
Results (2)

- Standard deviation computed based on predictions per instance on the test set
- Predictors random forest:

Feature	Importance
acc_watch_y_temp_mean_ws_120	0.4631
press_phone_pressure_temp_mean_ws_120	0.1398
mag_watch_x_temp_mean_ws_120	0.0465
temp_pattern_labelEating(b)labelEating	0.0404
mag_watch_y_temp_mean_ws_120	0.0385
temp_pattern_labelEating	0.0383
labelEating	0.0382
pca_4_temp_mean_ws_120	0.0208
pca_7	0.0131
pca_7_temp_mean_ws_120	0.0112

Results (3)

- Visual predictions:



Why is it so bad?

- Ideas?