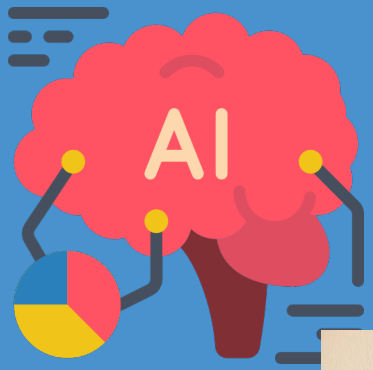
Artificial Intelligence in the Automotive Industry

Künstliche Intelligenz in der Automobilindustrie

Dr. Michael Nolting

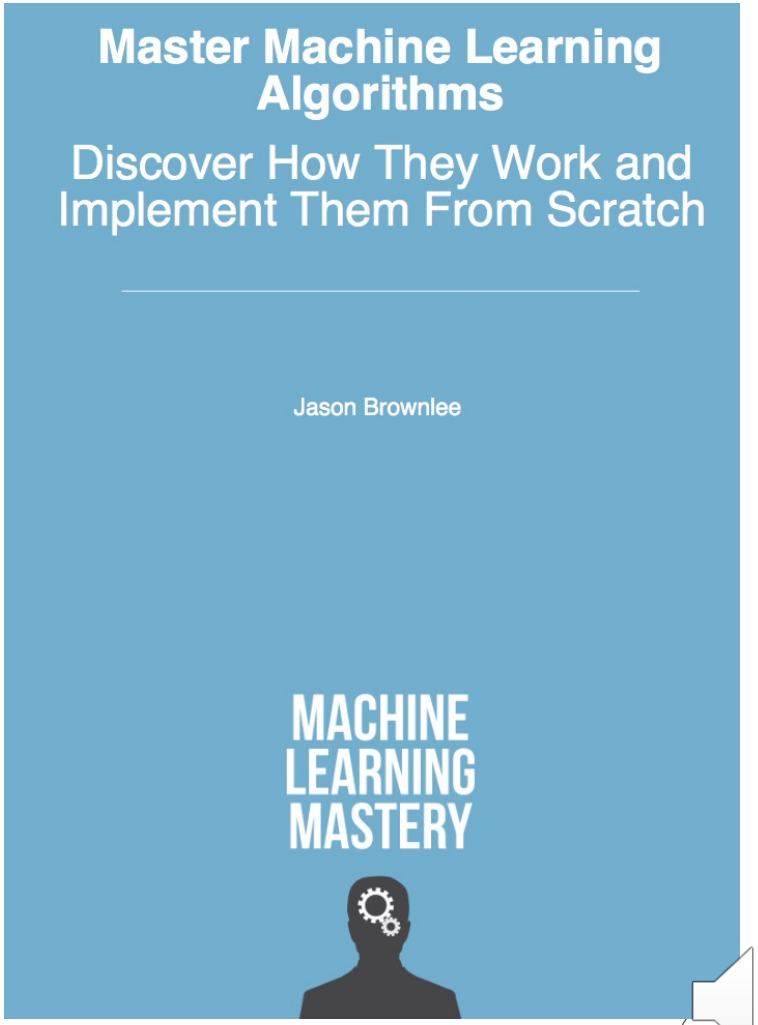
Lecture 5, 2021-05-21



Tutorials

- Tutorials are based on the book „Master Machine Learning Algorithms“ (big thanks to Jason Brownlee, www.machinelearningmastery.com)
- All tutorials are pre-implemented in Excel (Excel spreadsheet will be provided)
- All Excel spreadsheets have to be re-implemented in Python
  - You will roughly need 30 min to read the provided tutorial
  - Please try to re-implement each spreadsheet in Python (probably it will take 30 minutes per spreadsheet)
- If you do not have any Python experience, please do this free tutorial: https://www.codecademy.com/learn/learn-python

The tutorials are optional and not relevant for the oral exams



Lecture Overview

1. Introduction: The ABC of AI	**7. Supervised Learning: (Deep) Neural Networks**
Tutorial 1: Machine Learning Basics	Tutorial 7: Naïve Bayes
2. AI Methods & Automotive Value Chain	**8. Vision: One SOP per day**
Tutorial 2: Linear Regression	Tutorial 8: Gaussian Naïve Bayes
3. Autonomous Driving & Computer Vision	**9. Mission: Own your Code! Own your Data!**
Tutorial 3: Linear Regression Gradient Descent	Tutorial 9: k-Nearest Neighbors
4. Supervised Learning: Regression	**10. Organization: Building HPT**
Tutorial 4: Logistic Regression	Tutorial 10: Support Vector Machines
5. Supervised Learning: Classification	**11. Q & A – Exam**
Tutorial 5: Linear Discriminant Analysis	12. Bonus: 3 to 4 presentations from our PhD students
6. Unsupervised Learning: Clustering	
Tutorial 6: Classification and Regression Trees	

Agenda

01 Introduction (Training / Validation)

02 Methods
(Logistic Regression, Nearest Neighbors, Support Vector Machines)

03 Applications

04 Summary

Classification

Definition: "Systematic arrangement in groups or categories according to established criteria"

https://www.merriam-webster.com/dictionary/classification

Photo by Sereja Ris on Unsplash

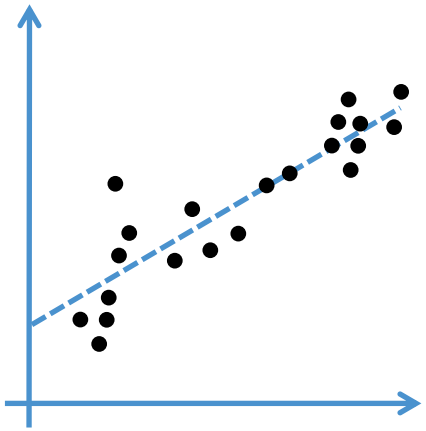
Photo by Hari Menon on Unsplash

Classification

"Systematic arrangement in groups or categories according to established criteria"

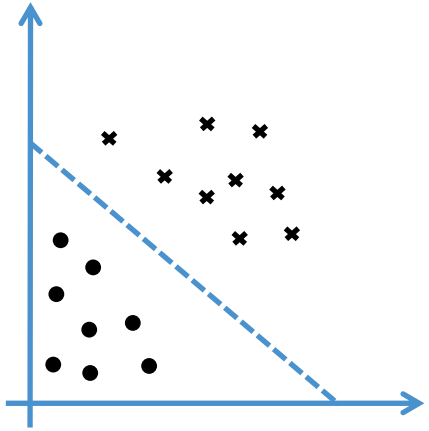
Recap – Supervised & Unsupervised Learning

Regression



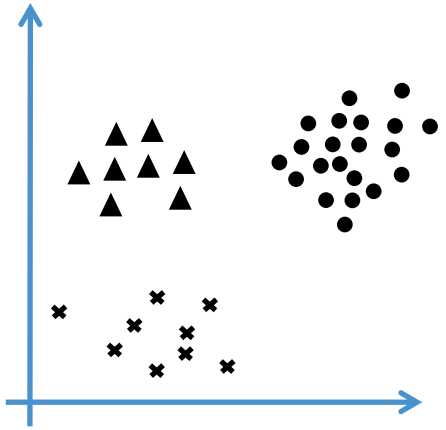
- Predicting **numerical values**
- Supervised

Classification



- Predicting **discrete** valued output
- Supervised

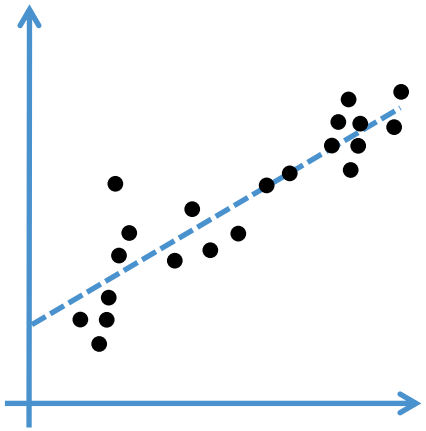
Clustering



- Predicting **discrete** valued output
- Unsupervised

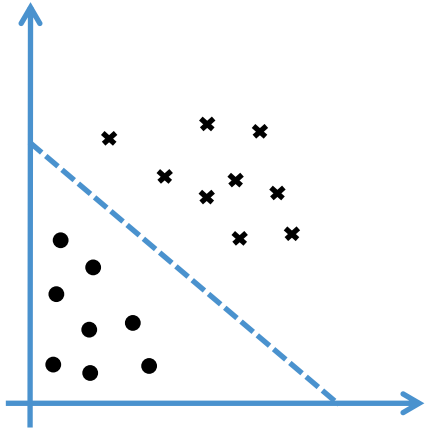
Applications

Regression



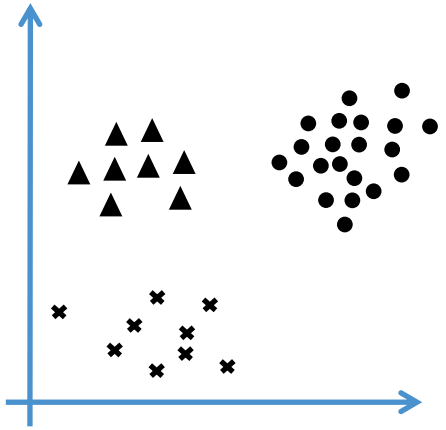
- House prices
- Sales planning
- Child's weight gain

Classification



- Object detection
- Spam detection
- Cancer detection

Clustering

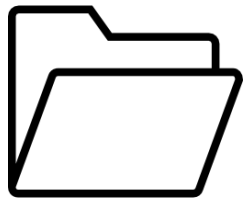


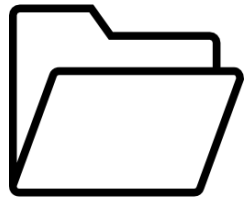
- Genome patterns
- Personalized news
- Point Cloud (LIDAR) processing

Data
(Features)

Classifier

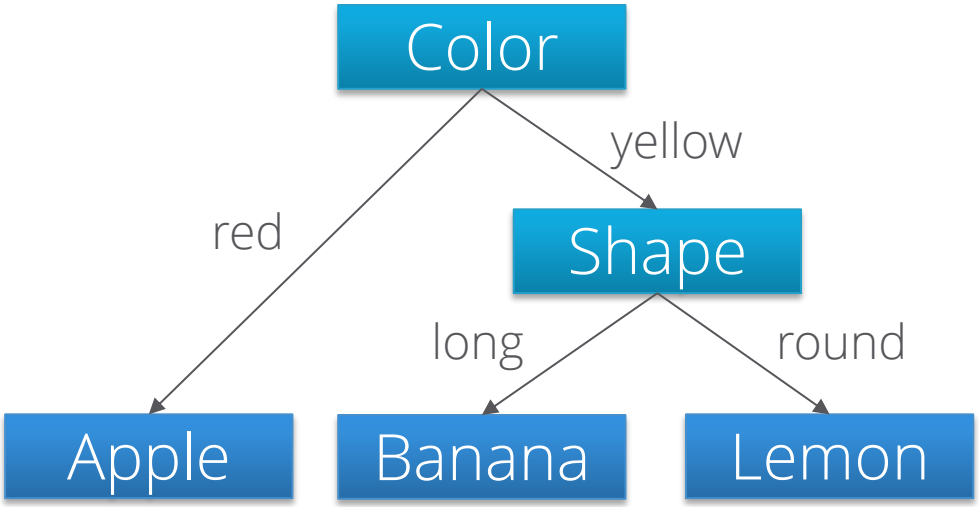
Classes





Email (Keywords, ...)	Spam?	Yes/No
Tumor (Size, ...)	Malignant?	Yes/No
Object (Color, ...)	What type?	Cat/Car/Fruit/...

Classic Method vs. Machine Learning Method

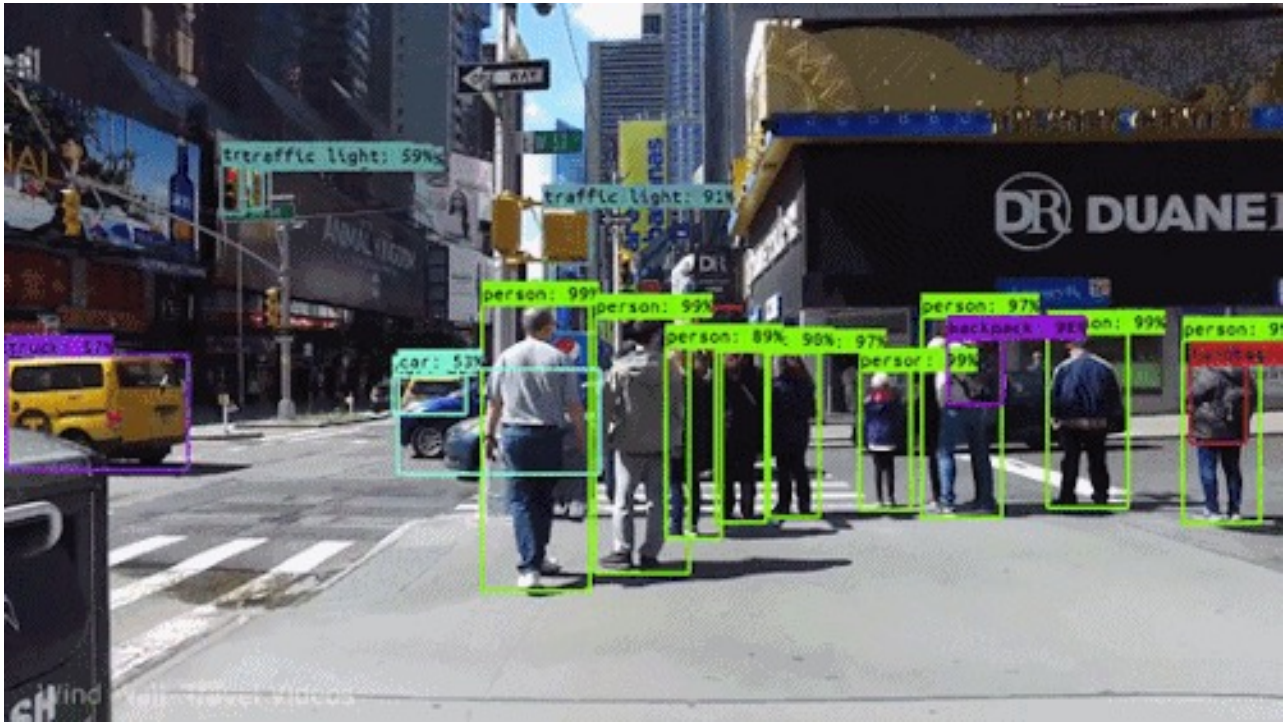


- E.g. Decision tree
  - Use a-priori knowledge to formulate classification rules

- Advantages of machine learning
  - Automatic generation of a-priori knowledge
  - Automatic generation of complex classification rules

  → Suitable for extreme large datasets

Classification - Example



Object Classification → Object Detection → Object Tracking

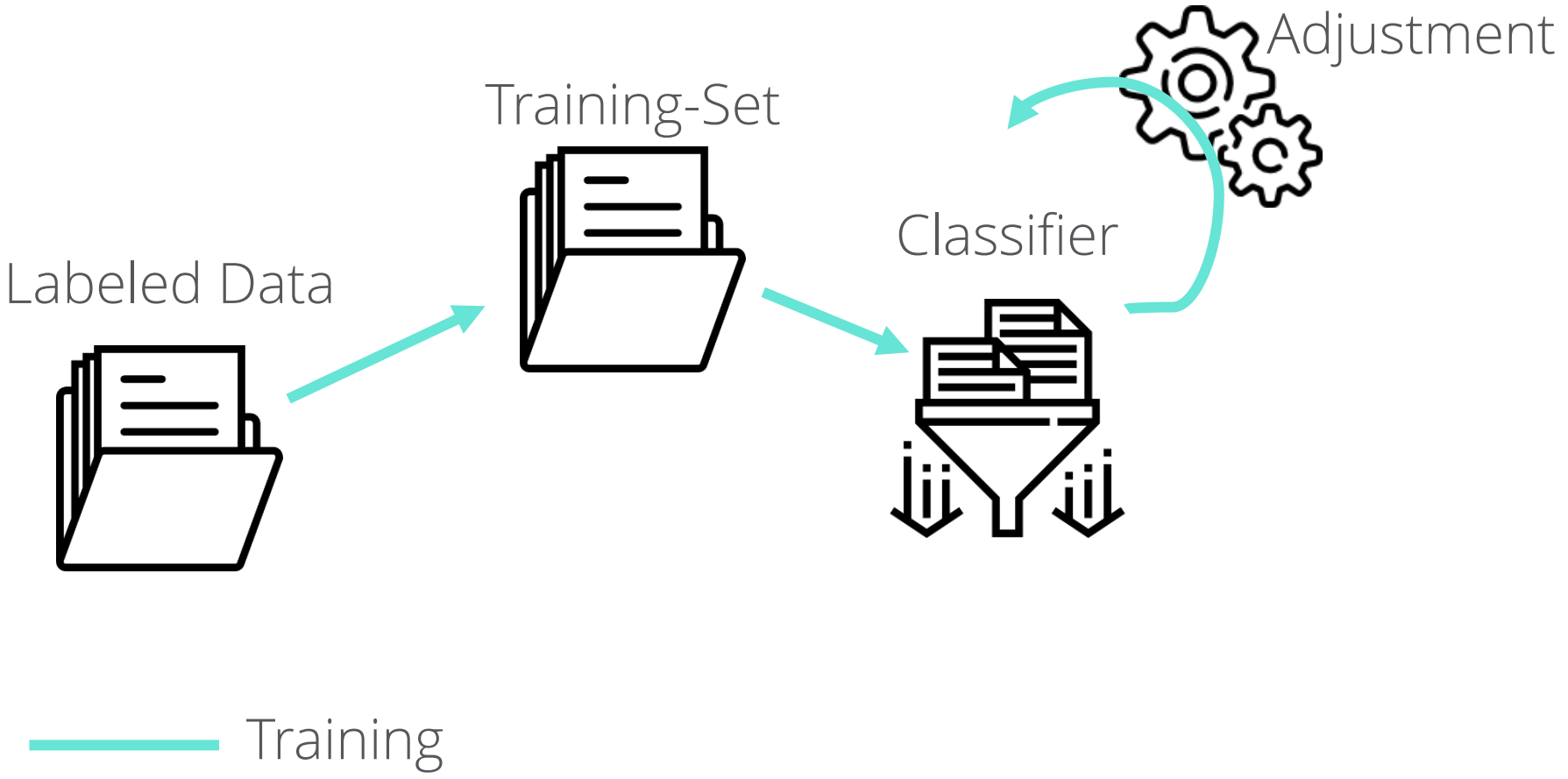
Formal Definition - Classification

$$C_{M(\theta)}: D \rightarrow Y$$

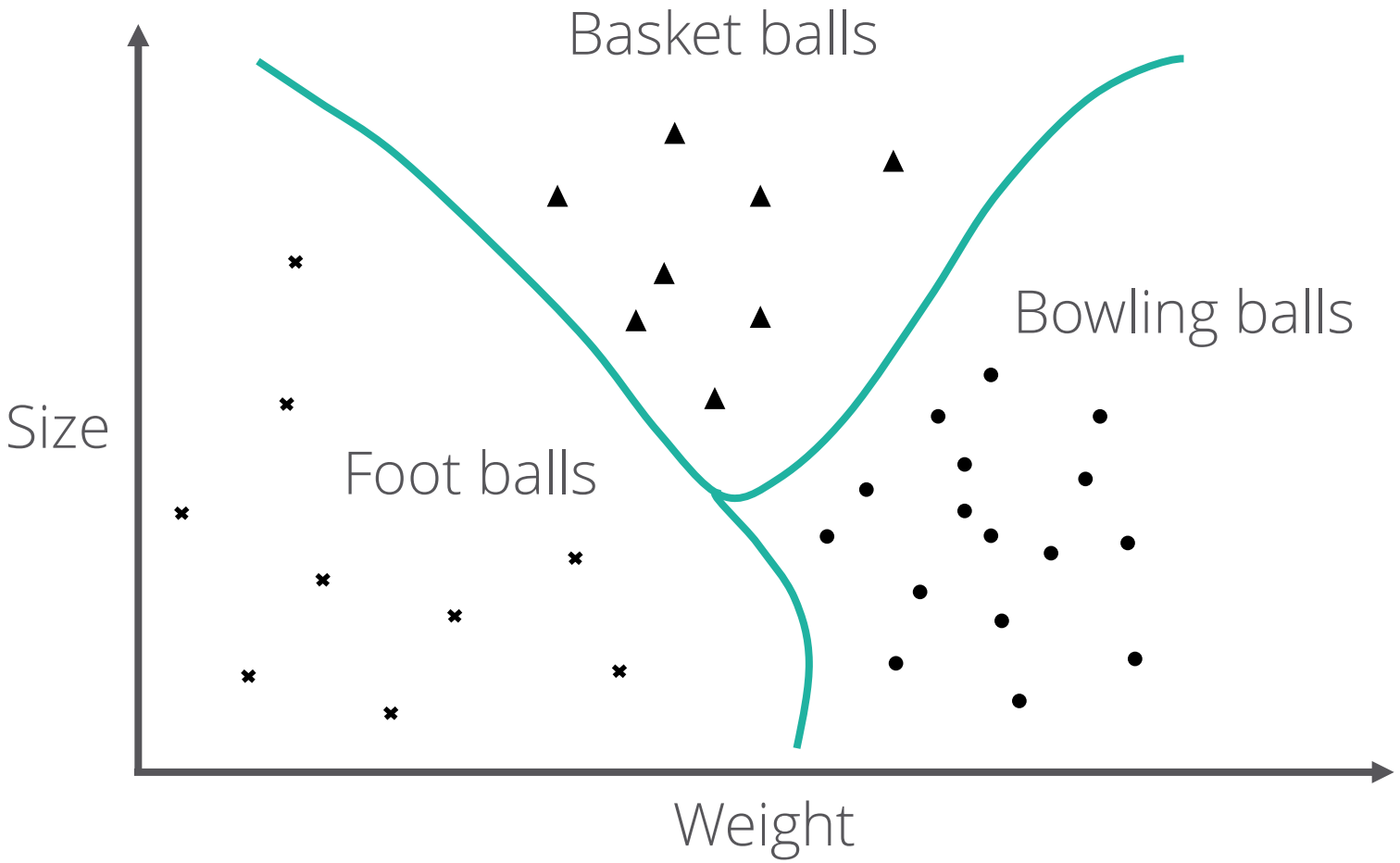
- Classifier *C*
- Model *M* with parameter *θ*
- Data space *D*
- *Labels Y*
- *Training Data O* $\subseteq$ *D* with known labels

- Training: Given O, find optimal parameter *θ*
- *Classification: Apply* $C_{M(\theta)}$ *on objects from D*

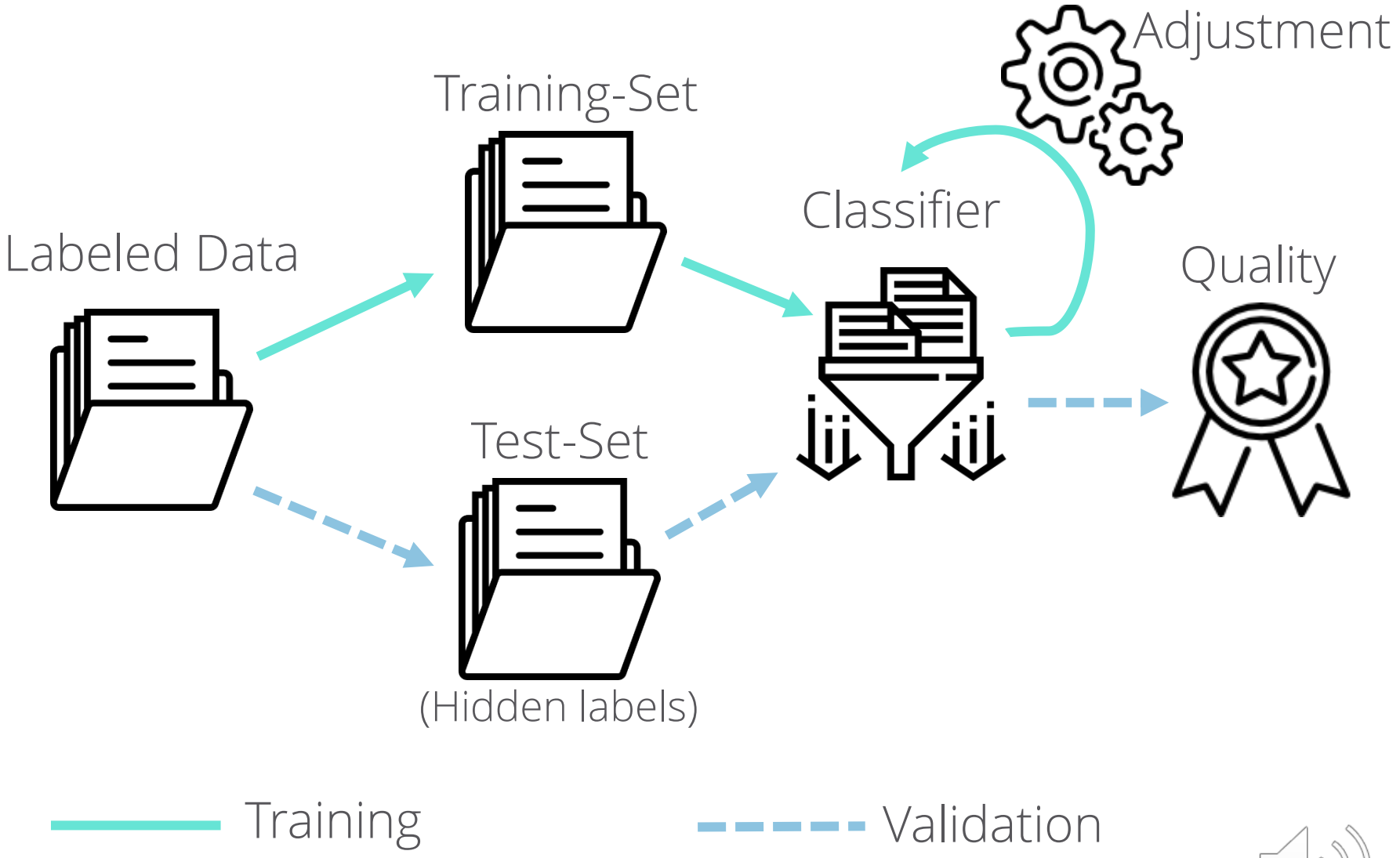
Supervised Learning - Classification



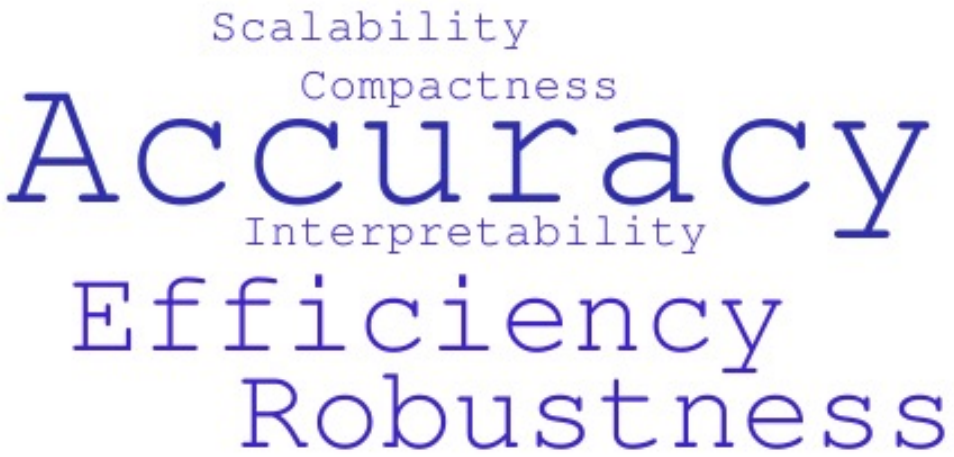
Classifier Training – Balls?



Supervised Learning - Classification



Quality Measures for Classifiers



Evaluation of Classifiers

- k-fold Cross Validation (e.g., 3)
  - Decompose data set evenly into k subsets of (nearly) equal size
  - Iteratively use k–1 partitions as training data and the remaining single partition as test data.

- Additional requirement: stratified folds
  - Class distributions in training and test set should represent the class distribution in *D* (or at least in *O*)

- Standard: 10-fold stratified cross validation

	Fold #1 - Car	Fold #2 - Car	Fold #3 - Pedestrian
Set #1	T	T	V
Set #2	T	V	T
Set #3	V	T	T

Confusion Matrix

Actual class

13 animals (8 cats, 5 dogs)

Predicted class

	Cat	Dog	
Predicted condition positive: Cat	5 TP	2 FP	Positives
Predicted condition negative: Dog	3 FN	3 TN	Negatives

- Recall: TP/(TP+FN)
- Precision: TP/(TP+FP)
- Specificity: TN/(TN+FP)

- True Positives: TP
- False Positives: FP (type 1 error)
- True Negatives: TN
- False Negatives: FN (type 2 error)

Agenda

01 Introduction (Training / Validation)

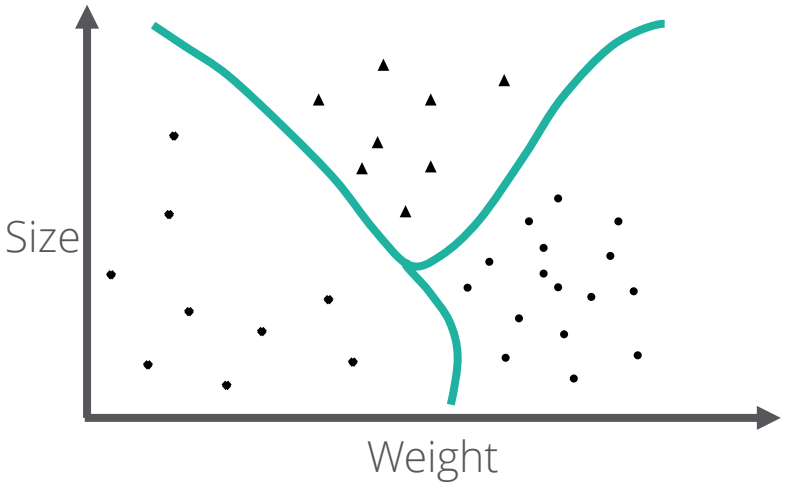
02 **Methods**
(Logistic Regression, Nearest Neighbors, Support Vector Machines)

03 Applications

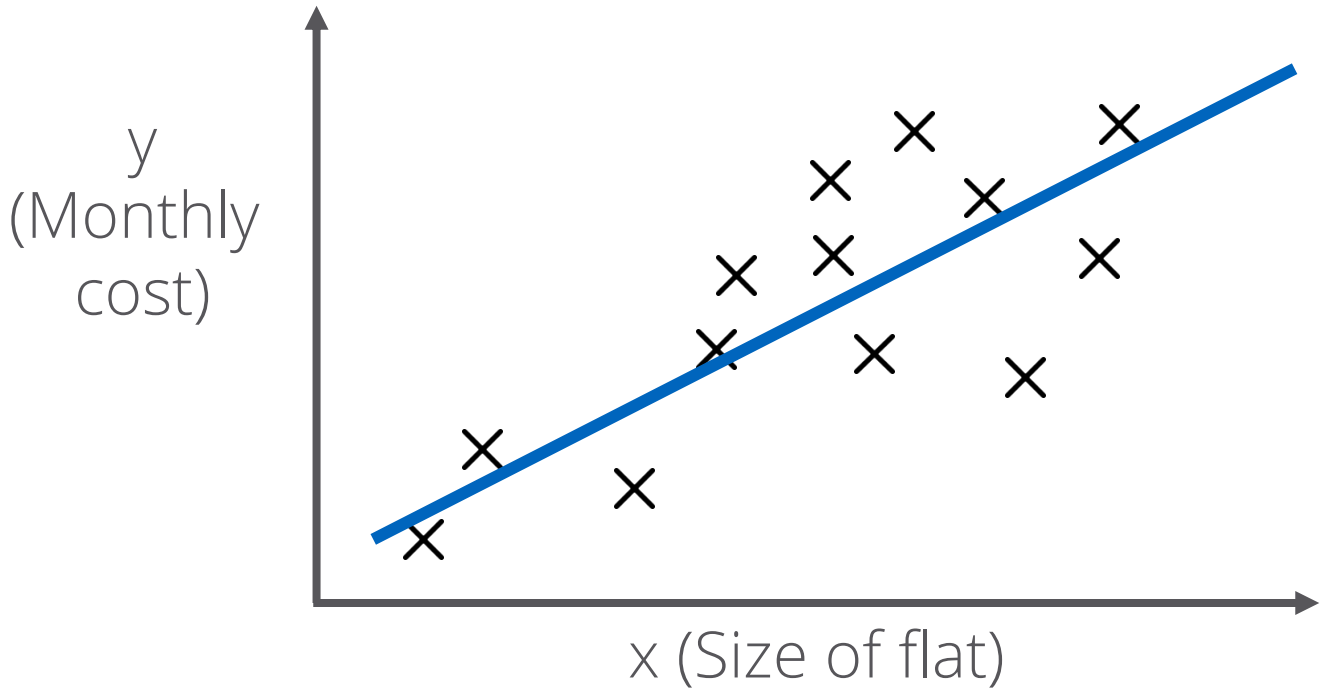
04 Summary

Overview of Methods

1. Decision Trees
2. **Logistic Regression**
3. **Nearest Neighbors**
4. **Support Vector Machine**
5. Neural Networks
6. ...

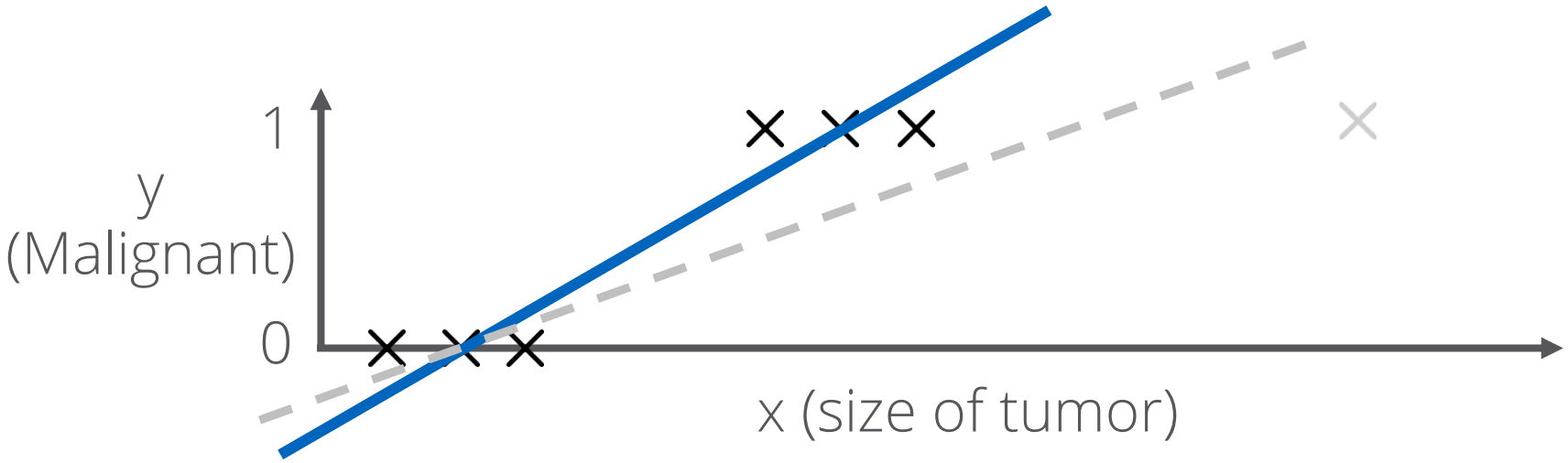


Recap Linear Regression



$$y = h_{\Theta}(x), y \in R$$

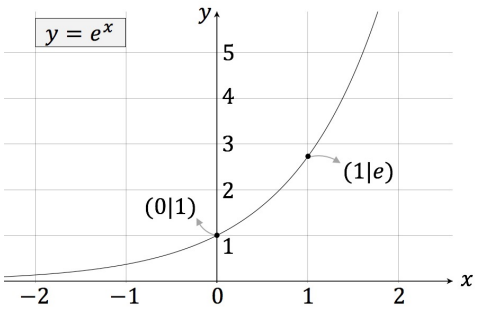
Linear Regression for Classification

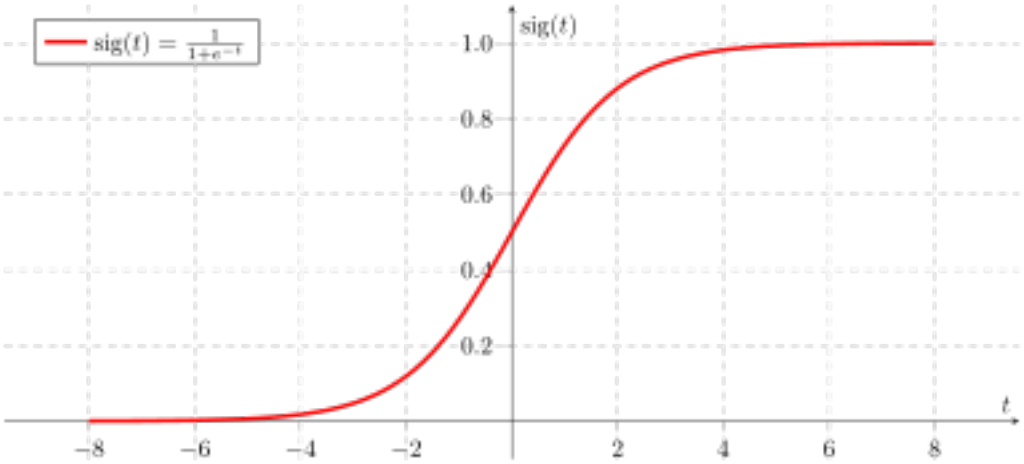


$$y = h_\Theta(x), y \in R$$

Sigmoid Function

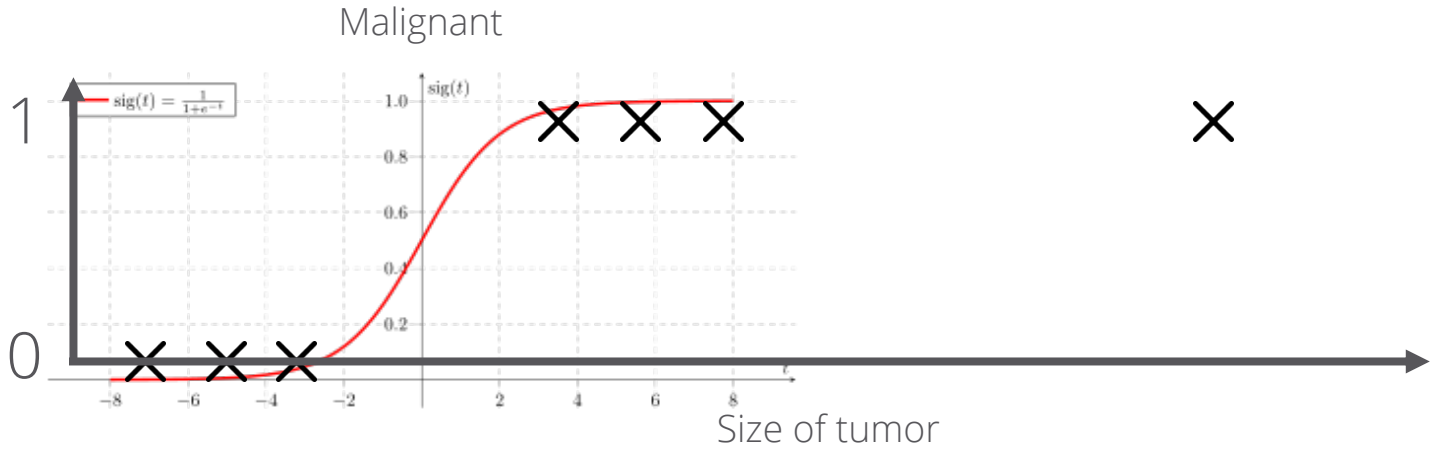
$$\mathrm{sig}(t) = \frac{1}{1 + e^{-t}}$$





t	e^{-t}	$1+e^{-t}$	$1/(1+e^{-t})$
$\to \infty$	$\to 0$	$\to 1$	$\to 1$
0	1	2	0,5
$\to -\infty$	$\to \infty$	$\to \infty$	$\to 0$

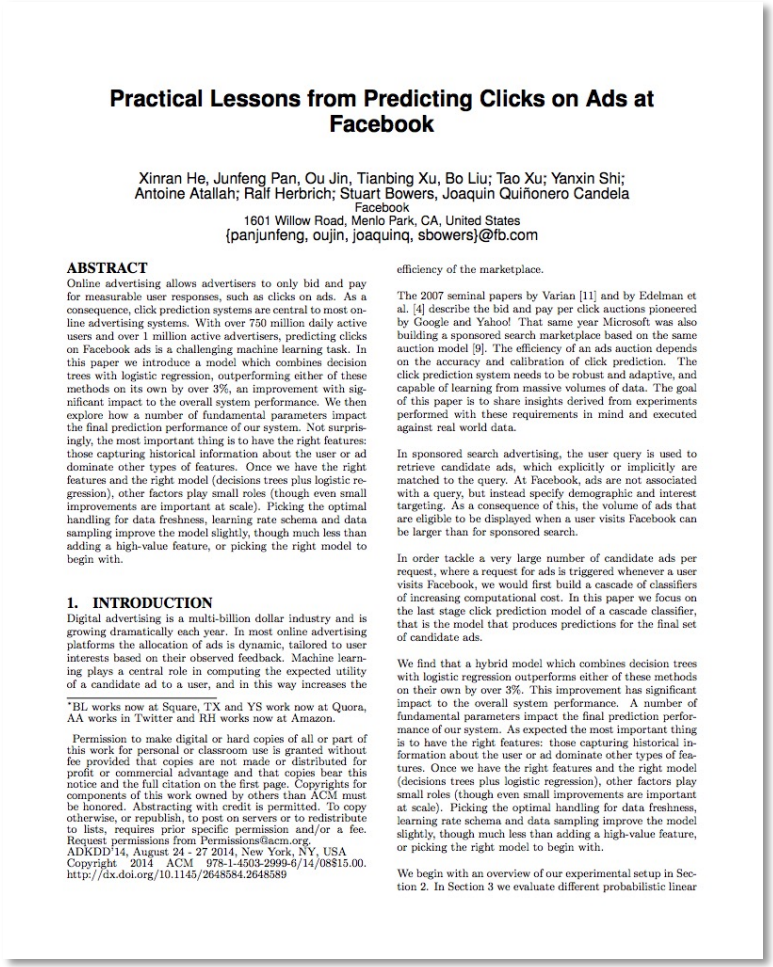
Logistic Regression



Probabilistic classification:

$$y = sig\ (h_\Theta(x)), y \in]0,1[$$

Logistic Regression is Powerful



Practical Lessons from Predicting Clicks on Ads at Facebook

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu; Tao Xu; Yanxin Shi;
Antoine Atallah; Ralf Herbrich; Stuart Bowers, Joaquin Quiñonero Candela
Facebook
1601 Willow Road, Menlo Park, CA, United States
{panjunfeng, oujin, joaquinq, sbowers}@fb.com

ABSTRACT

Online advertising allows advertisers to only bid and pay for measurable user responses, such as clicks on ads. As a consequence, click prediction systems are central to most online advertising systems. With over 750 million daily active users and over 1 million active advertisers, predicting clicks on Facebook ads is a challenging machine learning task. In this paper we introduce a model which combines decision trees with logistic regression, outperforming either of these methods on its own by over 3%, an improvement with significant impact to the overall system performance. We then explore how a number of fundamental parameters impact the final prediction performance of our system. Not surprisingly, the most important thing is to have the right features: those capturing historical information about the user or ad dominate other types of features. Once we have the right features and the right model (decisions trees plus logistic regression), other factors play small roles (though even small improvements are important at scale). Picking the optimal handling for data freshness, learning rate schema and data sampling improve the model slightly, though much less than adding a high-value feature, or picking the right model to begin with.

1. INTRODUCTION

Digital advertising is a multi-billion dollar industry and is growing dramatically each year. In most online advertising platforms the allocation of ads is dynamic, tailored to user interests based on their observed feedback. Machine learning plays a central role in computing the expected utility of a candidate ad to a user, and in this way increases the efficiency of the marketplace.

The 2007 seminal papers by Varian [11] and by Edelman et al. [4] describe the bid and pay per click auctions pioneered by Google and Yahoo! That same year Microsoft was also building a sponsored search marketplace based on the same auction model [9]. The efficiency of an ads auction depends on the accuracy and calibration of click prediction. The click prediction system needs to be robust and adaptive, and capable of learning from massive volumes of data. The goal of this paper is to share insights derived from experiments performed with these requirements in mind and executed against real world data.

In sponsored search advertising, the user query is used to retrieve candidate ads, which explicitly or implicitly are matched to the query. At Facebook, ads are not associated with a query, but instead specify demographic and interest targeting. As a consequence of this, the volume of ads that are eligible to be displayed when a user visits Facebook can be larger than for sponsored search.

In order tackle a very large number of candidate ads per request, where a request for ads is triggered whenever a user visits Facebook, we would first build a cascade of classifiers of increasing computational cost. In this paper we focus on the last stage click prediction model of a cascade classifier, that is the model that produces predictions for the final set of candidate ads.

We find that a hybrid model which combines decision trees with logistic regression outperforms either of these methods on their own by over 3%. This improvement has significant impact to the overall system performance. A number of fundamental parameters impact the final prediction performance of our system. As expected the most important thing is to have the right features: those capturing historical information about the user or ad dominate other types of features. Once we have the right features and the right model (decisions trees plus logistic regression), other factors play small roles (though even small improvements are important at scale). Picking the optimal handling for data freshness, learning rate schema and data sampling improve the model slightly, though much less than adding a high-value feature, or picking the right model to begin with.

We begin with an overview of our experimental setup in Section 2. In Section 3 we evaluate different probabilistic linear

*BL works now at Square, TX and YS work now at Quora, AA works in Twitter and RH works now at Amazon.

ADKDD'14, August 24 - 27 2014, New York, NY, USA

http://dx.doi.org/10.1145/2648584.2648589

Stochastic Gradient Descent (SGD) algorithm [2] applied to sparse linear classifiers. After feature transformation, an ad impression is given in terms of a structured vector $\boldsymbol{x} = (\boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_n})$ where $\boldsymbol{e}_i$ is the i-th unit vector and $i_1, \ldots, i_n$ are the values of the n categorical input features. In the training phase, we also assume that we are given a binary label $y \in \{+1, -1\}$ indicating a click or no-click.

Given a labeled ad impression $(\boldsymbol{x}, y)$, let us denote the linear combination of active weights as

$$s(y, \boldsymbol{x}, \boldsymbol{w}) = y \cdot \boldsymbol{w}^\top \boldsymbol{x} = y \sum_{j=1}^{n} w_{j,i_j} \,, \tag{2}$$

where $\boldsymbol{w}$ is the *weight* vector of the linear click score.

In the state of the art Bayesian online learning scheme for probit regression (BOPR) described in [7] the likelihood and prior are given by

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \Phi\left(\frac{s(y, \boldsymbol{x}, \boldsymbol{w})}{\beta}\right),$$
$$p(\boldsymbol{w}) = \prod_{k=1}^{N} N(w_k; \mu_k, \sigma_k^2) \,,$$

where $\Phi(t)$ is the cumulative density function of standard normal distribution and $N(t)$ is the density function of the standard normal distribution. The online training is achieved through expectation propagation with moment matching. The resulting model consists of the mean and the variance of the approximate posterior distribution of weight vector $\boldsymbol{w}$. The inference in the BOPR algorithm is to compute $p(\boldsymbol{w}|y, \boldsymbol{x})$ and project it back to the closest factorizing Gaussian approximation of $p(\boldsymbol{w})$. Thus, the update algorithm can be solely expressed in terms of update equations for all means and variances of the non-zero components $\boldsymbol{x}$ (see [7]):

$$\mu_{i_j} \leftarrow \mu_{i_j} + y \cdot \frac{\sigma_{i_j}^2}{\Sigma} \cdot v\left(\frac{s(y, \boldsymbol{x}, \boldsymbol{\mu})}{\Sigma}\right), \tag{3}$$

$$\sigma_{i_j}^2 \leftarrow \sigma_{i_j}^2 \cdot \left[1 - \frac{\sigma_{i_j}^2}{\Sigma^2} \cdot w\left(\frac{s(y, \boldsymbol{x}, \boldsymbol{\mu})}{\Sigma}\right)\right], \tag{4}$$

$$\Sigma^2 = \beta^2 + \sum_{j=1}^{n} \sigma_{i_j}^2 \,. \tag{5}$$

Here, the corrector functions v and w are given by $v(t) := N(t)/\Phi(t)$ and $w(t) := v(t) \cdot [v(t) + t]$. This inference can be viewed as an SGD scheme on the belief vectors $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

We compare BOPR to an SGD of the likelihood function

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \text{sigmoid}(s(y, \boldsymbol{x}, \boldsymbol{w})) \,,$$

where $\text{sigmoid}(t) = \exp(t)/(1 + \exp(t))$. The resulting algorithm is often called *Logistic Regression* (LR). The inference in this model is computing the derivative of the log-likelihood and walk a per-coordinate depending step size in the direction of this gradient:

$$w_{i_j} \leftarrow w_{i_j} + y \cdot \eta_{i_j} \cdot g(s(y, \boldsymbol{x}, \boldsymbol{w})) \,, \tag{6}$$

where g is the log-likelihood gradient for all non-zero components and given by $g(s) := [y(y+1)/2 - y \cdot \text{sigmoid}(s)]$. Note that (3) can be seen as a per-coordinate gradient descent like (6) on the mean vector $\boldsymbol{\mu}$ where the step-size η_{i_j} is automatically controlled by the belief uncertainty $\boldsymbol{\sigma}$. In Subsection 3.3 we will present various step-size functions η and compare to BOPR.

Both SGD-based LR and BOPR described above are stream learners as they adapt to training data one by one.

3.1 Decision tree feature transforms

There are two simple ways to transform the input features of a linear classifier in order to improve its accuracy. For continuous features, a simple trick for learning non-linear transformations is to bin the feature and treat the bin index as a categorical feature. The linear classifier effectively learns a piece-wise constant non-linear map for the feature. It is important to learn useful bin boundaries, and there are many information maximizing ways to do this.

The second simple but effective transformation consists in building tuple input features. For categorical features, the brute force approach consists in taking the Cartesian product, i.e. in creating a new categorical feature that takes as values all possible values of the original features. Not all combinations are useful, and those that are not can be pruned out. If the input features are continuous, one can do joint binning, using for example a k-d tree.

We found that boosted decision trees are a powerful and very convenient way to implement non-linear and tuple transformations of the kind we just described. We treat each individual tree as a categorical feature that takes as value the index of the leaf an instance ends up falling in. We use 1-of-K coding of this type of features. For example, consider the boosted tree model in Figure 1 with 2 subtrees, where the first subtree has 3 leafs and the second 2 leafs. If an instance ends up in leaf 2 in the first subtree and leaf 1 in second subtree, the overall input to the linear classifier will be the binary vector $[0, 1, 0, 1, 0]$, where the first 3 entries correspond to the leaves of the first subtree and last 2 to those of the second subtree. The boosted decision trees we use follow the Gradient Boosting Machine (GBM) [5], where the classic L_2-TreeBoost algorithm is used. In each learning iteration, a new tree is created to model the residual of previous trees. We can understand boosted decision tree based transformation as a supervised feature encoding that converts a real-valued vector into a compact binary-valued vector. A traversal from root node to a leaf node represents a rule on certain features. Fitting a linear classifier on the binary vector is essentially learning weights for the set of rules. Boosted decision trees are trained in a batch manner.

We carry out experiments to show the effect of including tree features as inputs to the linear model. In this experiment we compare two logistic regression models, one with tree feature transforms and the other with plain (non-transformed) features. We also use a boosted decision tree model only for comparison. Table 1 shows the results.

Tree feature transformations help decrease Normalized Entropy by more more than 3.4% relative to the Normalized Entropy of the model with no tree transforms. This is a very significant relative improvement. For reference, a typical feature engineering experiment will shave off a couple of tens of a percent of relative NE. It is interesting to see

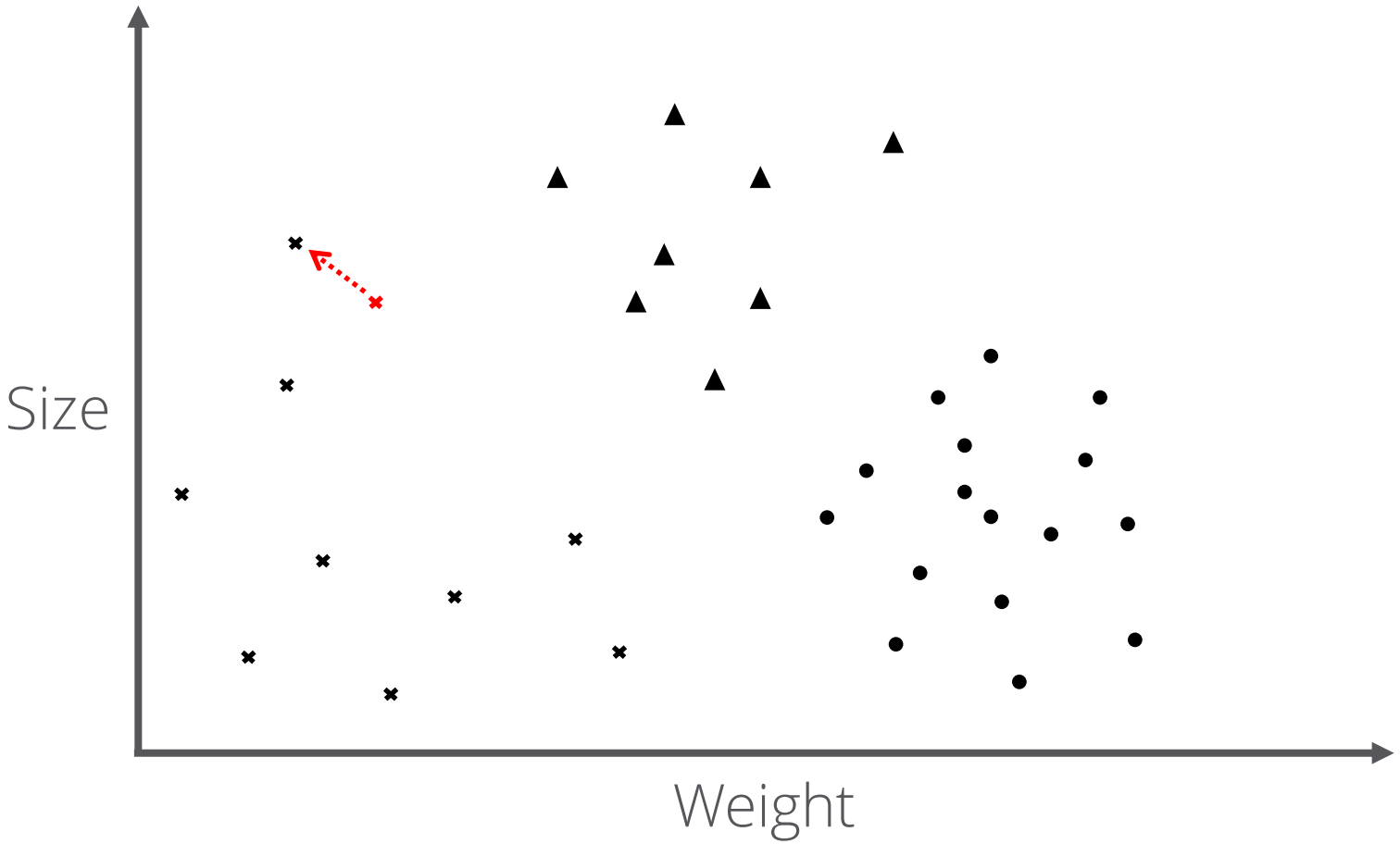
https://research.fb.com/publications/practical-lessons-from-predicting-clicks-on-ads-at-facebook/

Discussion Logistic Regression

- **Pros:**
  - Implementation: Easy to implement and to apply
  - Probabilistic: Returns the probability of an object whether it is in a certain class
  - Computation: Simple and quick training phase
  - Insights: Returns easy-to-understand parameter-based model, no black-box

- **Cons:**
  - Linearity: Hard to apply to non linear problems
  - Overfitting: Training data has to be balanced and well chosen

Nearest Neighbor



We instant-based classify a new object based on it´s nearest neighbor

Nearest Neighbor - Instance based learning

- No training and test phase
  - No generated model
- Store labeled training data
  - Points in a metric space
- Process training data when a new object should be classified
  - „lazy evaluation“

- Tradeoff between time and complexity
  - Hard to build a model based on a large dataset but it is easy to apply
  - Easy to save the large dataset but hard to search (high latencies)

Nearest Neighbor Flavors

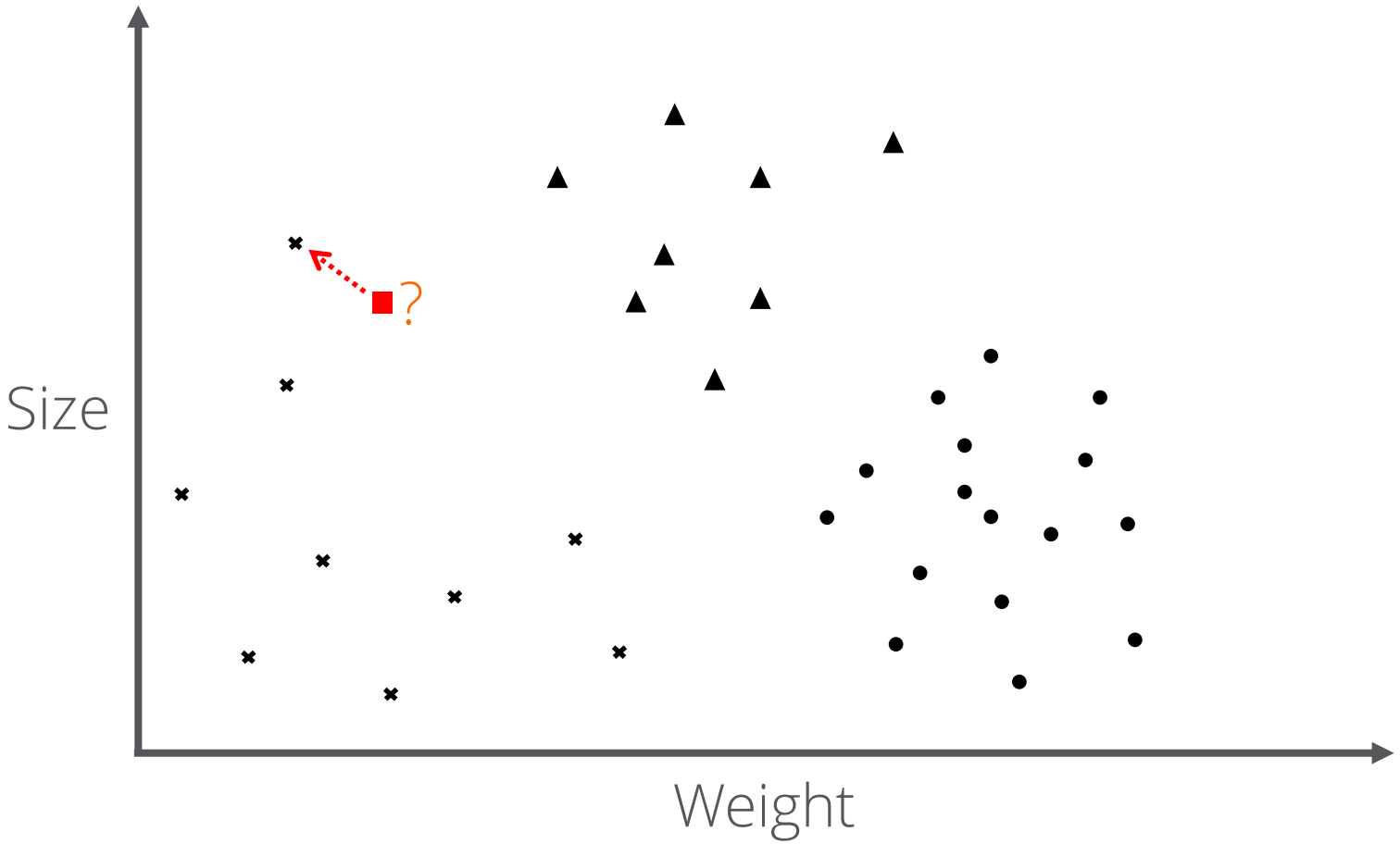
1. **NN Classifier**
   Only the nearest neighbor

2. **k-NN Classifier**
   k nearest neighbors (k>1)

3. **Weighted k-NN Classifier**
   The weighted distances to the k nearest neighbors

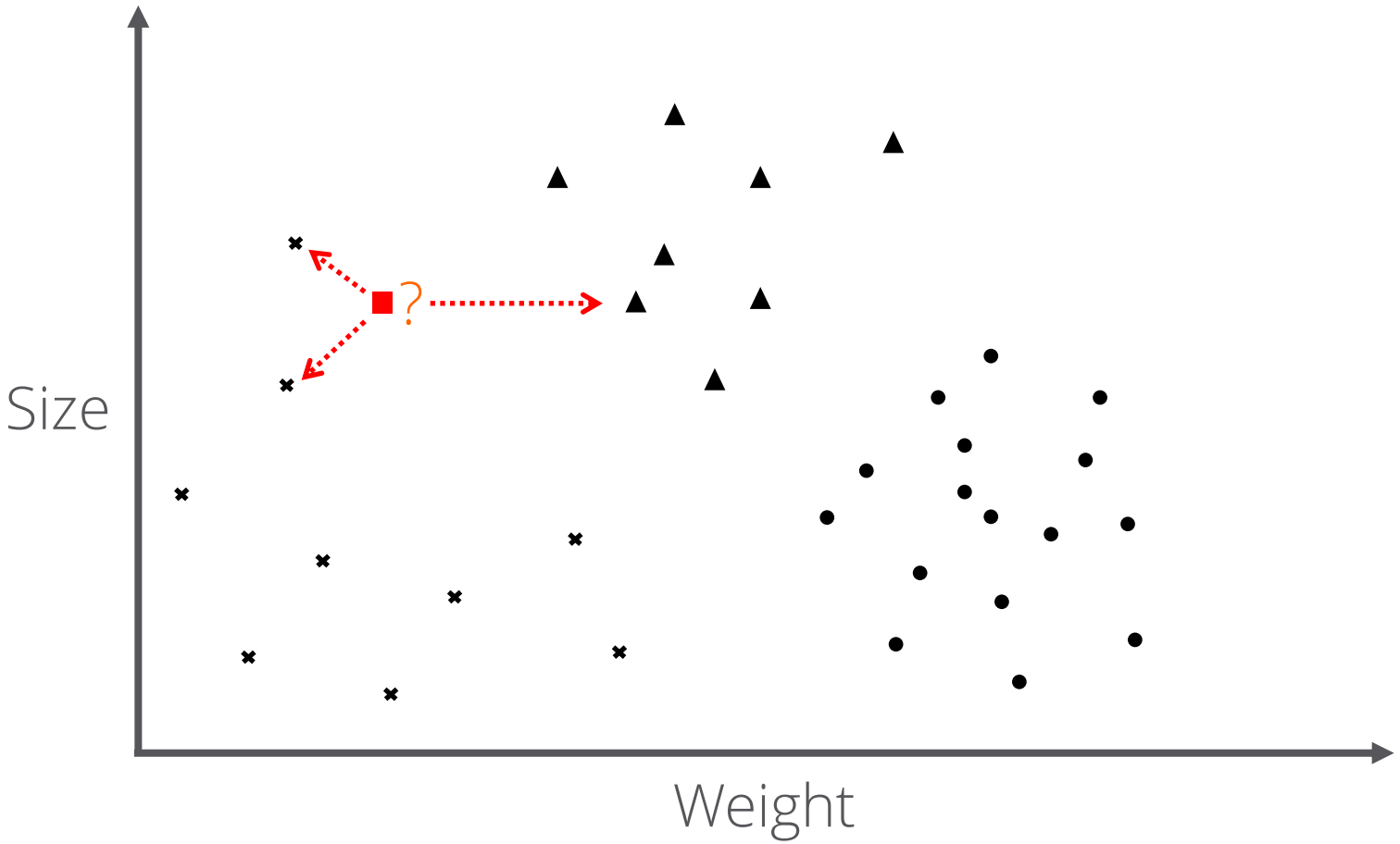
4. **Mean-based NN Classifier**
   The closest mean position of a class

Nearest Neighbor Flavors: NN



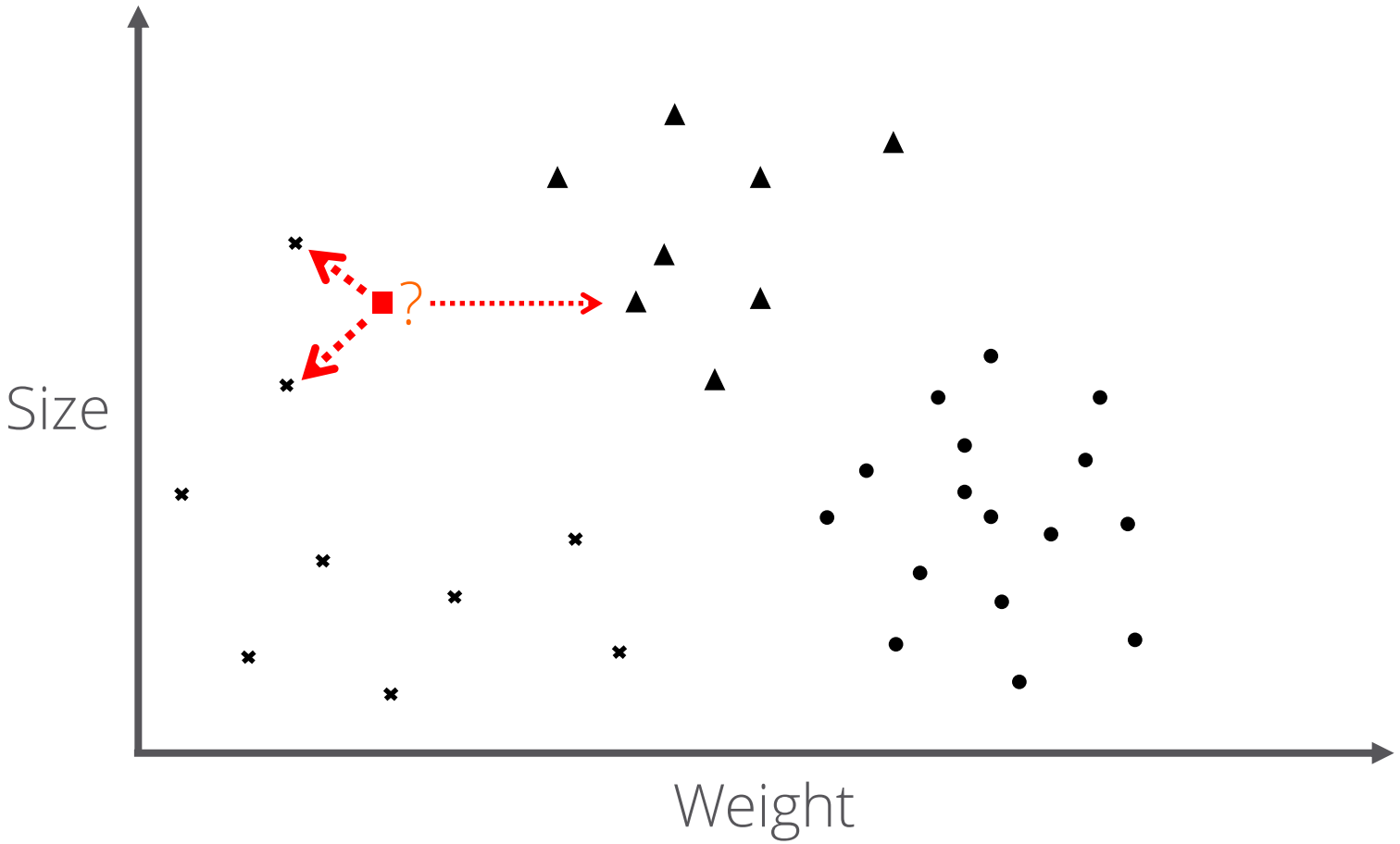
NN Classifier: Take only the nearest neighbor into consideration.

Nearest Neighbor Flavors: k-NN



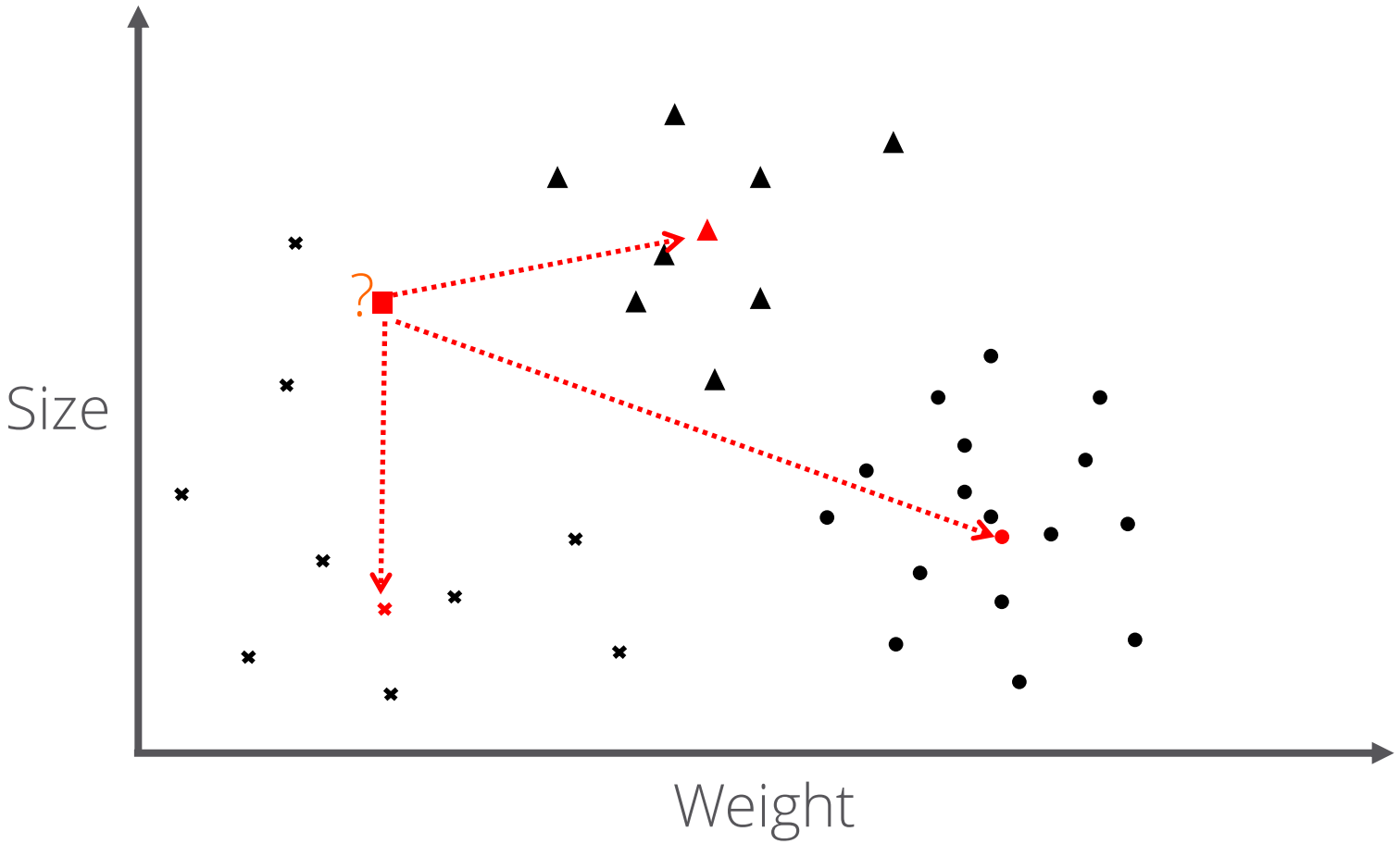
k-NN Classifier: Take k nearest neighbors (k > 1) into consideration.

Nearest Neighbor Flavors: Weighted k-NN



Weighted k-NN Classifier: Use weights for the classes of the K nearest neighbors

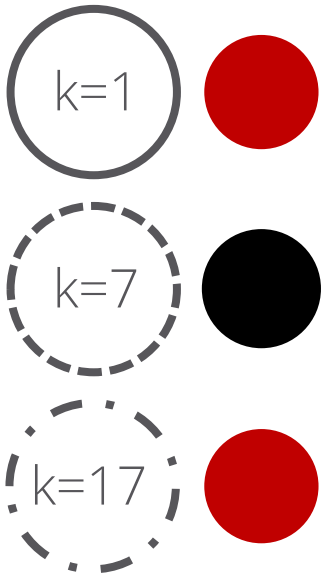
Nearest Neighbor Flavors: Mean-based NN

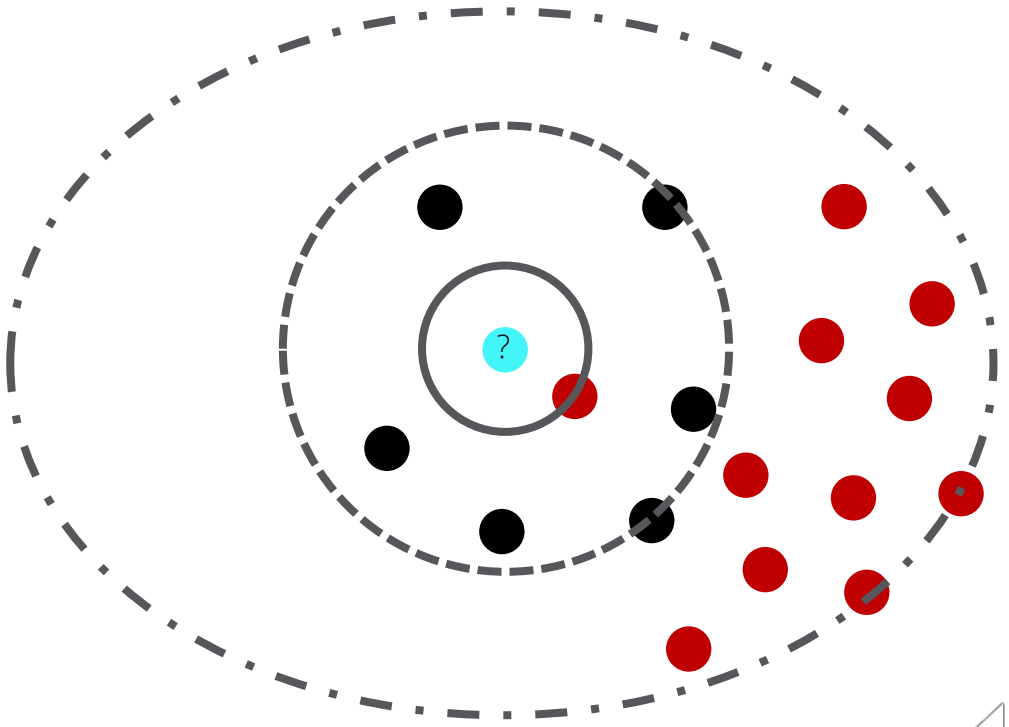


Mean-based NN Classifier: Consider the closest mean position of a class

k-NN Classifier

- How to choose k?
  - Generalization vs Overfitting
  - Large k: Many objects from different classes
  - Small k: Sensitivity against outliers
  - Practice: 1 << k < 10





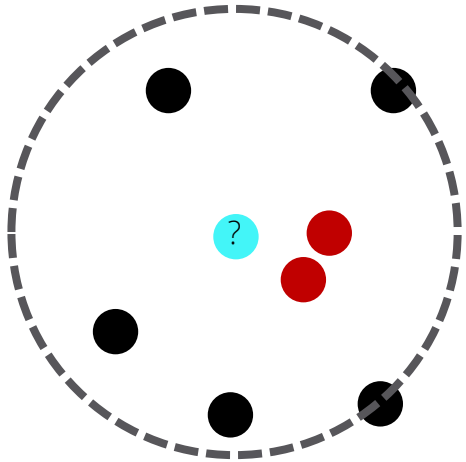
Weighted k-NN Classifier

- How to weight the neighbors?
  - Frequency of the neighbors' class
  - Distance to the neighbors

Normal

Weighted (Frequency)

Weighted (Distance)



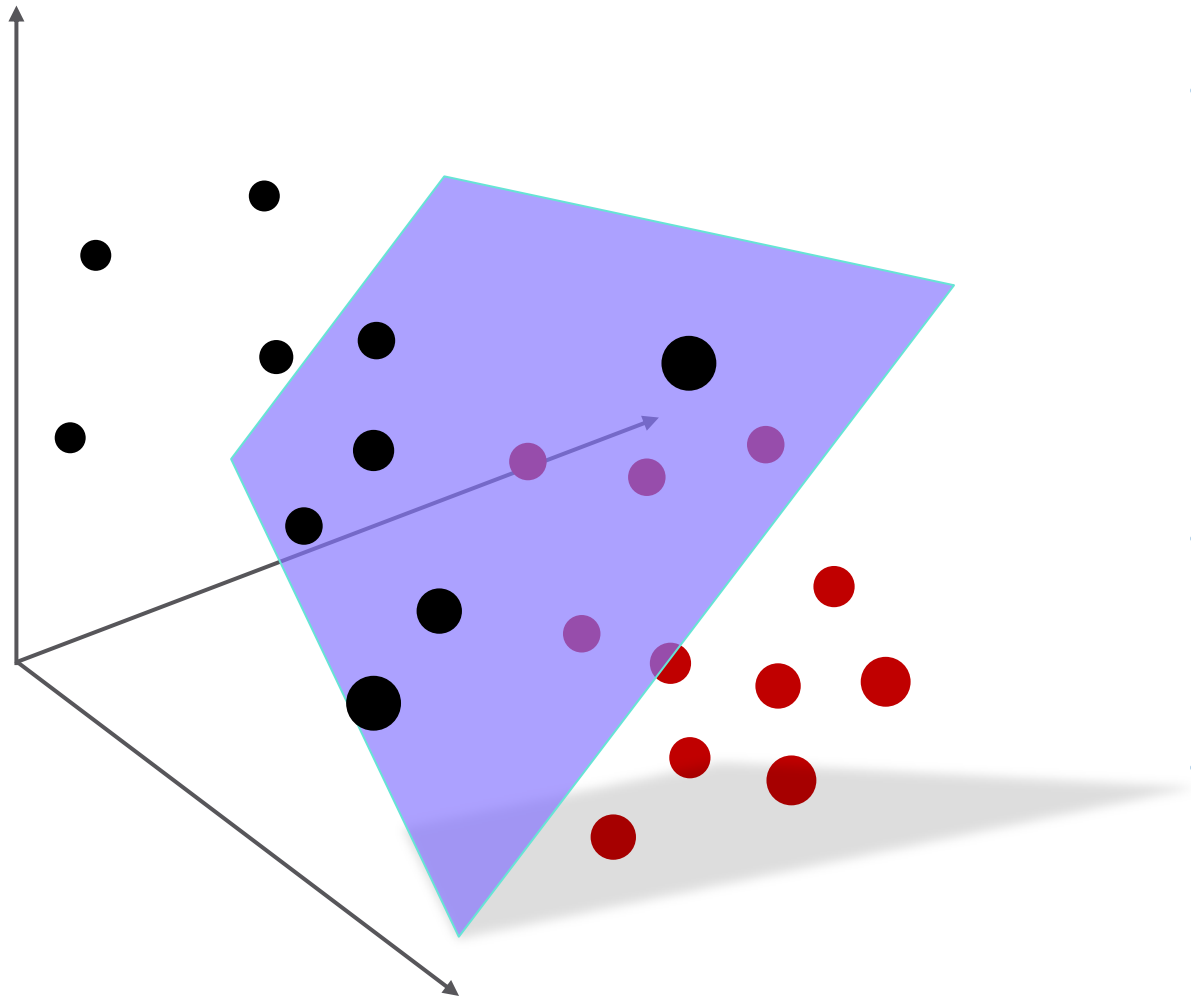
Discussion of NN Classifiers

- **Pros:**
  - Applicability: Easy to calculate distances
  - Accuracy: Great results for many applications
  - Incremental: Easy adoption of new training data
  - Robust: Handles noise by averaging (k-NN)

- **Contras:**
  - Efficiency: Processing complexity increases with training data
    - Counteracting by setting up an index structure (requires training phase)
  - Dimensionality: Not every dimension may be relevant / curse of dimensionality
    - Weight dimensions (scale axes)

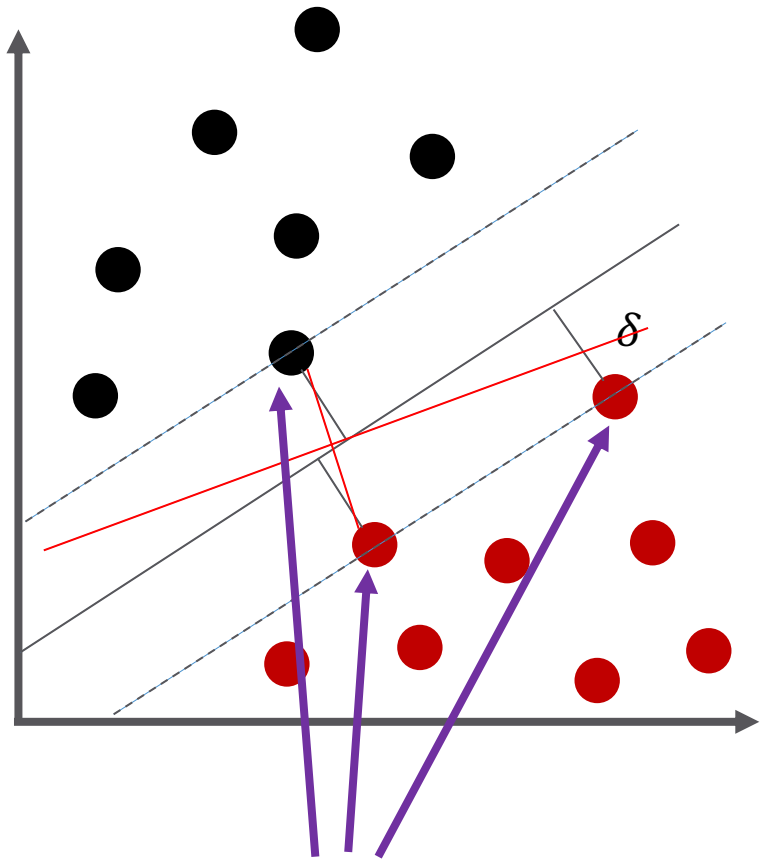
- **Not Pro not Con:**
  - Does not yield explicit knowledge about classes

Support Vector Machines (SVM)



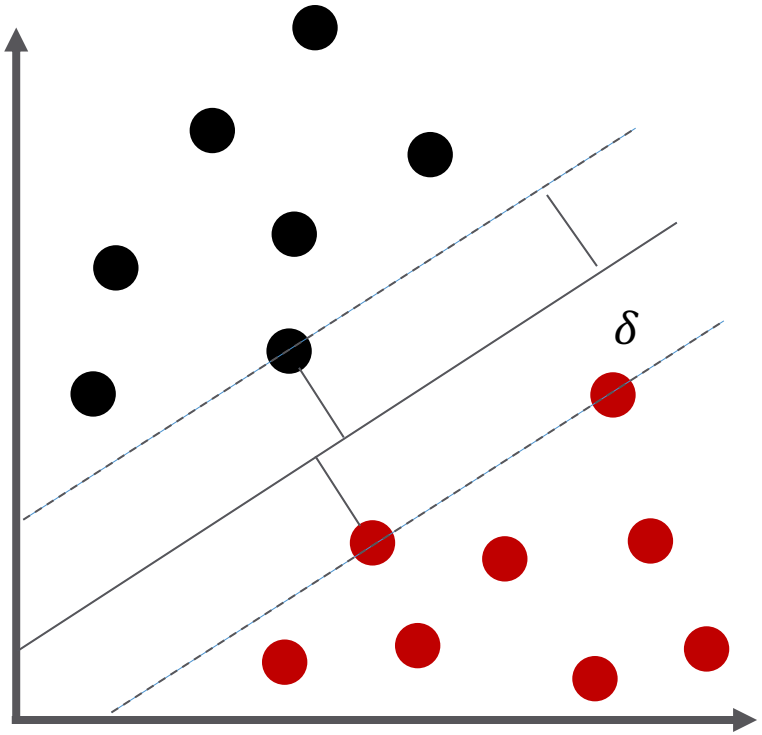
- Linear separation
  - Objects in R^d
  - Two classes
  - Hyper plane separates both classes
- Training
  - Compute hyper plane
- Classification
  - Distance to hyper plane

SVM - Maximum Margin Hyperplane (MMH)



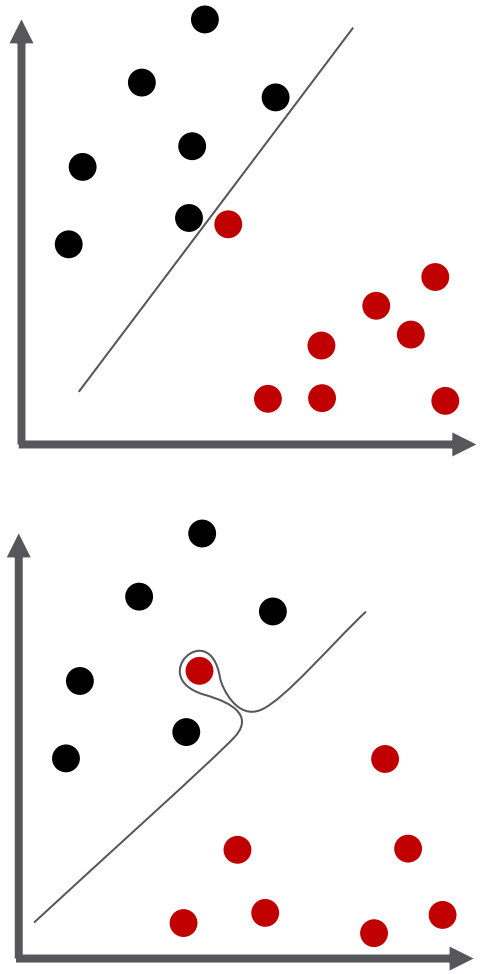
- Max. distance to hyper plane
  - At least δ (margin)
- High generalization
  - Maximum of stability
- Support vectors
  - Only depending on objects with distance δ

SVM – Formal Definition



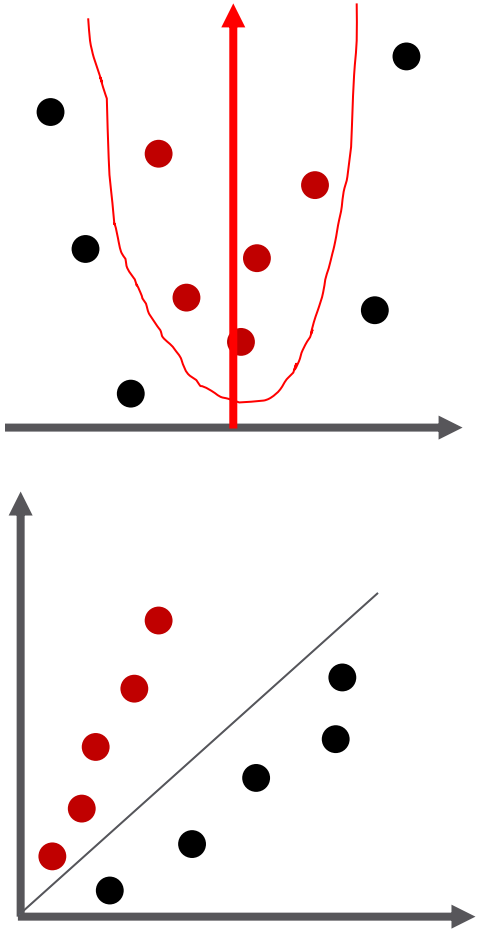
- Training data: $(x_1, y_1) \dots (x_n, y_n)$
  with $x \in R^d, y \in \{-1,1\}$
- Hyper plane: $w \cdot x - b = 0$
  with *w* normal vector, $\frac{b}{\|w\|}$ offset from origin,
- Margin: $\delta = \frac{1}{\|w\|}$
- Training: Minimize $\|w\|$
  with $y_i(w \cdot x_i - b) \geq 1$ for i=1… n
- Classification:
  if $(w \cdot x - b) \geq 0$, y=1; else y=-1
  with Data $x \in R^d$

SVM - Soft Margin



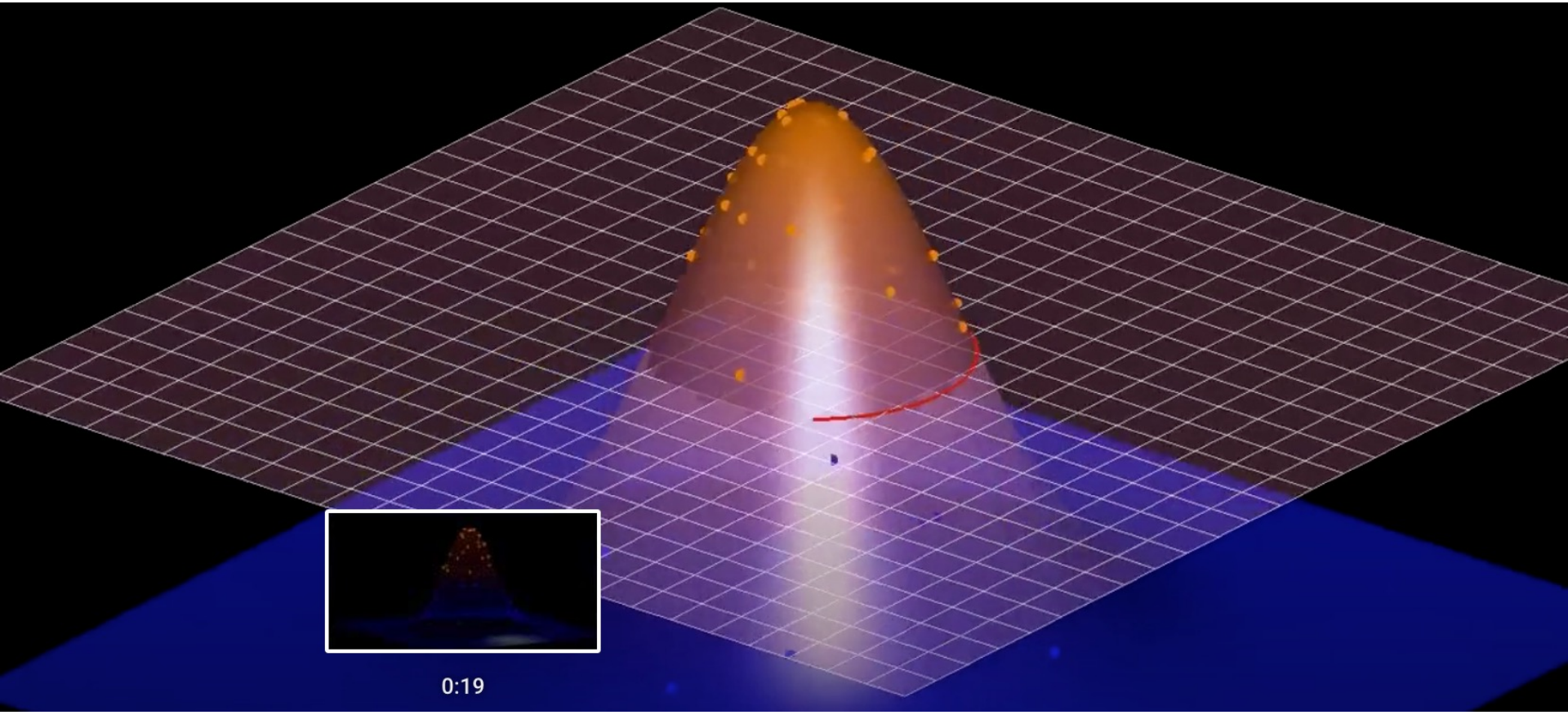
- Linear separation
  - Not always possible
  - Not always optimal
- Tradeoff between Error and Margin
  - Allow classification error to maximize margin

SVM - Space Transformation



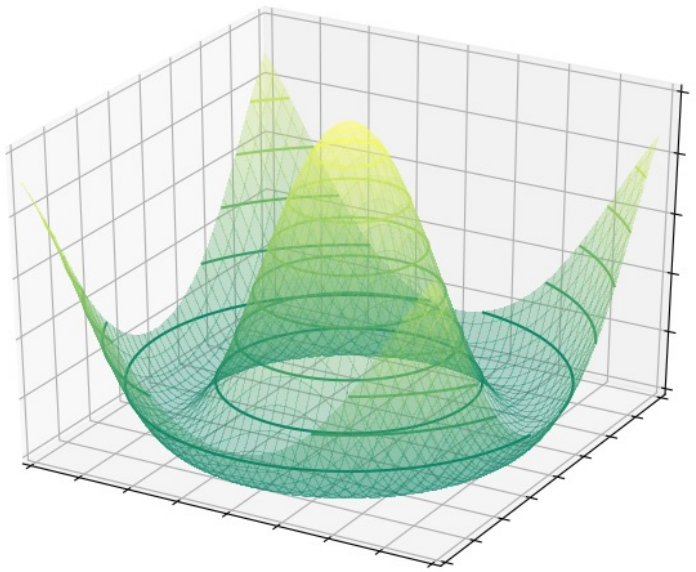
- Non Linear data
  - Too many errors with Soft Margin
- Use higher dimensional space
  - Increase dimensions until linear separation is possible
  - Transform Hyper plane back to lower dimensions
  - Hyper plane becomes non-linear
- Example: Quadratic transformation
  - Hyper plane becomes polynomial of degree 2

SVM – Kernel Machines Visualization



https://www.youtube.com/watch?v=9NrALgHFwTo

SVM - Kernel Machines



- Space transformations
  - Lower to higher dimensions
  - Computational complex
- Hyperplane transformation
  - Higher to lower dimension
  - Feasibility not guaranteed
  - Computational complex
- Kernel
  - Computational elegant
  - Calculate dot product without full space transformation

SVM - Kernel Machines

Replace the scalar product with a non-linear kernel function

1. **Polynominal:**
   $k(x_i, x_j) = (x_i \cdot x_j)^d$

2. **Gaussian radial bias function (RBF):**
   $k(x_i, x_j) = exp\ (-\lambda\ ||x_i - x_j||^2)\ for\ \lambda>0$

3. **Linear, Sigmoid, Hyperbolic ...**

SVM – Kernel Examples

With space transformation:

1. $f:R^3 \rightarrow R^9$

   $f(x)=(x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$

   $x=(1,2,3), y=(4,5,6)$

   $f(x)=(1,2,3,2,4,6,3,6,9)$

   $f(y)=(16,20,24,20,25,30,24,30,36)$

   $f(x) \cdot f(y)=16+40+72+40+100+180+72+180+324=1024$

With kernel trick:

1. $k(x,y) = (x \cdot y)^2$
2. $k(x,y)=(4+10+18)^2=32^2=1024$

→ No transformation to R^9 required

Discussion SVM

- **Pros:**
  - Accuracy: High classification rate
  - Effective: Even when number of dimensions > number of samples
  - Robust: Low tendency to overfitting
  - Compact Models: "Plane in Space"
  - Versatile: Different Kernel Function

- **Cons:**
  - Efficiency: Long training phase
  - Complexity: High implementation effort
  - Black-Box: Hard to understand models

Agenda

01 Introduction (Training / Validation)

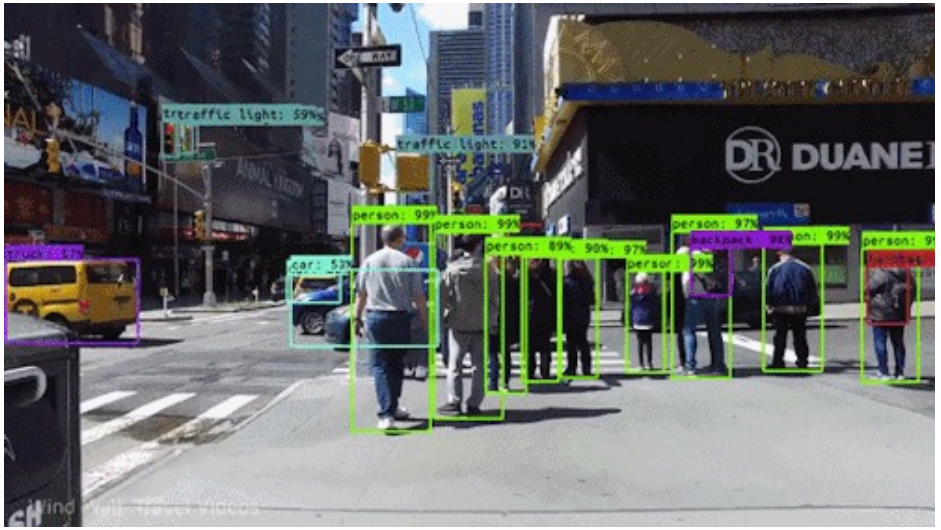
02 Methods
(Logistic Regression, Nearest Neighbors, Support Vector Machines)

03 **Applications**

04 Summary

Classification for Automotive Technology

- Example: **Perception**
  - Camera outputs pixel array
  - Classification adds value to each pixel
  - Pixel segmentation
  - Object detection
  - Object tracking



Vehicle Detection and Tracking

Vehicle Detection and Tracking

1. Get training data
2. Extract features from images
3. Generate a model based on the features
4. Take one video frame and classify the features of the sub-images
5. Merge classified areas

Training Data

- Required label: „car“ or „no car“

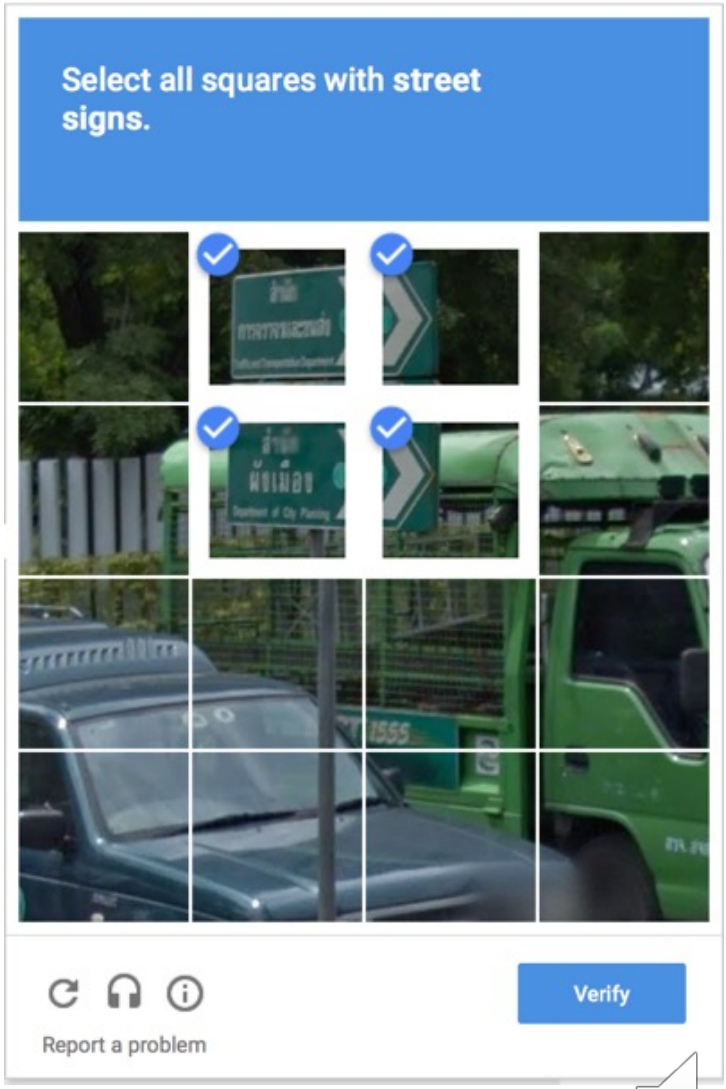
- Required Images:
  - Same format used for classification
  - Represenative for what we expect to find in the videostream
  - 8000 images (90 % training and 10 % test)

www.cvlibs.net/datasets/kitti/

Training Data

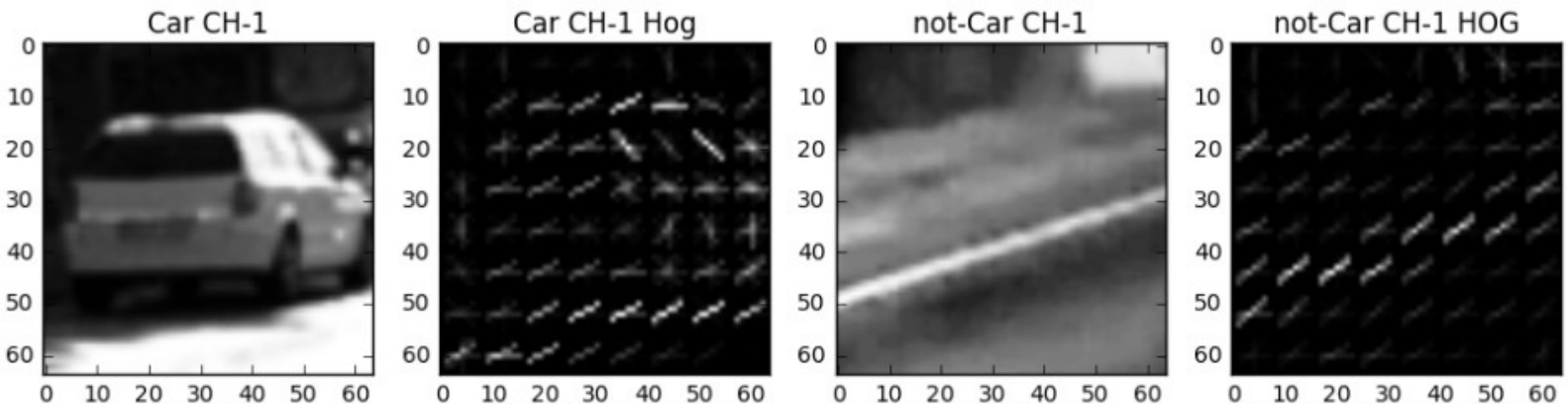
- How to get labeled data?
  - Label data by yourself
  - Pay someone else to label your data
  - Let other label your data for free

- Collection of labeled data
  - Digits: MNIST
    - http://yann.lecun.com/exdb/mnist/
    - 70k images
  - Cars: KITTI
    - www.cvlibs.net/datasets/kitti/
    - 80k images



Feature Extraction

- Histogram of Oriented Gradients (HOG)
  - compressed & encoded version of the image



https://eu.udacity.com/course/self-driving-car-engineer-nanodegree--nd013

Build SVM Classifier

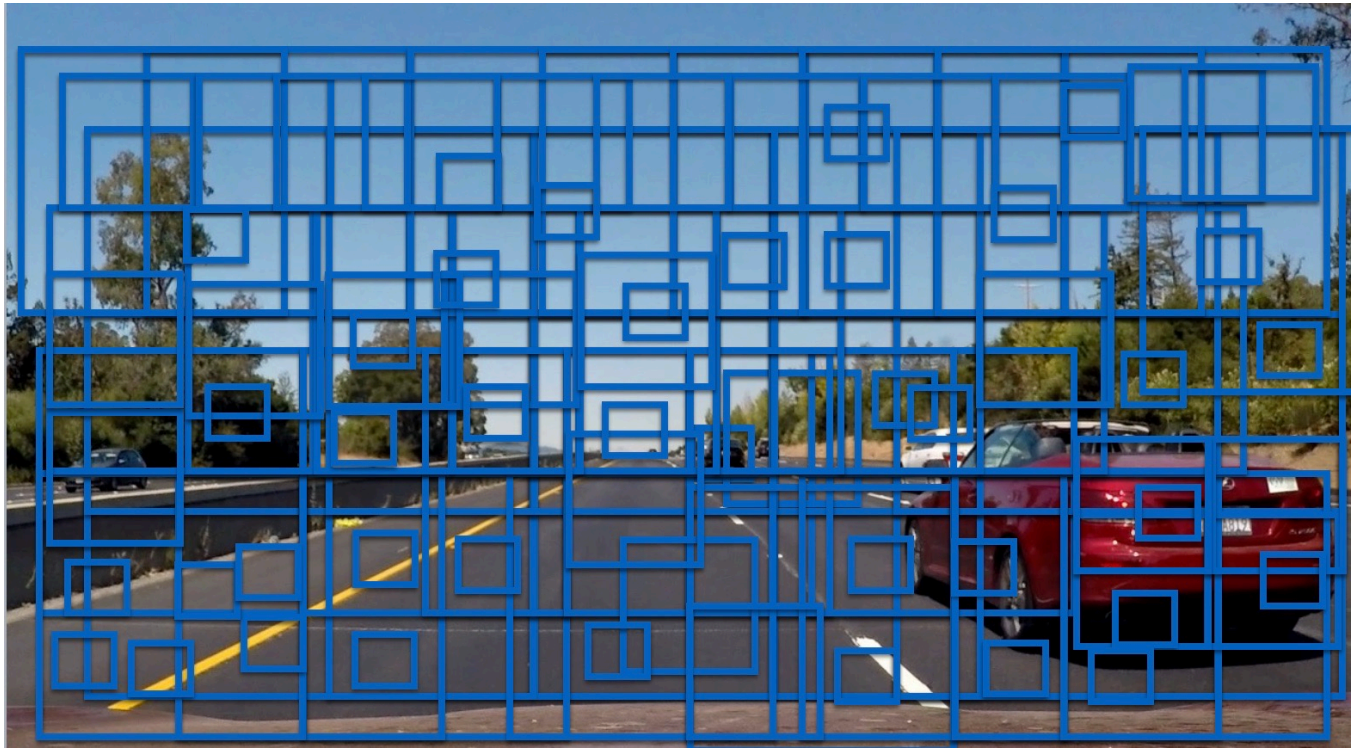
- Machine learning libraries (python)
  - scikit-learn (http://scikit-learn.org/)


```
>>> from sklearn import svm
>>> clf = svm.SVC()
>>> clf.fit(training_features, training_labels)
>>> clf.score(test_features, test_labels)
>>> clf.predict(new_feature)
```

  - Training: 1.44 Seconds
  - Test: Accuracy = 0.9848
  - Prediction: 0 or 1

Classifiy sub-images

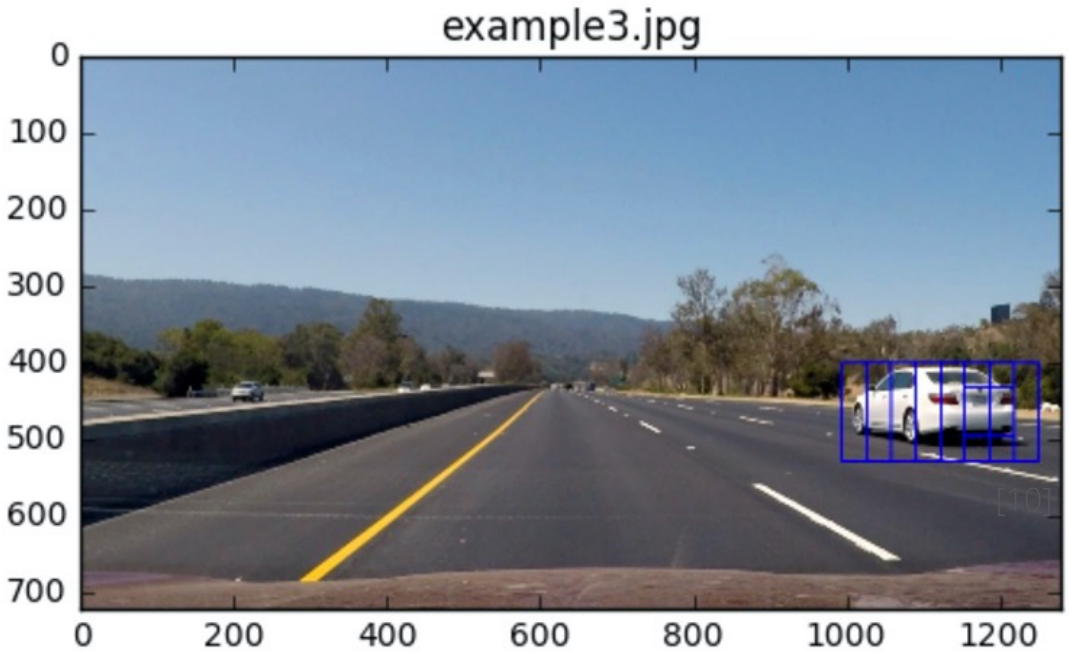
- Produce sub-images of each frame for the classification

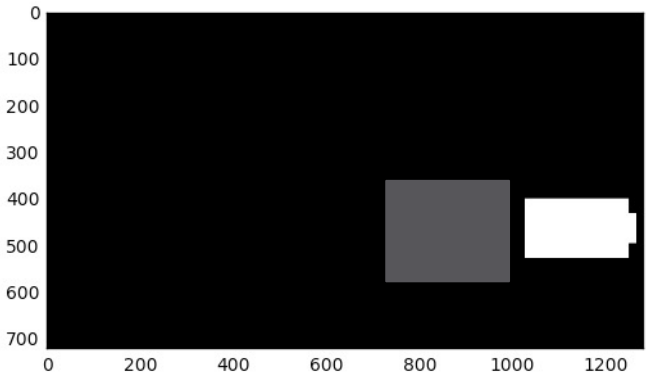


https://eu.udacity.com/course/self-driving-car-engineer-nanodegree--nd013

Merge classified areas

- Merge classes of sub-images





Agenda

01 Introduction (Training / Validation)

02 Methods
(Logistic Regression, Nearest Neighbors, Support Vector Machines)

03 Applications

04 **Summary**

Summary

1. **Classification** a **supervised learning** problem and is about assigning given classes to data.
2. We need labeled data for training and validation (hidden label).
3. We have several criteria to measure the **quality of a classifier**.
4. The concepts of **Logistic regression, Nearest Neighbor and SVM**
5. We can use linear regression together with a sigmoid function as classification method.
6. Nearest Neighbor is an instance based learning method, no training is required.
7. **SVMs** are linear classifier using a maximum margin hyper plane.
8. With the **kernel trick**, SVMs can be used for non linear classification.
9. Classification is very important for the **perception**, eg. in cars.
10. Acquiring lots of **labeled data** is a problem.
11. We have access to good and easy to use **python libraries** for classification.
12. We have to **extract features** from images for the classification.