Anomaly Detection in Machine Learning and Computer Vision

Thomas G. Dietterich, Distinguished Professor (Emeritus)

Irené Tematelewo, Ph.D. Student

Oregon State University, Corvallis, OR USA 97331

Anomaly Detection Use Cases

- **Data Cleaning**
  - Remove corrupted data from the training data
  - Example: Typos in feature values, feature values interchanged, test results from two patients combined
- **Fraud Detection, Cyber Attack, etc.**
  - At training or test time, illegal behavior creates anomalous data
- **Open Category Detection**
  - At test time, the classifier is given an instance of a novel category
  - Example: Self-driving car (trained in Europe) encounters a kangaroo (in Australia)
- **Novel Sub-category Detection**
  - At test time, the classifier is given a new kind of instance for a known category
  - Example: Chihuahua shown to a classifier trained only on Beagle and Golden Retriever
  - Example: New subtype of known disease
- **Out-of-Distribution Detection**
  - At test time, the classifier is given an instance collected in a different way
  - Example: Chest X-Ray classifier trained only on front views is shown a side view
  - Example: Self-driving car trained in clear conditions must operate during rainy conditions

Anomaly Detection

Definition of “anomaly”:

- A data point that is generated by a different process than the process that is generating the “nominal” points
- Examples: sensor failures, fraud, cyber-attack, etc.

Challenges:

- Little or no labeled data
- Anomalies are rare
- Anomalies may not come from a well-defined probability distribution (especially in adversarial settings)
- Nuisance Novelty: Not all anomalies are relevant to the task or use-case
    - Irrelevant features in web site behavior or internet traffic
    - Changes in image background or context

Strategy:

- Because anomalies are rare, the main strategy for detecting them is to look for outliers: points that are far away from most of the data

Technical Approaches

- Let D be our training data
- **Density Estimation Methods**
  - Surprise: $A(x_q) = -\log P_D(x_q)$
  - Model the joint distribution $P_D(x)$ of the input data points $x_1, \ldots \in D$
  - Issues:
    - Vulnerable to nuisance novelty
    - High-dimensional density estimation requires exponential amounts of training data
- **Quantile Methods**
  - Find a smooth function f such that $\{x: f(x) \geq 0\}$ contains $1 - \alpha$ of the training data
  - Anomaly score $A(x) = -f(x)$
    - Based on kernel techniques, so requires a distance metric and a choice of kernel hyperparameters; vulnerable to irrelevant features
- **Distance-Based Methods**
  - Anomaly score $A(x_q) = \min_{x \in D} \|x_q - x\|$
  - Issues:
    - Requires a good distance metric; vulnerable to irrelevant features
- **Projection Methods**
  - Anomaly score $A(x_q) = \frac{1}{K} \sum_k A\left(\Pi_k(x_q)\right)$ where Π_k is a sparse random projection to a lower-dimensional space
  - Can fix the problem of irrelevant features

Approach 1: Density Estimation

- Parametric Models
- Mixture Models
- Kernel Density Estimation
- Noise-Contrastive Estimation
- Flow Methods

What is Density Estimation?

- Given a data set $\{x_1, \ldots, x_N\}$ where $x_i \in \mathbb{R}^d$
- We assume the data have been drawn iid from an unknown probability density: $x_i \sim P(x_i)$
- Goal: Estimate P

- Requirements
  - $P(x) \geq 0 \; \forall x \in \mathbb{R}^d$ must be non-negative everywhere
  - $\int_{x \in \mathbb{R}^d} P(x) dx \; = 1$ must integrate to 1

How to Evaluate a Density Estimator

- Suppose I have computed a density estimator $\hat{P}$ for P. How can I evaluate it?
- A good density estimator should assign high density where P is large and low density where P is low
- Standard metric: the Log Likelihood of a separate test data set

$$\sum_i \log \hat{P}(x_i)$$

Important: <u>Holdout</u> Likelihood

- If we use our training data to construct $\hat{P}$, we cannot use that same data to evaluate $\hat{P}$.
- Solution:
  - Given our initial data $S = x_1, \dots, x_N$
  - Randomly split into S_{train} and S_{test}
  - Compute $\hat{P}$ using S_{train}
  - Evaluate $\hat{P}$ using S_{test}

$$\sum_{x_i \in S_{test}} \log \hat{P}(x_i)$$

Reminder: Densities, Probabilities, Events

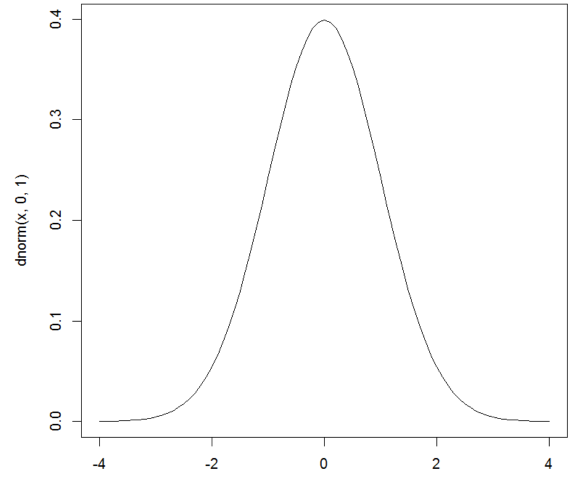
- A density μ is a "measure" over some space $\mathcal{X}$
- A density can be > 1 but must integrate to 1
- An "event" is a subspace (region) $E \subseteq \mathcal{X}$
- The probability of an event is obtained by integration

$$P(E) = \int_{x \in E} \mu(x) dx$$

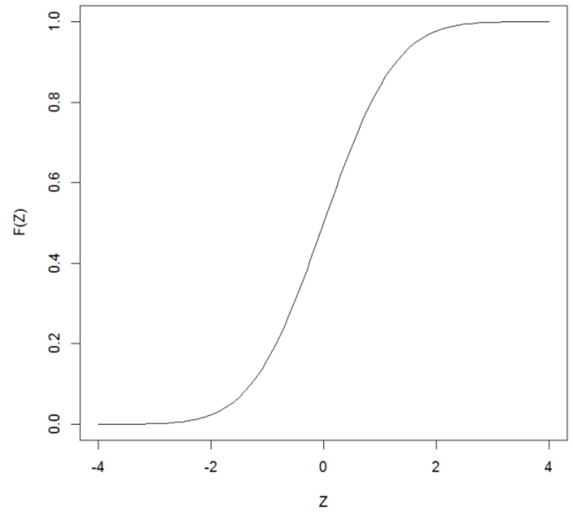
Example: The Gaussian (normal) Distribution

- Normal probability density function (pdf)

$$P(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp - \frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2$$



- Normal cumulative distribution function (cdf)
  - $F(z;\mu,\sigma) =$ probability of the event $[-\infty, z]$
  - $F(z;\mu,\sigma) = \int_{\infty}^{z} P(x;\mu,\sigma)dx$

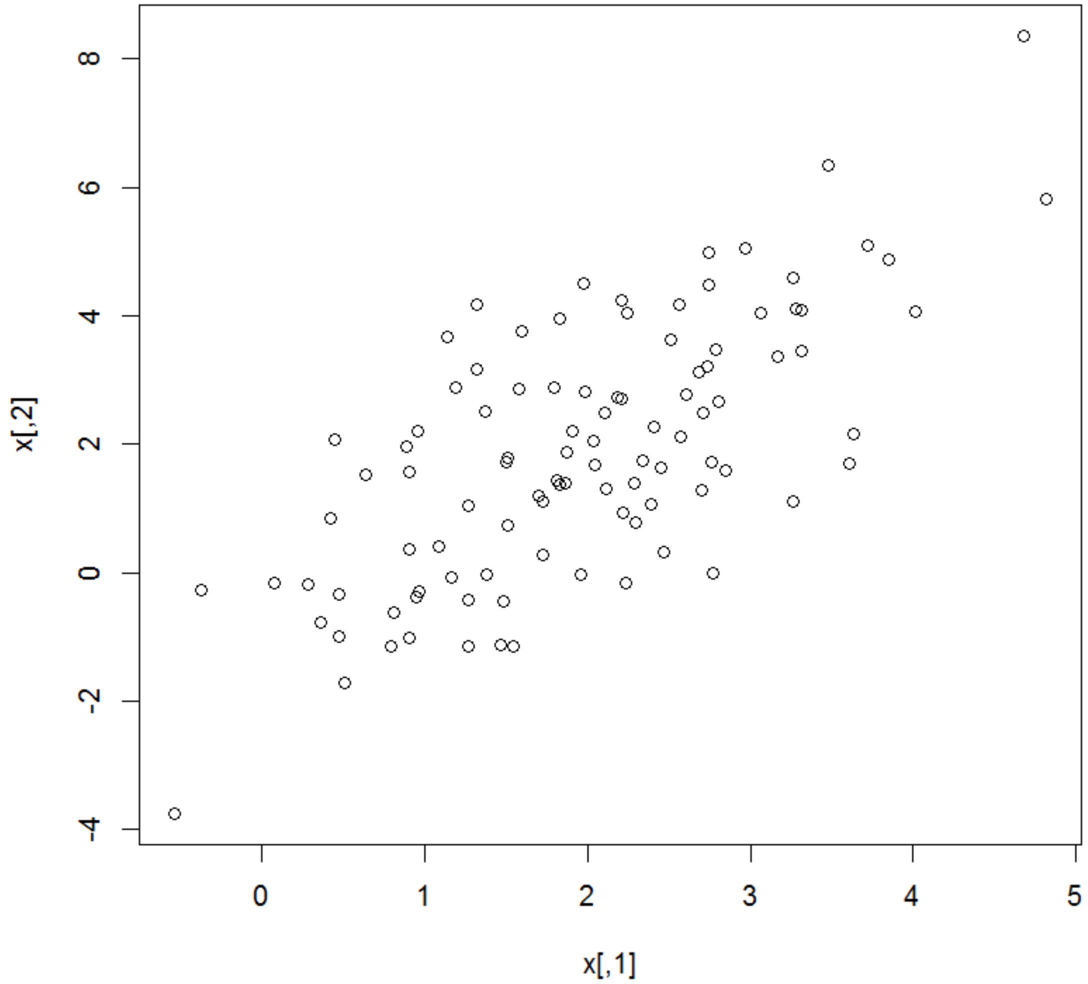


Parametric Density Estimation

- Assume $P(x) = \mathrm{Normal}(x|\mu, \Sigma)$ is the multivariate Gaussian distribution
- $P(x) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp -\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)$
- Fit by computing the first and second moments:
  - $\hat{\mu} = \frac{1}{N}\sum_i x_i$ mean
  - $\hat{\Sigma} = \frac{1}{N}\sum_i (x_i - \hat{\mu})(x_i - \hat{\mu})^\top$ covariance matrix

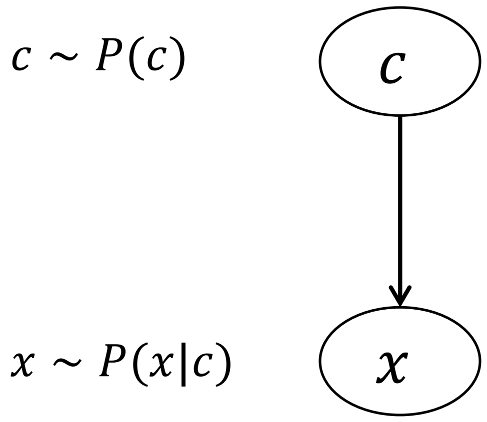
Example

- Sample 100 points from multivariate Gaussian with $\mu = (2,2)$ and $\Sigma = \begin{bmatrix} 1 & 1.5 \\ 1.5 & 4 \end{bmatrix}$
- Estimates:
  - $\hat{\mu} = (1.968731, 1.894511)$
  - $\hat{\Sigma} = \begin{bmatrix} 1.081423 & 1.462467 \\ 1.462467 & 4.000821 \end{bmatrix}$



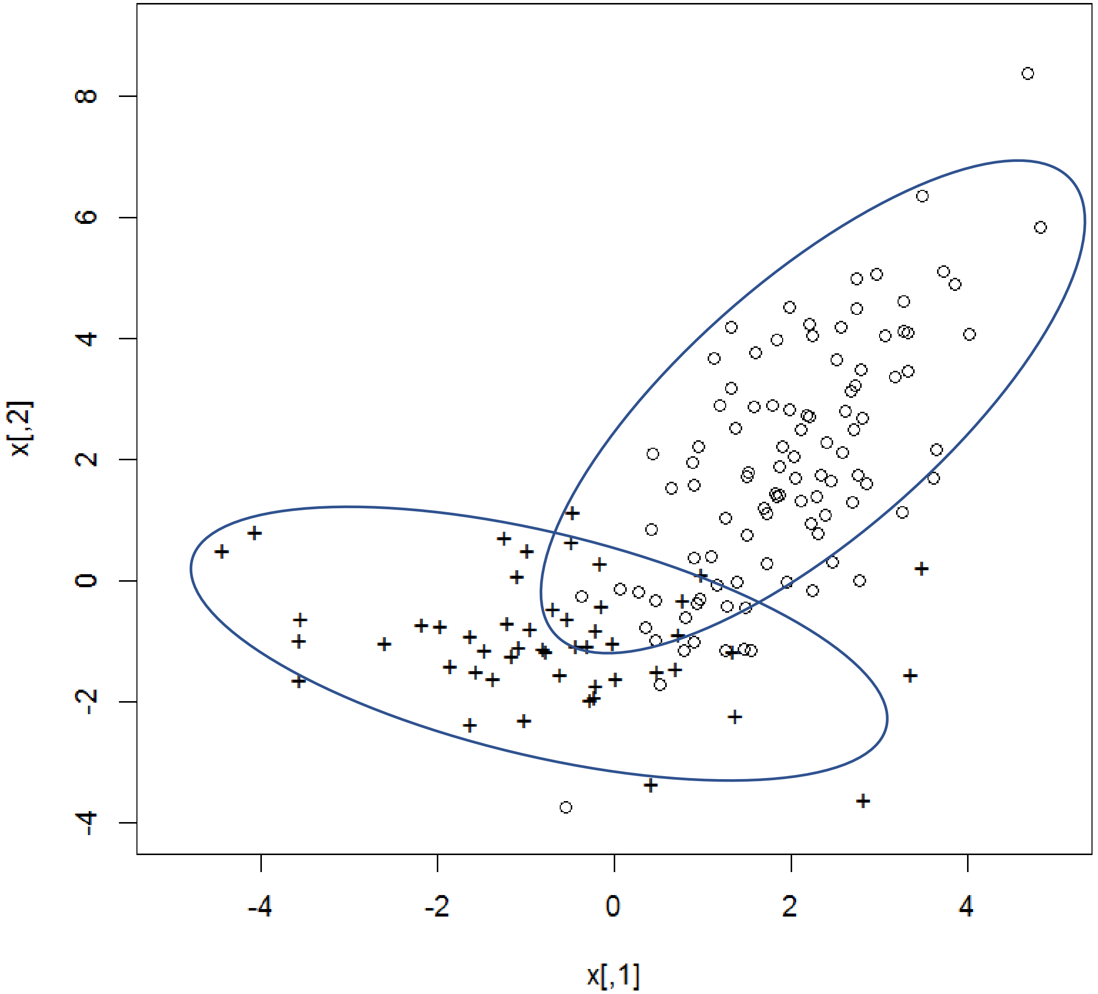
Mixture Models

- $P(x) = \sum_c P(c)P(x|c)$
- c indexes the mixture component
- Each mixture component has its own conditional density $P(x|c)$



Mixture Models

- Example
  - $P(c=1) = 2/3$
  - $P(c=2) = 1/3$
  - $P(x|c=1)$ “o”
  - $P(x|c=2)$ “+”



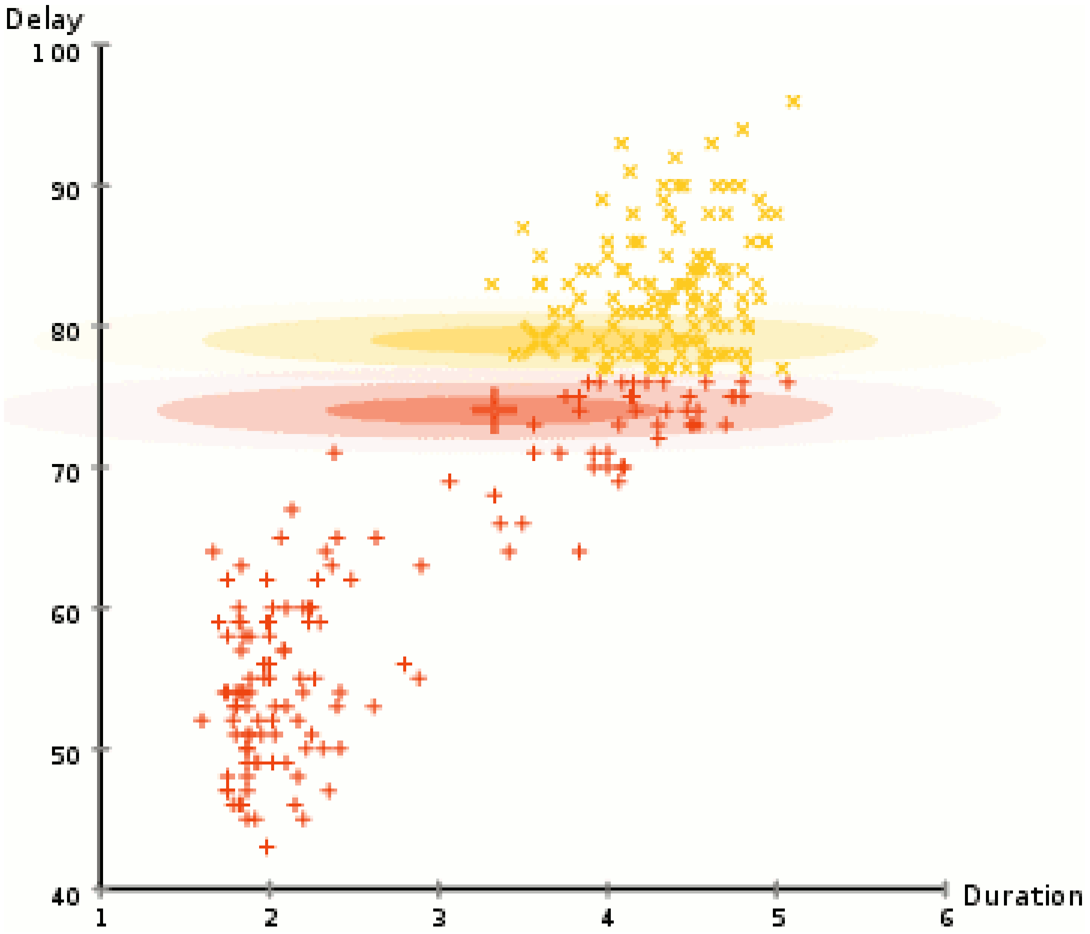
Fitting Mixture Models: Expectation Maximization (EM)

- Choose the number of mixture components K
- Create a matrix R of dimension $K \times N$. This will represent $P(c_i = k|x_i) = R[k, i]$
  - This is called the "membership probability" or "responsibility". The probability that data point i was generated by component k
- Randomly initialize $R[k, i] \in (0,1)$ subject to the constraint that $\sum_k R[k, i] = 1$

EM Algorithm Main Loop

- M step: Maximum likelihood estimation of the model parameters using the data $x_1, \dots, x_N$ and responsibilities R
  - $\hat{P}(c = k) \coloneqq \frac{1}{N} \sum_i R[k, i]$ "mixing proportions"
  - $\hat{\mu}_k \coloneqq \frac{1}{N\ \hat{P}(c=k)} \sum_i R[k, i] \cdot x_i$ "component means"
  - $\hat{\Sigma}_k \coloneqq \frac{1}{N\ \hat{P}(c=k)} \sum_i R[k, i] \cdot \left[x_i x_i^\top\right]$ "component covariances"
- E step: Re-estimate responsibilities R
  - $R[k, i] \coloneqq \hat{P}(c = k)\mathrm{Normal}\left(x_i \middle| \hat{\mu}_k, \hat{\Sigma}_k\right)$
  - $R[k, i] \coloneqq R[k, i] / \sum_k R[k, i]$ normalize the responsibilities

Animation of EM for Old Faithful data set



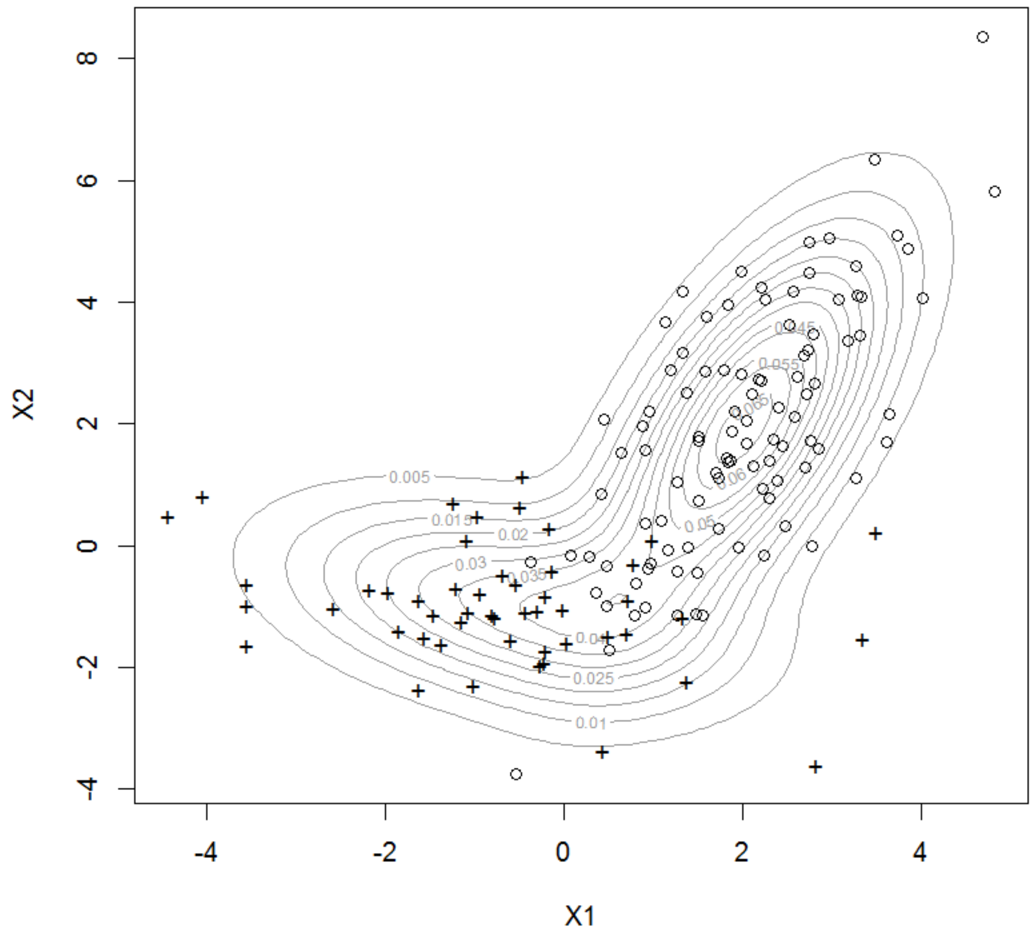
Results on Our Example [R mclust package]

Estimates

- mixing proportions
  - $(0.649, 0.351)$
- means:
  - $(2.014, 1.967)$
  - $(-0.631, -0.957)$

True values

- Mixing proportions:
  - $(0.667, 0.333)$
- Means:
  - $(2.000, 2.000)$
  - $(-1.000, -1.000)$



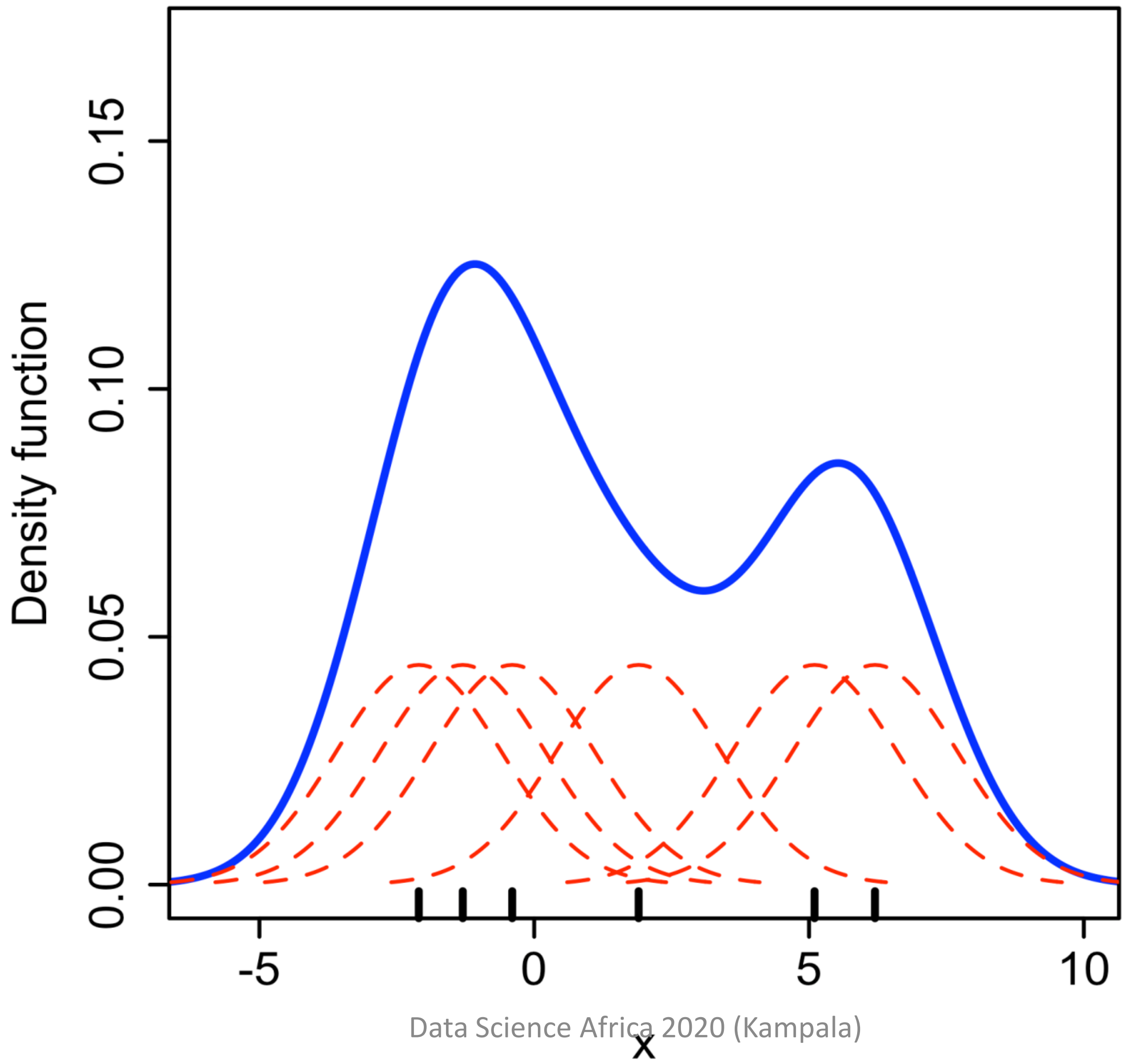
Mixture Model Summary

- Can fit very complex distributions
- EM is a local search algorithm
  - Different random initializations can find different fitted models
  - There are some new moment-based methods that find the global optimum
- Difficult to choose the number of mixture components
  - There are many heuristics for this
- Need to check for covariances going to zero
  - Leads to infinite likelihood concentrated on a single point
  - Can add a regularization term to prevent this

Kernel Density Estimation

- Define a mixture model with one mixture component for each data point
  - $\hat{P}(x) = \frac{1}{N} \sum_{i=1}^{N} K(\|x - x_i\|, \sigma^2)$
  - Often use a Gaussian Kernel $K(x, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{x^2}{2\sigma^2}\right]$
  - Often use a fixed scale σ^2. The scale is also called the "bandwidth"

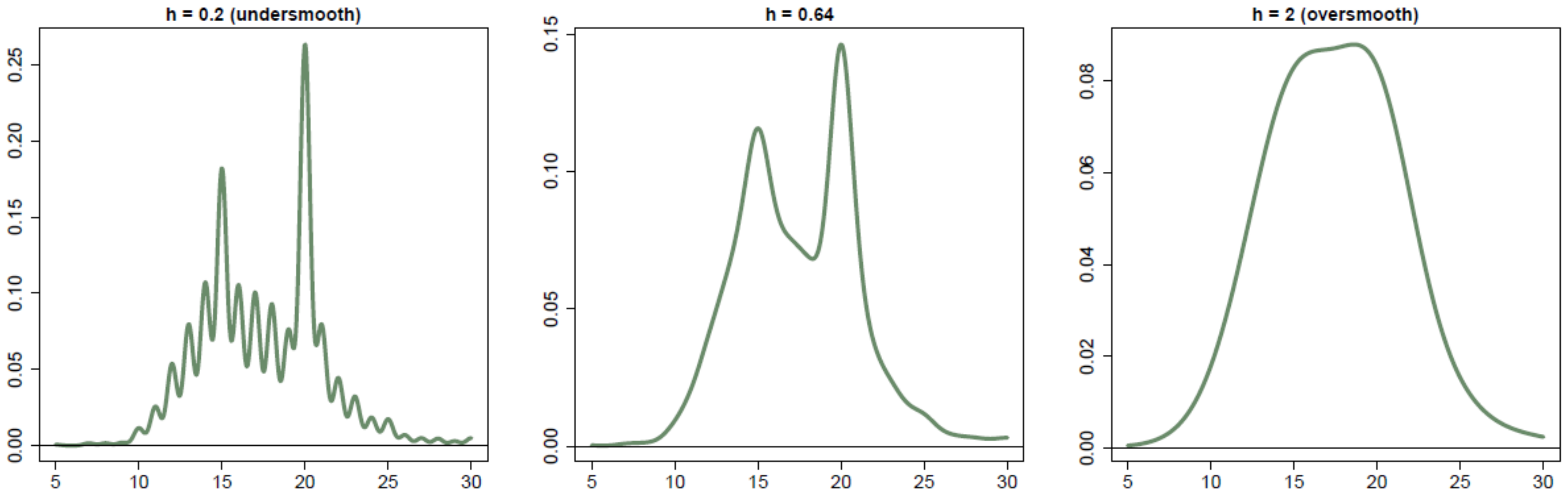
One-Dimensional Example



Source: wikipedia

Design Decisions

- Choice of Kernel: generally not super important as long as it is local
- Choice of bandwidth is very important



Challenges

- KDE in high dimensions suffers from the “Curse of Dimensionality”
- The amount of data required to achieve a desired level of accuracy scales exponentially with the dimensionality d of the problem: $\exp \frac{d+4}{2}$

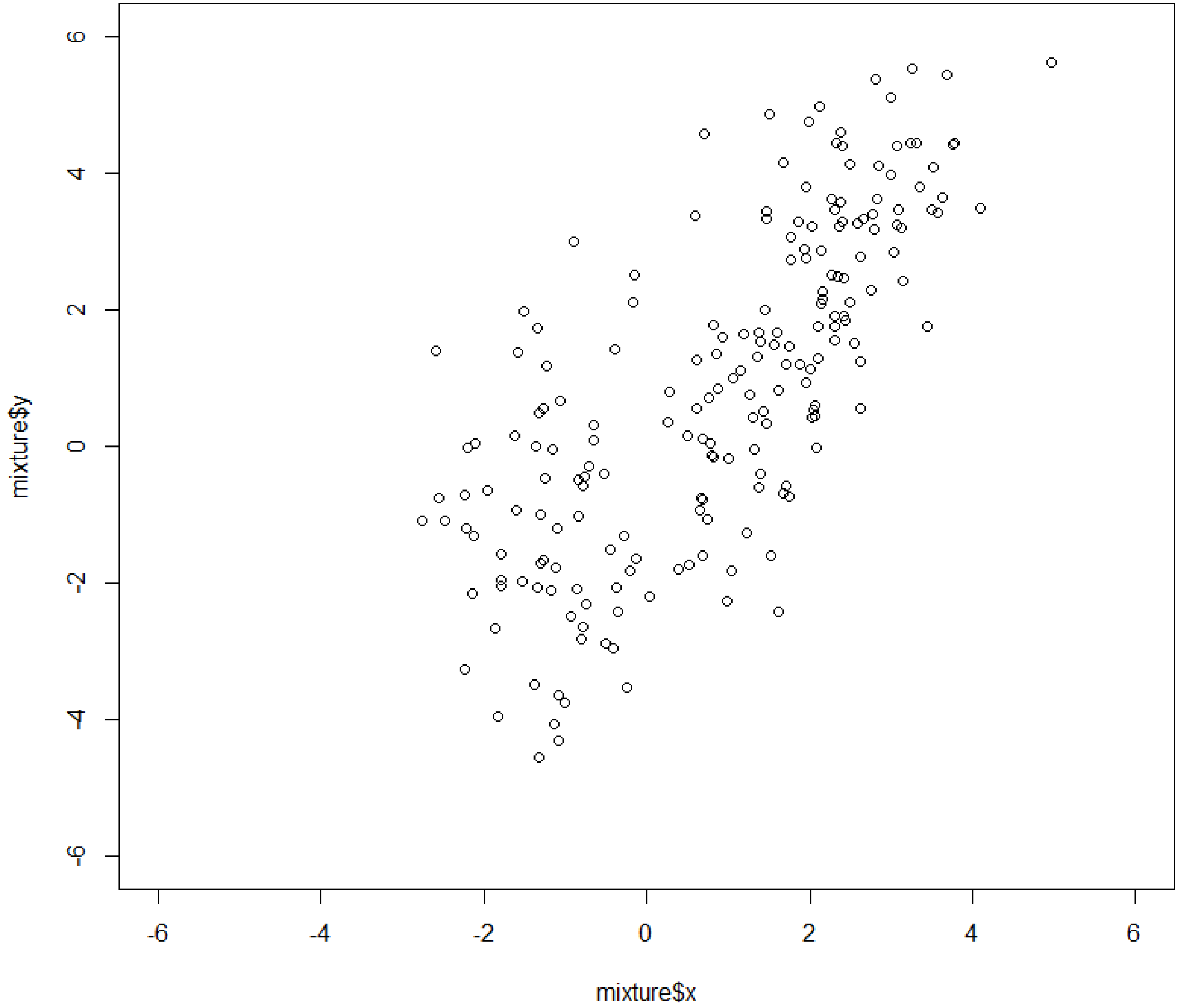
Noise-Contrastive Estimation

- Idea:
  - Label all points in D to belong to class 0
  - Uniformly sample points from a "box" that contains D and label those points as class 1
  - Fit a flexible machine learning model $\hat{f}$ to the data
  - Anomaly score $A(x_q) = \hat{f}(x_q)$

- History
  - "Well known statistical folklore" according to Hastie, Tibshirani & Friedman (2016) *Elements of Statistical Learning* 2nd edition
  - Pihlaja, Guttman & Hyvarinen (2010) "A Family of Computationally Efficient and Simple Estimators for Unnormalized Statistical Models". UAI 2010

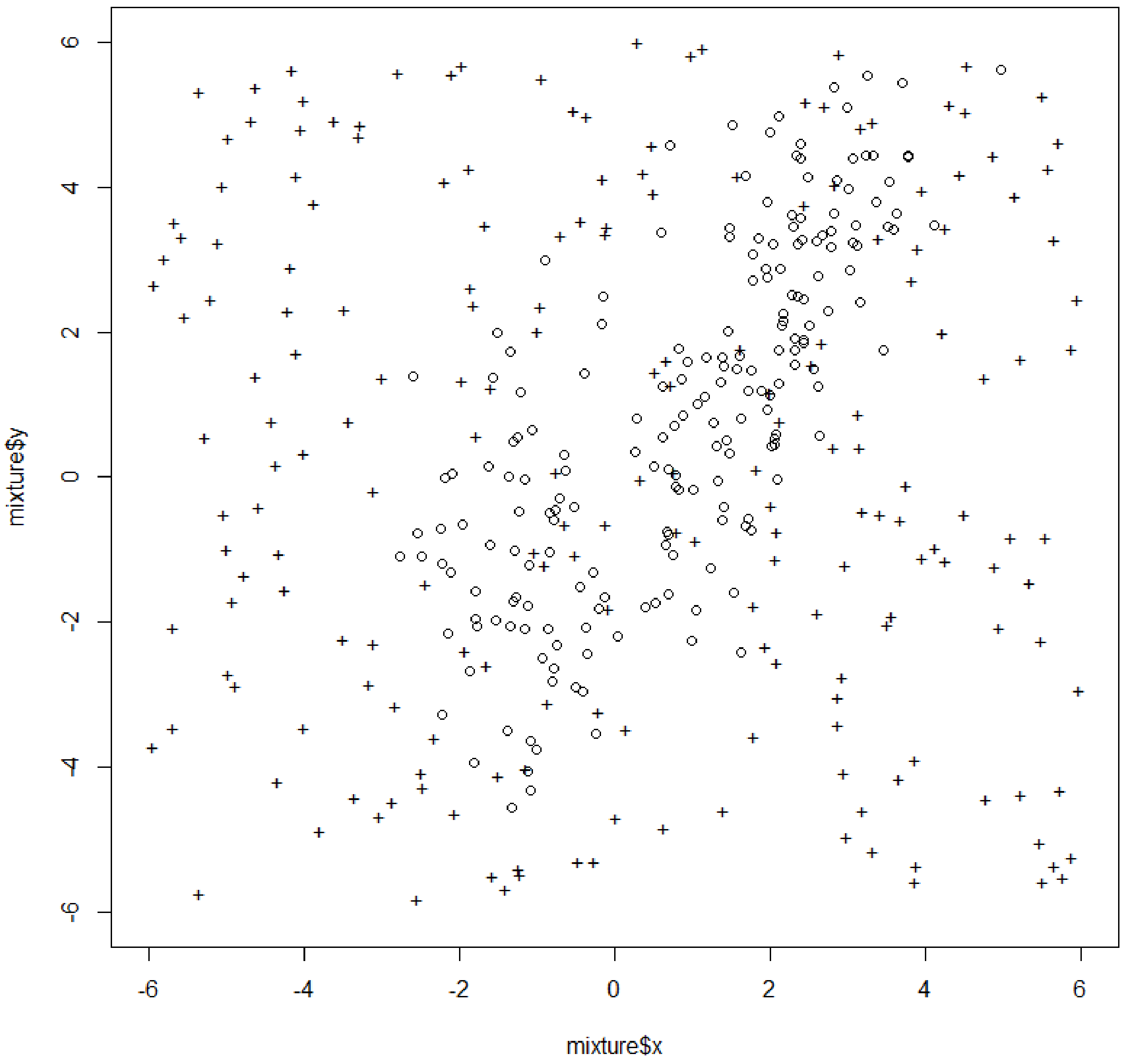
Example

- Training data D



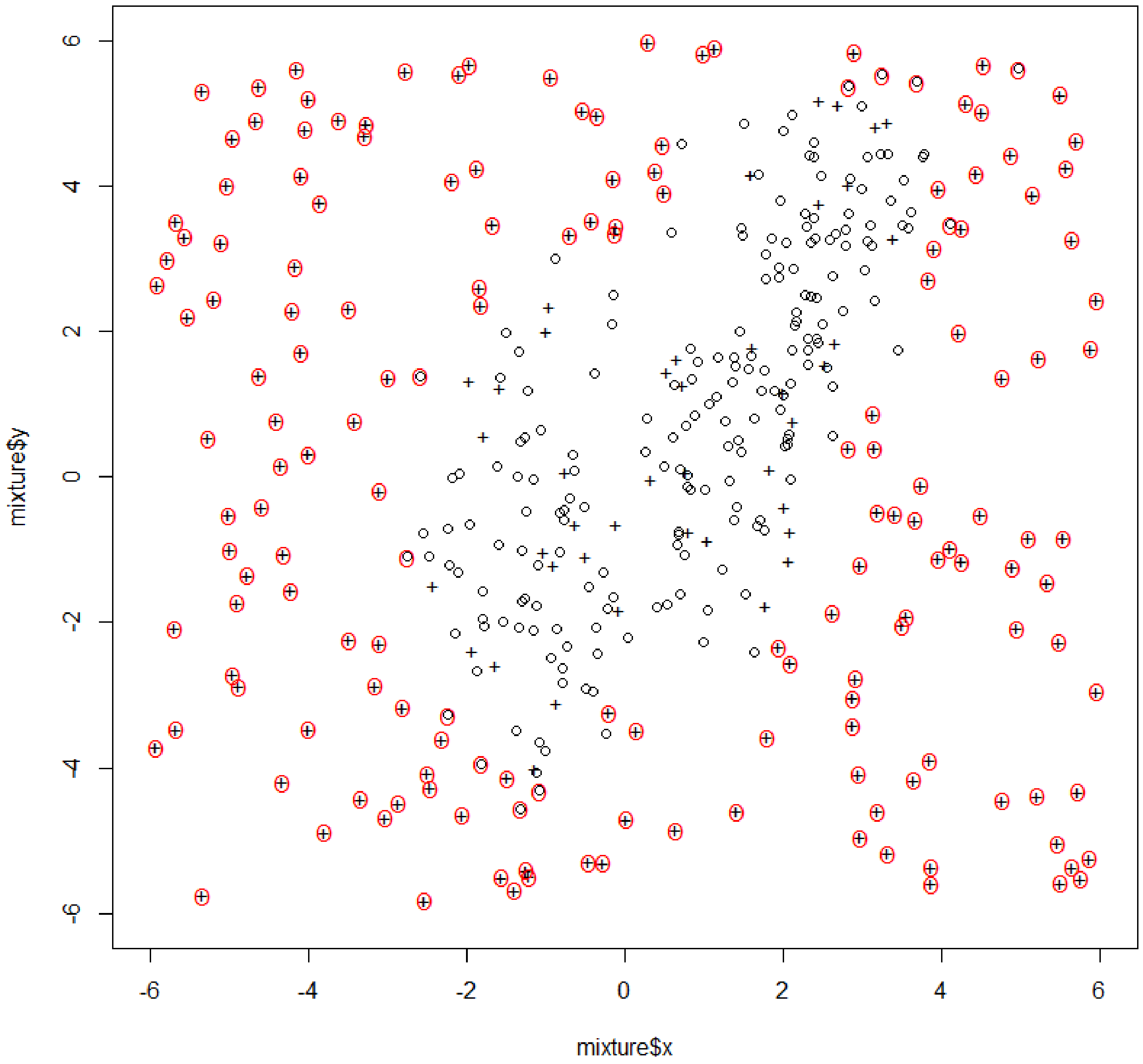
Example

- Training data D
- Random sample N



Example

- Training data D
- Random sample N
- Points x where $\hat{f}(x) > 0.5$



Fitted Function

- gbm (gradient boosting machine) with the logit link function

$$\hat{f}(x) = \log \frac{P(class = 1)}{P(class = 0)} = \sum_{\ell=1}^{L} w_\ell f_\ell(x)$$

  - where each f_ℓ is a decision tree of depth ≤ 4
  - learned via “boosting”
  - # of trees L determined via the OOB method (gbm.perf)
  - In this case, $L = 325$
- gbm is very flexible and works well for classification problems with fewer than ≈100 dimensions
- Implementations are available in sklearn and R. “xgboost” is a similar package that is somewhat faster

Extracting a density estimate from noise-contrastive learning

- $\hat{f}(x) = \log \frac{P(class=1)}{P(class=0)} = \sum_{\ell=1}^{L} w_\ell f_\ell(x)$
- The numerator is the uniform distribution, so each point had probability 1/200

$$\log \frac{1}{200} - \log P_D(x) = \sum_{\ell=1}^{L} w_\ell f_\ell(x)$$

$$\log P_D(x) = \log \frac{1}{200} - \sum_{\ell=1}^{L} w_\ell f_\ell(x)$$

$$A(x) = -\log P_D(x) \approx \sum_{\ell=1}^{L} w_\ell f_\ell(x)$$

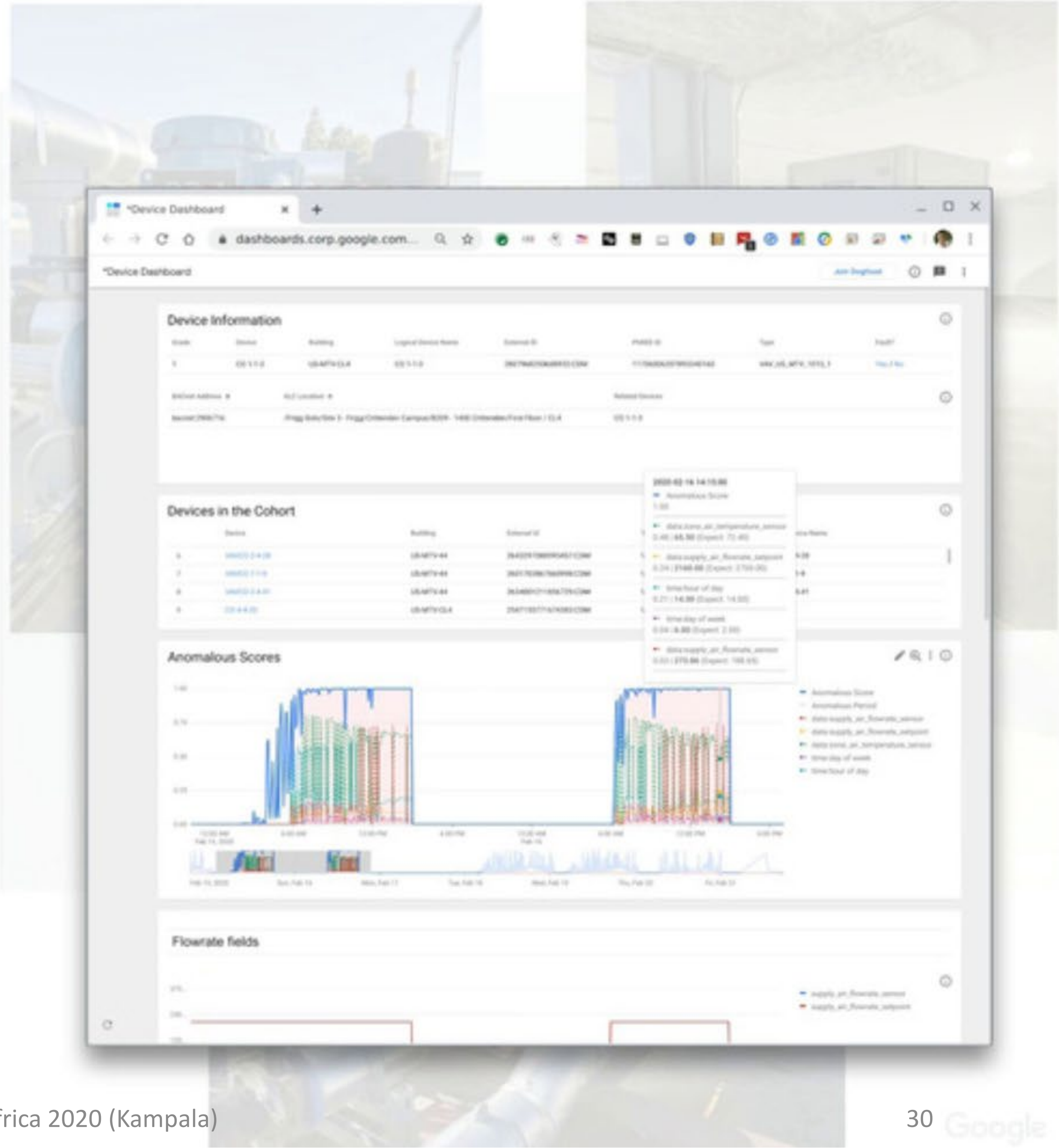
Case Study: Smart Buildings

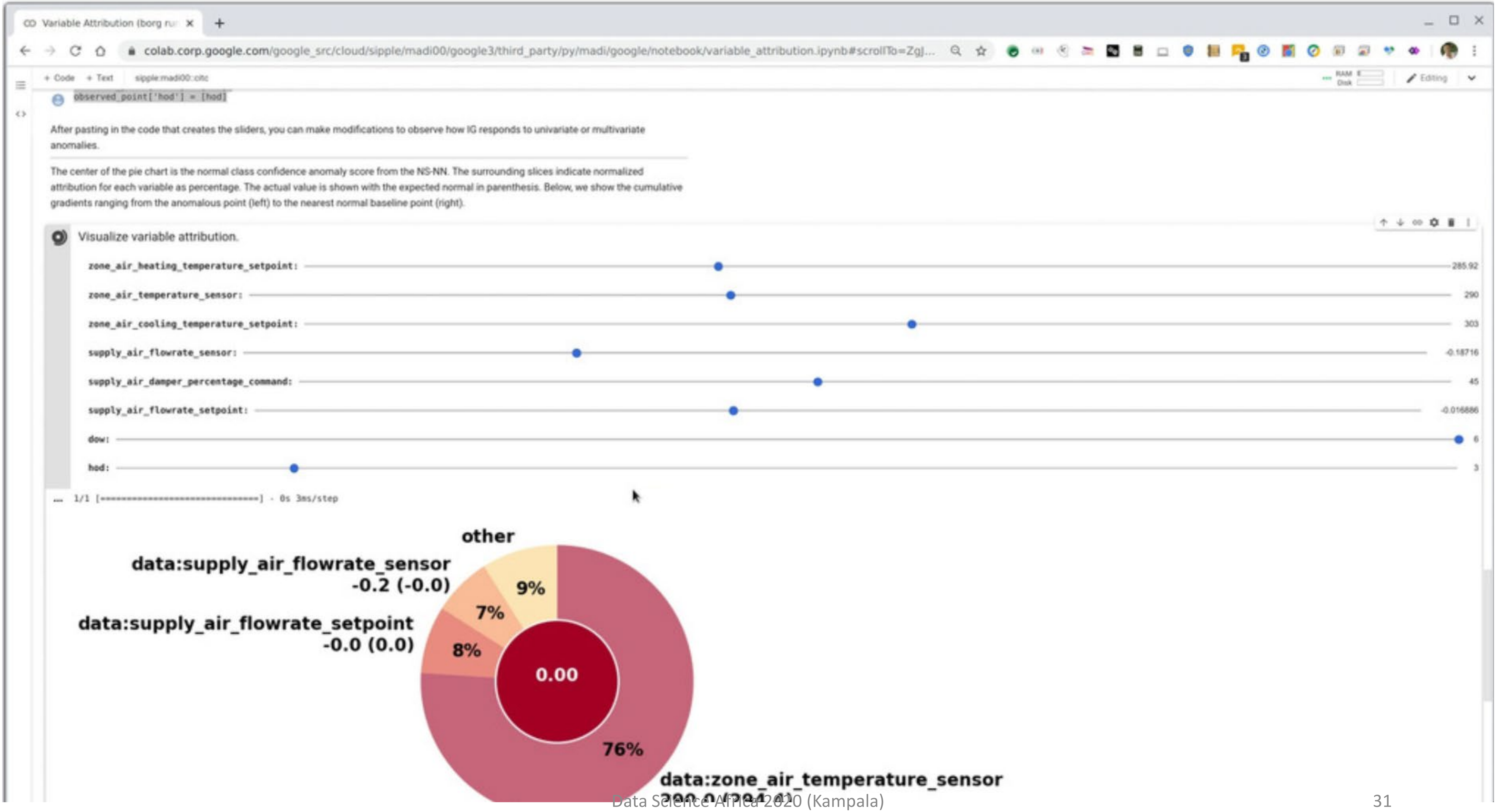
John Sipple (ICML 2020)

Objective: *Make buildings smarter, secure and reduce energy use! Improve occupant comfort and productivity while also improving facilities' operation efficiencies.*

120 million measurements daily, generated by over **15,000 climate control devices, in 145 Google buildings**

Since going live in June 2019, FDD has created **458 facilities technician work orders, with a 44% True Positive rate**





Variable Attribution (borg ru
colab.corp.google.com/google_src/cloud/sipple/madi00/google3/third_party/py/madi/google/notebook/variable_attribution.ipynb#scrollTo=ZgJ...
+ Code
+ Text
sipple:madi00::citc
observed_point['hod'] = [hod]
After pasting in the code that creates the sliders, you can make modifications to observe how IG responds to univariate or multivariate anomalies.
The center of the pie chart is the normal class confidence anomaly score from the NS-NN. The surrounding slices indicate normalized attribution for each variable as percentage. The actual value is shown with the expected normal in parenthesis. Below, we show the cumulative gradients ranging from the anomalous point (left) to the nearest normal baseline point (right).
Visualize variable attribution.
zone_air_heating_temperature_setpoint: 285.92
zone_air_temperature_sensor: 290
zone_air_cooling_temperature_setpoint: 303
supply_air_flowrate_sensor: -0.18716
supply_air_damper_percentage_command: 45
supply_air_flowrate_setpoint: -0.016886
dow: 6
hod: 3
1/1 [==============================] - 0s 3ms/step
other
data:supply_air_flowrate_sensor
-0.2 (-0.0)
data:supply_air_flowrate_setpoint
-0.0 (0.0)
9%
7%
8%
0.00
76%
data:zone_air_temperature_sensor

Deep Density Estimation using Flow Models

Factoring the Joint Density into Conditional Distributions:

- Let $x = (x_1, \ldots, x_d)$ be a vector of random variables. We wish to model the joint distribution $P(x)$.
- By the chain rule of probability, we can write this as
$$P(x) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \cdots P(x_d|x_1, \ldots, x_{d-1})$$
- We can model $P(x_1)$ using any 1-D method (e.g., KDE).
- We can model each conditional distribution using regression methods

Linear Regression = Conditional Density Estimation

- Let $\mathrm{x} = (x_1, \ldots, x_J)$ be the predictor variables and y be the response variable
- Standard least-squares linear regression:
- $P(y|x) \sim \mathrm{Normal}(y;\ \mu(x), \sigma^2)$
  - where $\mu(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_J x_J$
  - and σ^2 is a fitted constant
- Neural networks trained to minimize squared error model the mean as $\mu(x) = NNet(x; W)$, where W are the weights of the neural network. The variance can be estimated as $\frac{1}{N}\sum_i \left(y_i - \mu(x_i)\right)^2$

Deep Neural Networks for Density Estimation

Masked Auto-Regressive Flow (Papamarkarios, 2017)

- Apply the chain rule of probability
- $P\left(x_j \middle| x_{1:j-1}\right) = \mathrm{Normal}\left(x_j; \mu_j, \left(\exp \alpha_j\right)^2\right)$ where
  - $\mu_j = f_j\left(x_{1:j-1}\right)$ and $\alpha_j = g_j\left(x_{1:j-1}\right)$ are implemented by neural networks
- Re-parameterization trick for sampling $x_j \sim P\left(x_j \middle| x_{1:j-1}\right)$
  - Let $u_j \sim \mathrm{Normal}(0,1)$
  - $x_j = u_j \exp \alpha_j + \mu_j$
  - (rescale by the standard deviation and displace by the mean)

Transformation View

- Equivalent model: $\mathbf{u} \sim \mathrm{Normal}(\mathbf{u}; 0, I)$ is a vector of J standard normal random variates
- $\mathbf{x} = F(\mathbf{u})$ transforms those random variates into the observed data
- To compute $P(\mathbf{x})$ we can invert this function and evaluate $\mathrm{Normal}(F^{-1}(x); 0, I)$ “almost”
- $P(x) = \mathrm{Normal}(F^{-1}(x); 0, I) \left|\det\left[\frac{\partial F^{-1}(x)}{\partial x}\right]\right|$
  - where $\det$ denotes the determinant
  - This ensures that P integrates to 1

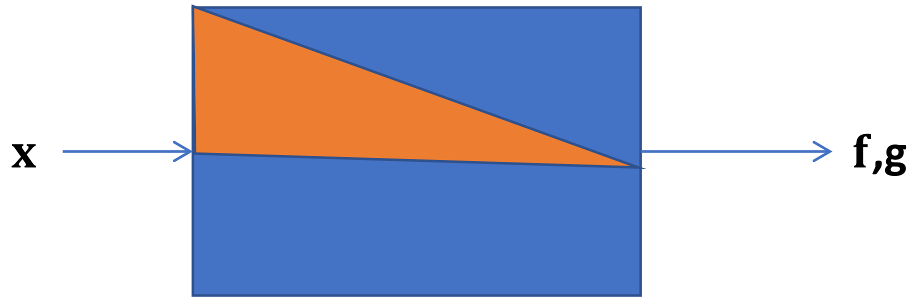
Derivative of F^{-1} is Simple

- Because of the "triangular" structure created by the chain rule,
- $\left|\det\left[\frac{\partial F^{-1}(x)}{\partial x}\right]\right| = \exp\left(-\sum_j \alpha_j\right)$

F is Easy to Invert

- $u_j = (x_j - \mu_j)\exp(-\alpha_j)$
  - where $\mu_j = f_j(x_{1:j-1})$ and $\alpha_j = g_j(x_{1:j-1})$

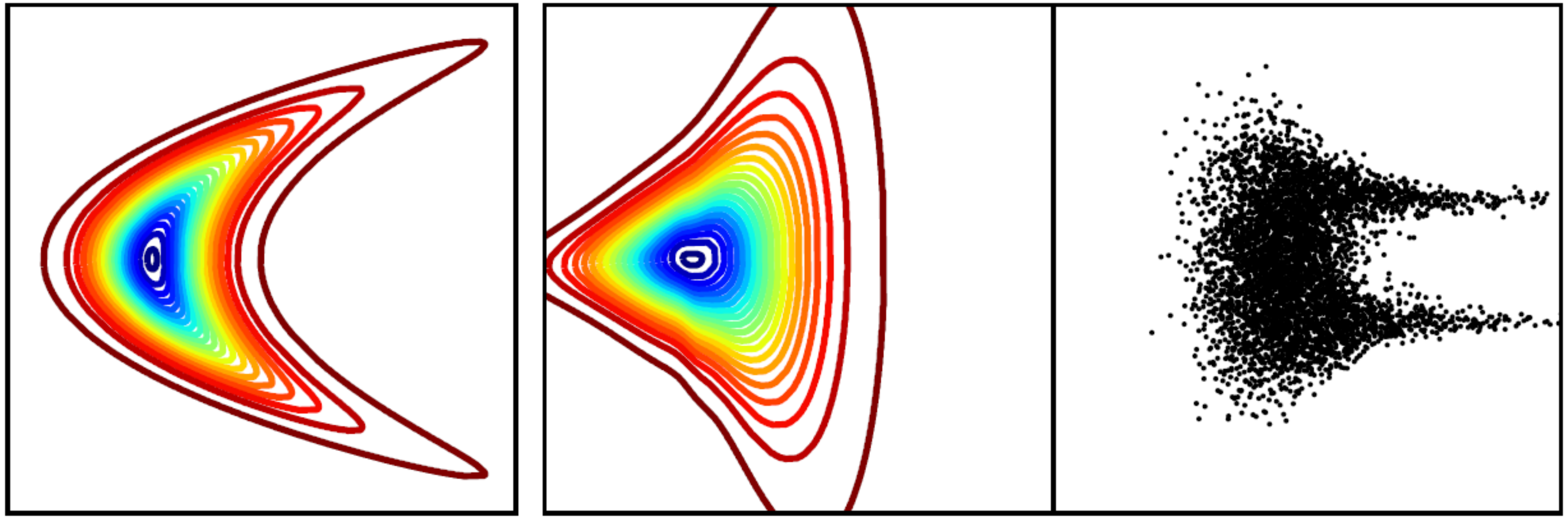
Masked Auto-Regressive Flow



- "mask" ensures that f_j and g_j only depend on $x_{1:j-1}$

Stacking MAFs

- One MAF network is often not sufficient

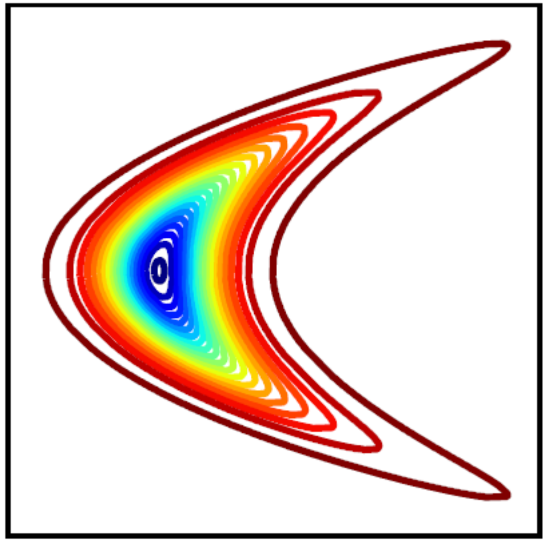


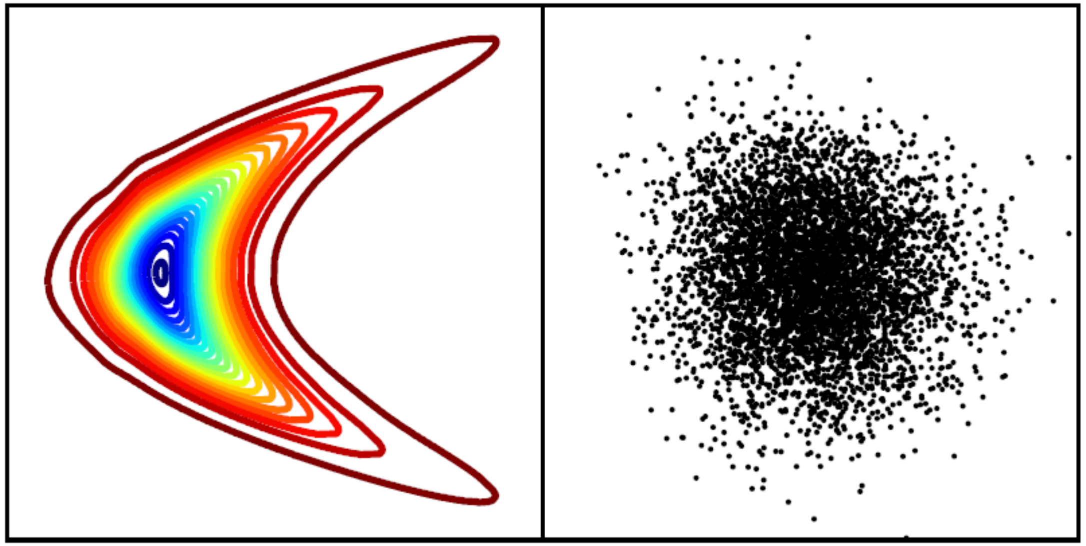
True Density

Fitted Density from single MAF network

Distribution of the $\mathbf{u}$ values

Stack MAFs until the $\mathbf{u}$ values are Normal(0,I)





True Density

Fitted Density from stack of 5 MAFs

Distribution of the $\mathbf{u}$ values

Test Set Log Likelihood

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Gaussian	-7.74 ± 0.02	-3.58 ± 0.75	-27.93 ± 0.02	-37.24 ± 1.07	96.67 ± 0.25
MADE	-3.08 ± 0.03	3.56 ± 0.04	-20.98 ± 0.02	-15.59 ± 0.50	148.85 ± 0.28
MADE MoG	$\mathbf{0.40 \pm 0.01}$	8.47 ± 0.02	$\mathbf{-15.15 \pm 0.02}$	-12.27 ± 0.47	153.71 ± 0.28
Real NVP (5)	-0.02 ± 0.01	4.78 ± 1.80	-19.62 ± 0.02	-13.55 ± 0.49	152.97 ± 0.28
Real NVP (10)	0.17 ± 0.01	8.33 ± 0.14	-18.71 ± 0.02	-13.84 ± 0.52	153.28 ± 1.78
MAF (5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	$\mathbf{-11.75 \pm 0.44}$	155.69 ± 0.28
MAF (10)	0.24 ± 0.01	$\mathbf{10.08 \pm 0.02}$	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF MoG (5)	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	$\mathbf{-11.68 \pm 0.44}$	$\mathbf{156.36 \pm 0.28}$

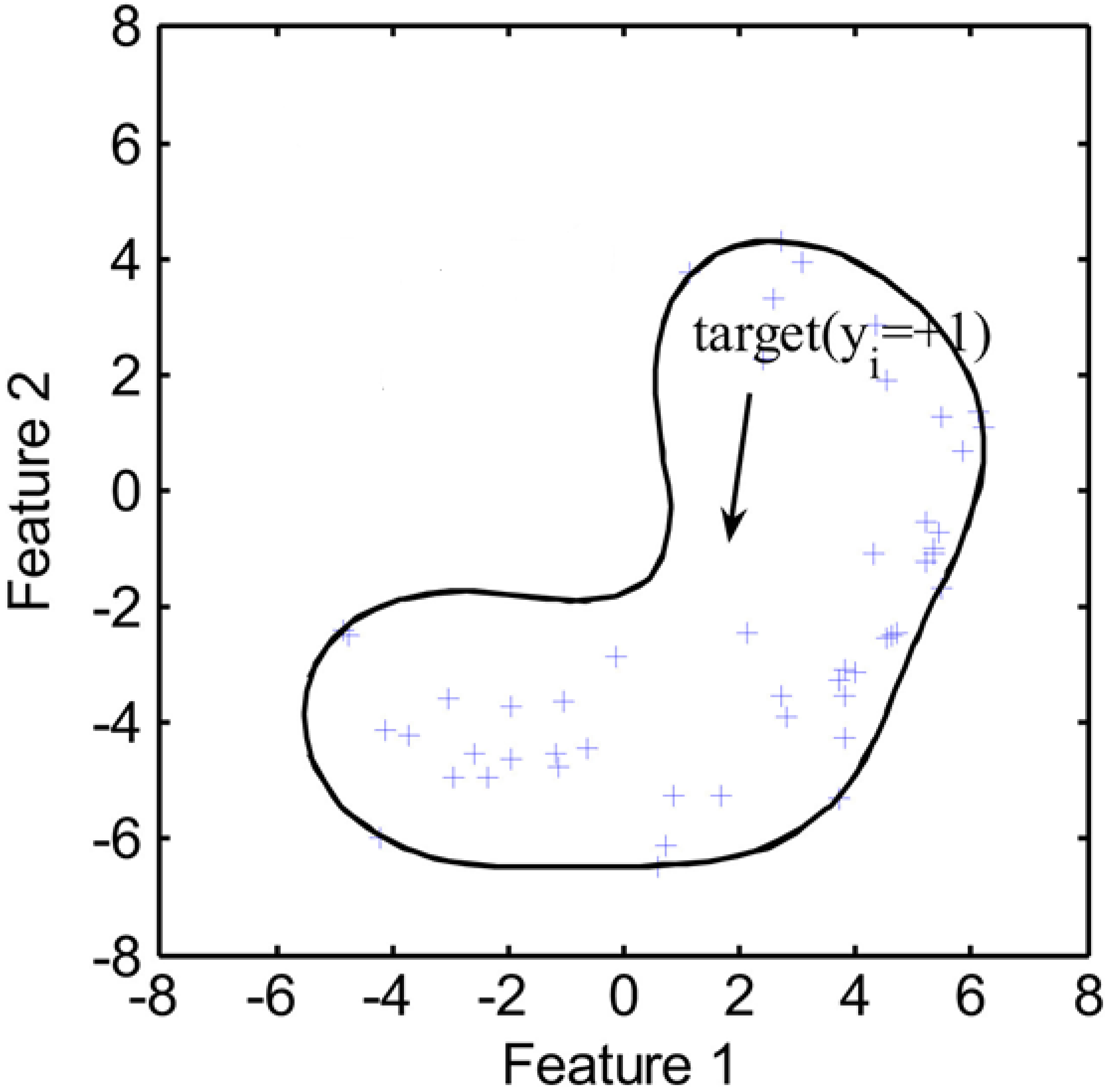
Priyank, Kobyzev, Yu & Brubaker (ICML 2020): Use a Student t distribution instead of a Gaussian.
This allows you to generate distributions with heavy tails, which Gaussians cannot do

Approach 2: Quantile Methods

- Surround the data by a function f that captures $1 - \epsilon$ of the training data
  - One-Class Support Vector Machine (OCSVM)
    - f is a hyperplane in "kernel space"
  - Support Vector Data Description (SVDD)
    - f is a sphere is "kernel space"
- Closely related to kernel density estimation:

$$f(x) = \sum_{x_i \in SV} \alpha_i K(x, x_i) - \rho$$

where SV is the set of "support vectors". These are a carefully-selected subset of the training data points. ρ is a scalar parameter



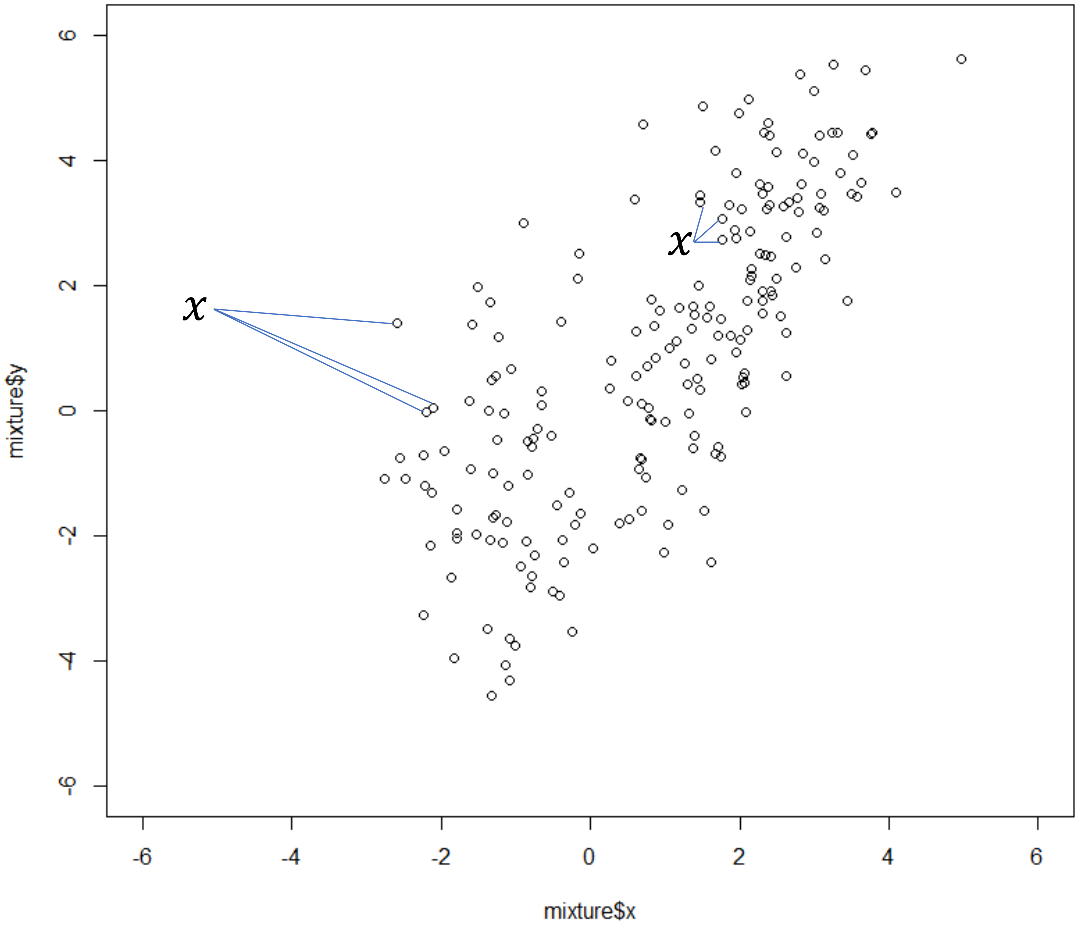
Approach 3: Distance-Based Methods

- Do we really need to estimate probability densities?
- In most applications, we just need a way of ranking the anomalies

- Define a distance $d(x_i, x_j)$
- $A\left(x_q\right) = \min_{x \in D} d(x_q, x)$

- This can be made more robust by looking at the average distance to the k-nearest points
  - “k-nn anomaly detection”

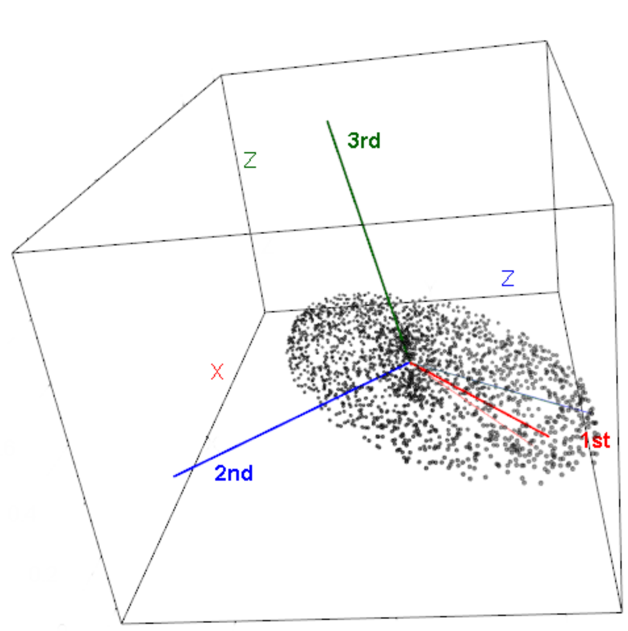
- This can be normalized by dividing by the distance of each neighbor to *their* k-nearest neighbors
  - “Local Outlier Factor (LOF)”

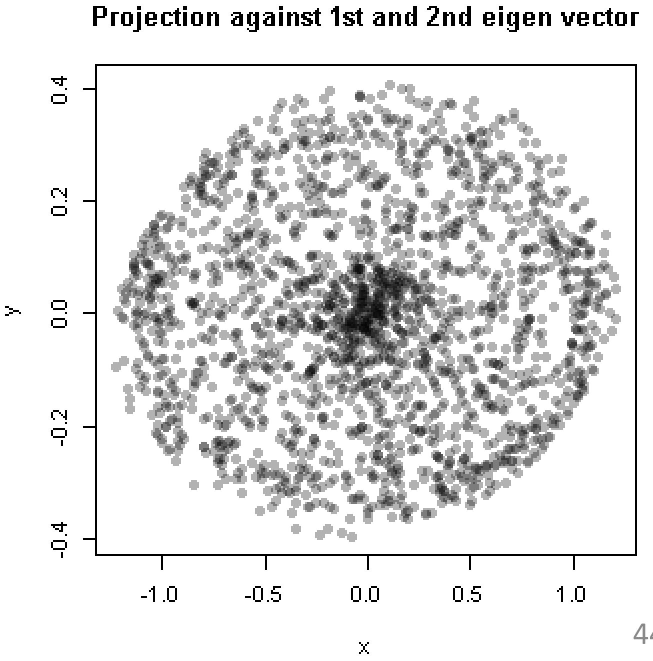


Computing Distances

- Method 1:
  - Rotate the data to obtain orthogonal dimensions (optional)
    - Apply Principle Component Analysis
  - Rescale each feature to have zero mean and unit variance
  - Then use Euclidean distance
- Method 2:
  - Fit a multi-variate Gaussian distribution to your data
    - Mean vector: $\boldsymbol{\mu}$
    - Covariance matrix: $\boldsymbol{\Sigma}$
  - Compute the Mahalanobis Distance:
    - $d_{MH}(x, x') = (x - \boldsymbol{\mu})^{\top}\boldsymbol{\Sigma}^{-1}(x - \boldsymbol{\mu})$
    - This handle the correlation structure of the data
- Method 3:
  - Fit a random forest classifier
  - Use the random forest similarity distance: probability that x and x' are sent to the same leaf in a randomly-selected tree
- Remember:
  - All data transformation parameters should be computed on the training data only
  - Then applied to the validation and test data

PCA applied to an ellipsoidically shaped point cloud





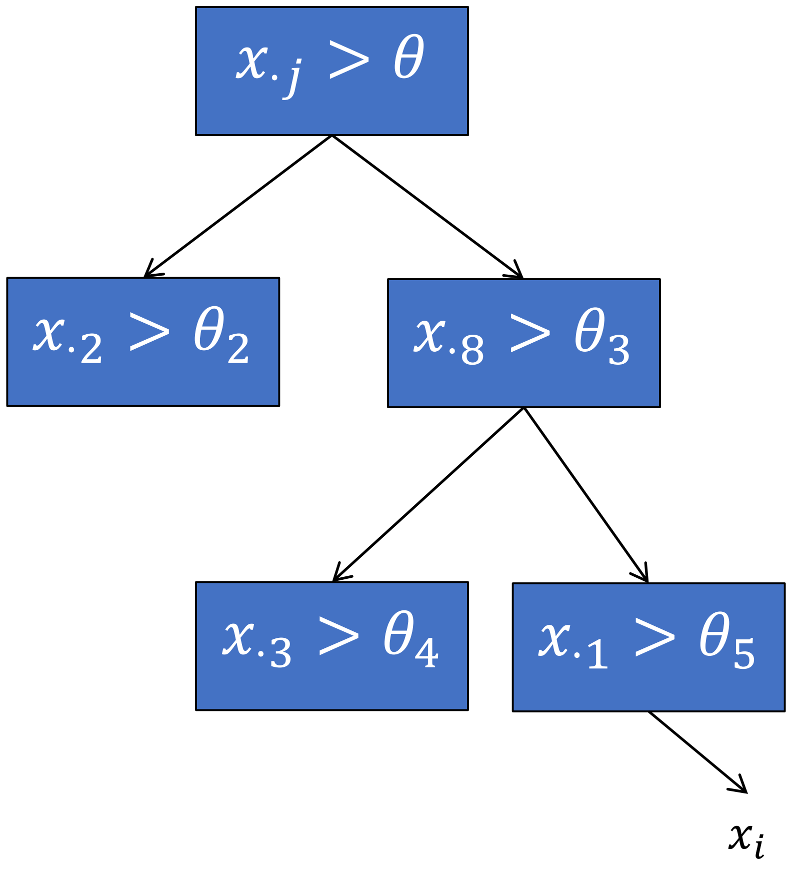
https://www.joyofdata.de/blog/illustration-of-principal-component-analysis-pca/

Approach 4: Projection Methods

- Isolation Forest [Liu, Ting, Zhou, 2011]
- LODA [Pevny, 2016]

Isolation Forest [Liu, Ting, Zhou, 2011]

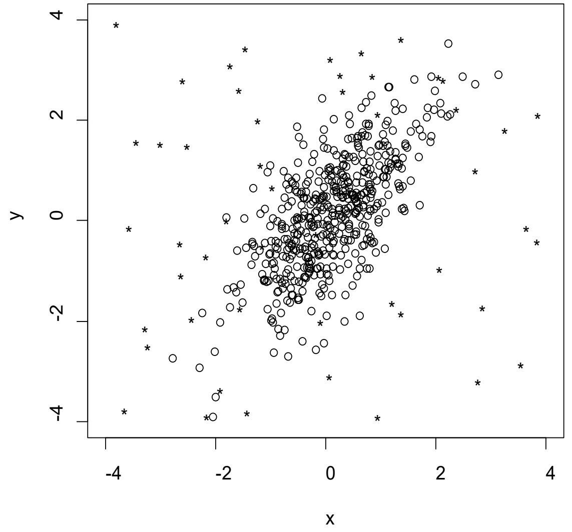
- Construct a fully random binary tree
  - choose attribute j at random
  - choose splitting threshold θ uniformly from $\left[\min(x_{\cdot j}), \max(x_{\cdot j})\right]$
  - until every data point is in its own leaf
  - let $d(x_i)$ be the depth of point x_i
- repeat L times
  - let $\bar{d}(x_i)$ be the average depth of x_i
  - $A(x_i) = 2^{-\left(\frac{\bar{d}(x_i)}{r(x_i)}\right)}$
    - $r(x_i)$ is the expected depth



LODA: Lightweight Online Detector of Anomalies

[Pevny, 2016]

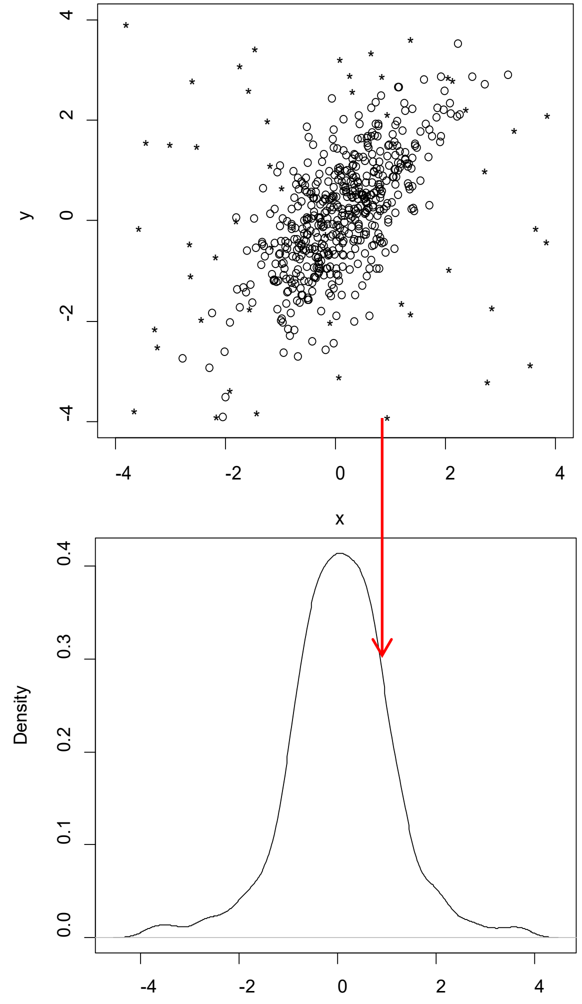
- $\Pi_1, \dots, \Pi_L$ set of L sparse random projections
- $f_1, \dots, f_L$ corresponding 1-dimensional density estimators
- $S(x) = \frac{1}{L} \sum_\ell - \log f_\ell(x)$ average "surprise"



LODA: Lightweight Online Detector of Anomalies

[Pevny, 2016]

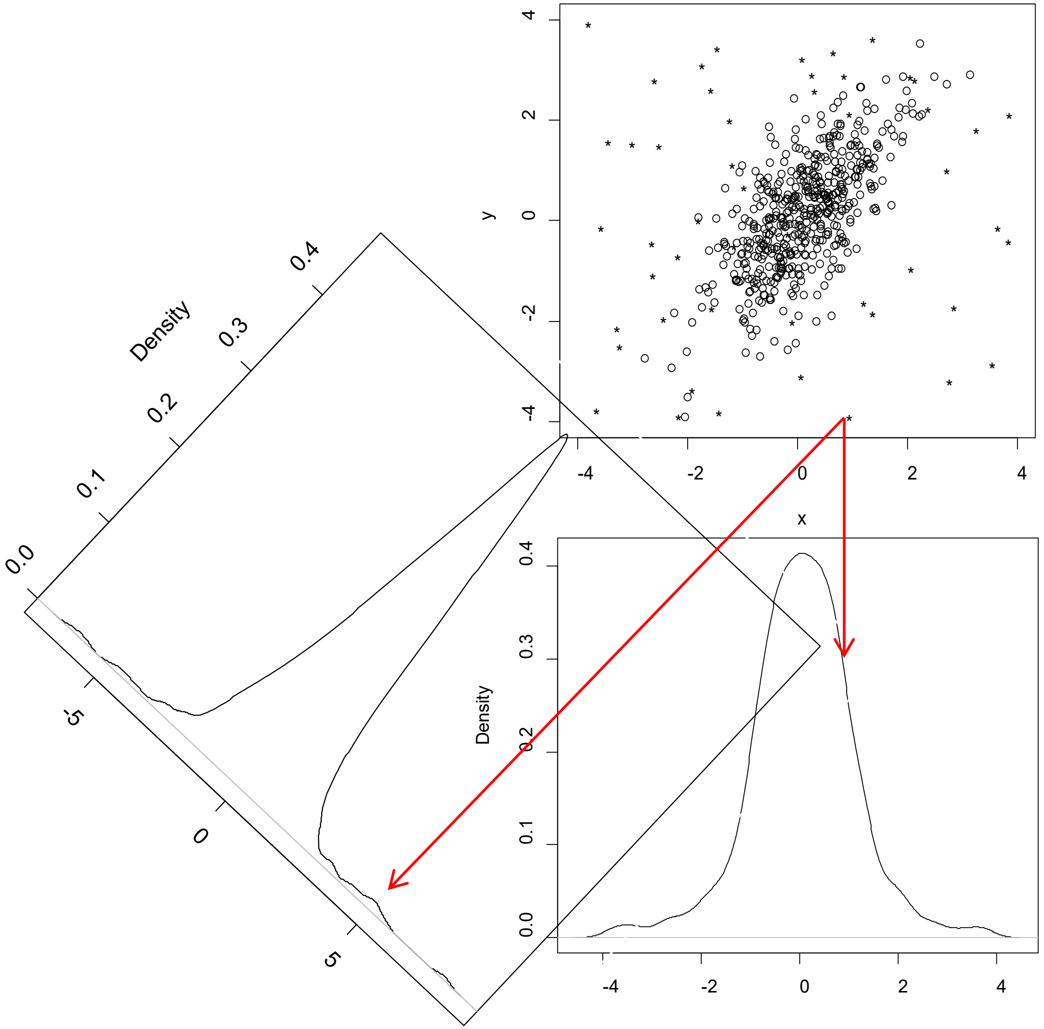
- $\Pi_1, \ldots, \Pi_L$ set of L sparse random projections
- $f_1, \ldots, f_L$ corresponding 1-dimensional density estimators
- $S(x) = \frac{1}{L} \sum_\ell - \log f_\ell(x)$ average “surprise”



LODA: Lightweight Online Detector of Anomalies

[Pevny, 2016]

- $\Pi_1, \ldots, \Pi_L$ set of L sparse random projections
- $f_1, \ldots, f_L$ corresponding 1-dimensional density estimators
- $S(x) = \frac{1}{L} \sum_\ell - \log f_\ell(x)$ average "surprise"



Benchmarking Study

[Andrew Emmott]

- Most AD papers only evaluate on a few datasets
- Often proprietary or very easy (e.g., KDD 1999)
- Research community needs a large and growing collection of public anomaly benchmarks

[Emmott, Das, Dietterich, Fern, Wong, 2013; KDD ODD-2013]
[Emmott, Das, Dietterich, Fern, Wong. 2016; arXiv 1503.01158v2]
[Emmott, MS Thesis. 2020]

Benchmarking Methodology

- Select 19 data sets from UC Irvine repository
- Choose one or more classes to be “anomalies”; the rest are “nominals”
- Manipulate
  - Relative frequency
  - Point difficulty
  - Irrelevant features
  - Clusteredness
- 20 replicates of each configuration
- Result: 11,888 Non-trivial Benchmark Datasets

Algorithms

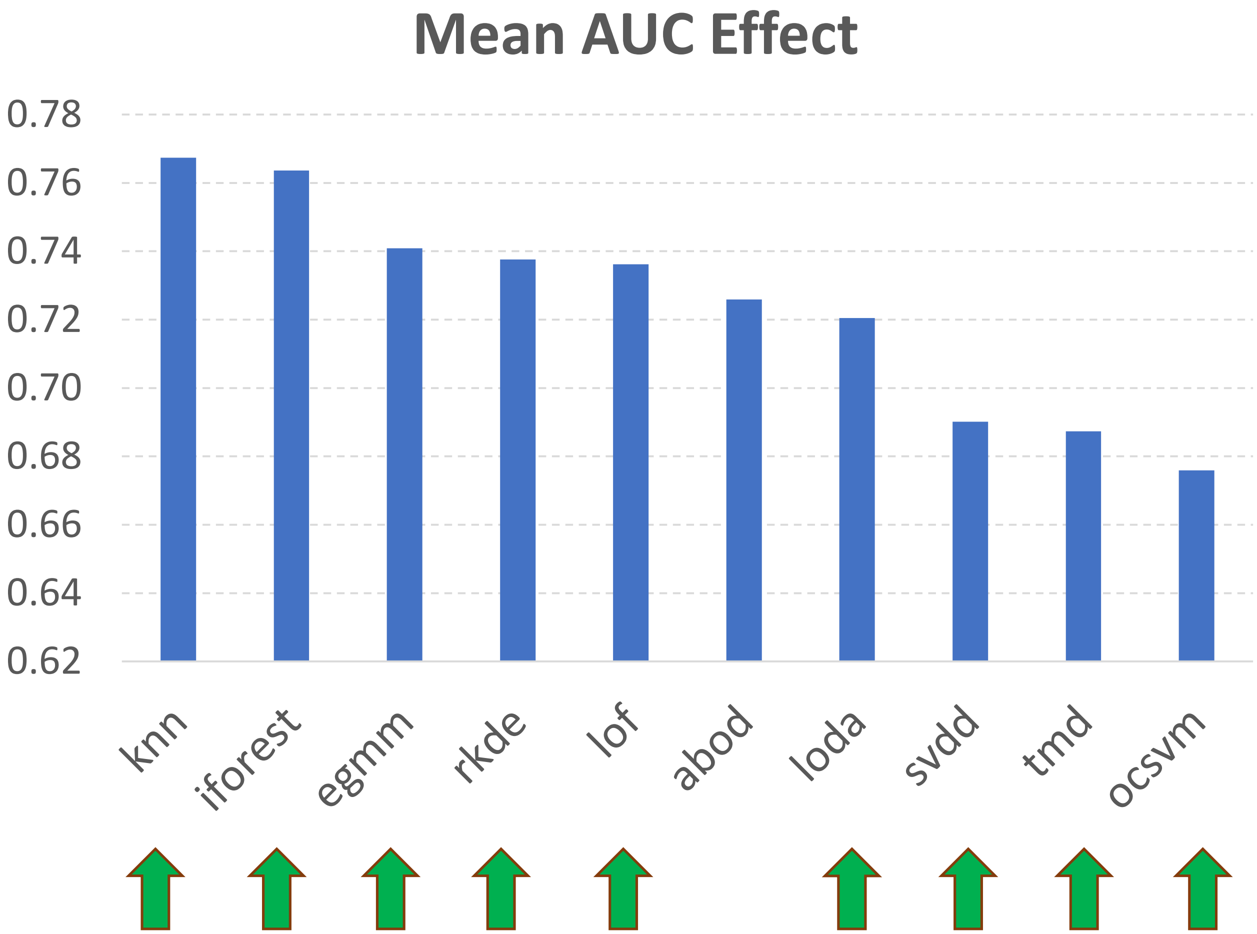
- Density-Based Approaches
  - RKDE: Robust Kernel Density Estimation (Kim & Scott, 2008)
  - EGMM: Ensemble Gaussian Mixture Model (our group)
- Quantile-Based Methods
  - OCSVM: One-class SVM (Schoelkopf, et al., 1999)
  - SVDD: Support Vector Data Description (Tax & Duin, 2004)
- Neighbor-Based Methods
  - k-NN: Mean distance to k-nearest neighbors
  - LOF: Local Outlier Factor (Breunig, et al., 2000)
  - ABOD: kNN Angle-Based Outlier Detector (Kriegel, et al., 2008)
- Projection-Based Methods
  - IFOR: Isolation Forest (Liu, et al., 2008)
  - LODA: Lightweight Online Detector of Anomalies (Pevny, 2016)

Analysis of Variance

- Linear ANOVA
  - $metric \sim rf + pd + cl + ir + mset + algo$
    - rf: relative frequency
    - pd: point difficulty
    - cl: normalized clusteredness
    - ir: irrelevant features
    - mset: “Mother” set
    - algo: anomaly detection algorithm
- Validate the effect of each factor
- Assess the *algo* effect while controlling for all other factors
- $metric$: area under the ROC curve for the nominal vs. anomaly binary decision

Benchmarking Study Results

- 19 UCI Datasets
- 8 Leading “feature-based” algorithms
- 11,888 non-trivial benchmark datasets
- Mean AUC effect for “nominal” vs. “anomaly” decisions
    - Controlling for
        - Parent data set
        - Difficulty of individual queries
        - Fraction of anomalies
        - Irrelevant features
        - Clusteredness of anomalies
- Baseline method: Distance to nominal mean (“tmd”)
- Best methods: K-nearest neighbors and Isolation Forest (projection method)
- Worst methods: Kernel-based OCSVM and SVDD



Employs a distance

Anomaly Detection Exercise

Part 2: Anomaly Detection in Computer Vision

- **Challenges:**
- No easy distance metrics
- Very high dimension
- High degree of nuisance novelty in natural images

- **State-of-the-art methods have difficulty deciding that SVHN house numbers are anomalies compared to CelebA!**

Faces from CelebA

House Number from SVHN

Anomaly Detection in Computer Vision

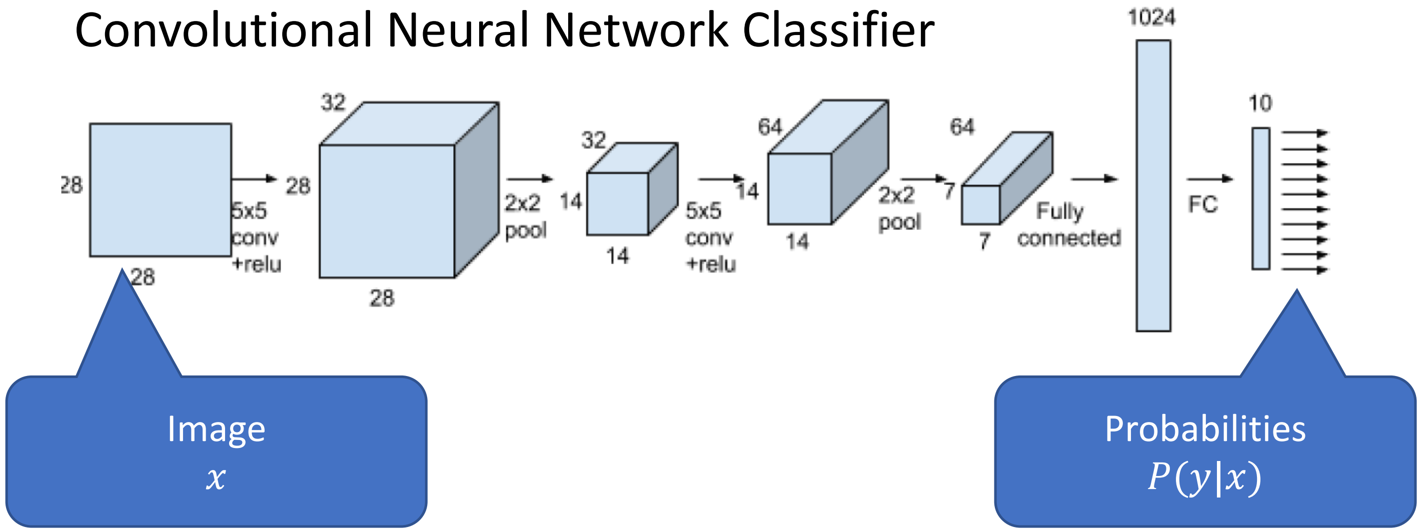
- **Proposed Methods**

1. Train a Classifier and extract an anomaly score
2. Autoencoders: Learn a latent space image representation`
3. Deep Density Estimation: Learn a probability distribution over images
4. Learn a Distance Metric
5. Self-Supervision on Auxiliary Tasks

Methods (1): Classifier-Based Approaches: Indecision

- **Classifier approach**
- Learn classifier $f(x) = P(y|x)$
- Compute a measure of uncertainty:
  - $A(x) = 1 - \arg\max_y P(y|x_q)$
  - $A(x) = H\left(P(y|x_q)\right)$
- This should not work, because the classifier should discard all aspects of x that are irrelevant to classification.
  - Density estimation: Learn $P(x)$
  - Classification: Learn $P(y|x)$
  - Surprise: It works pretty well

Convolutional Neural Network Classifier

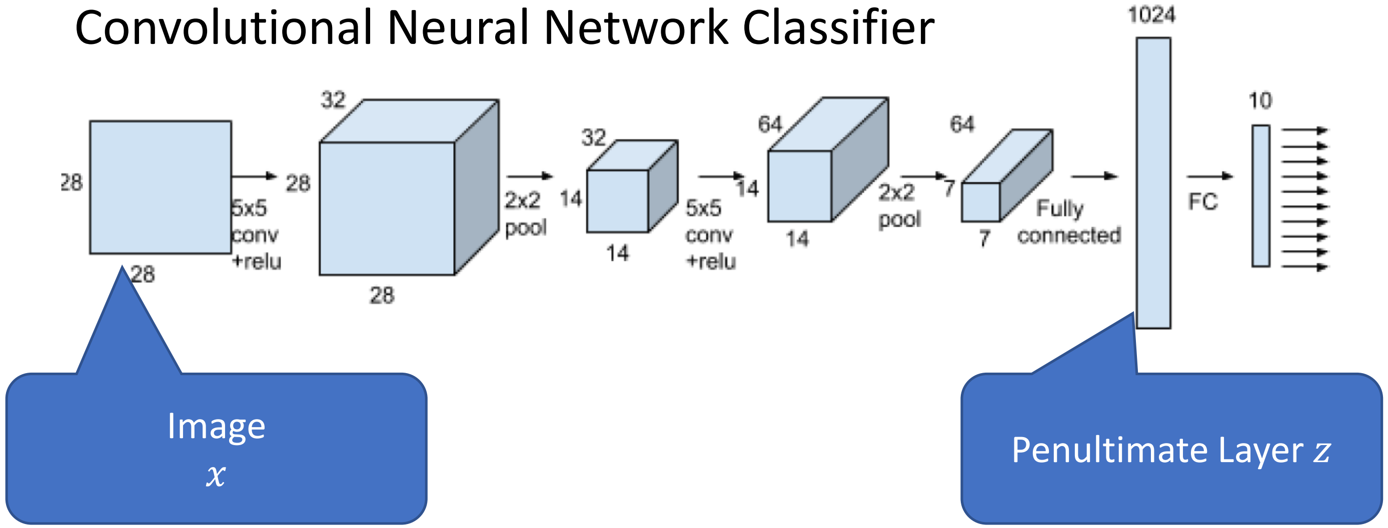


- Hendrycks & Gimpel (ICLR 2017) “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”

Methods (1): Classifier-Based Approaches: Probability models

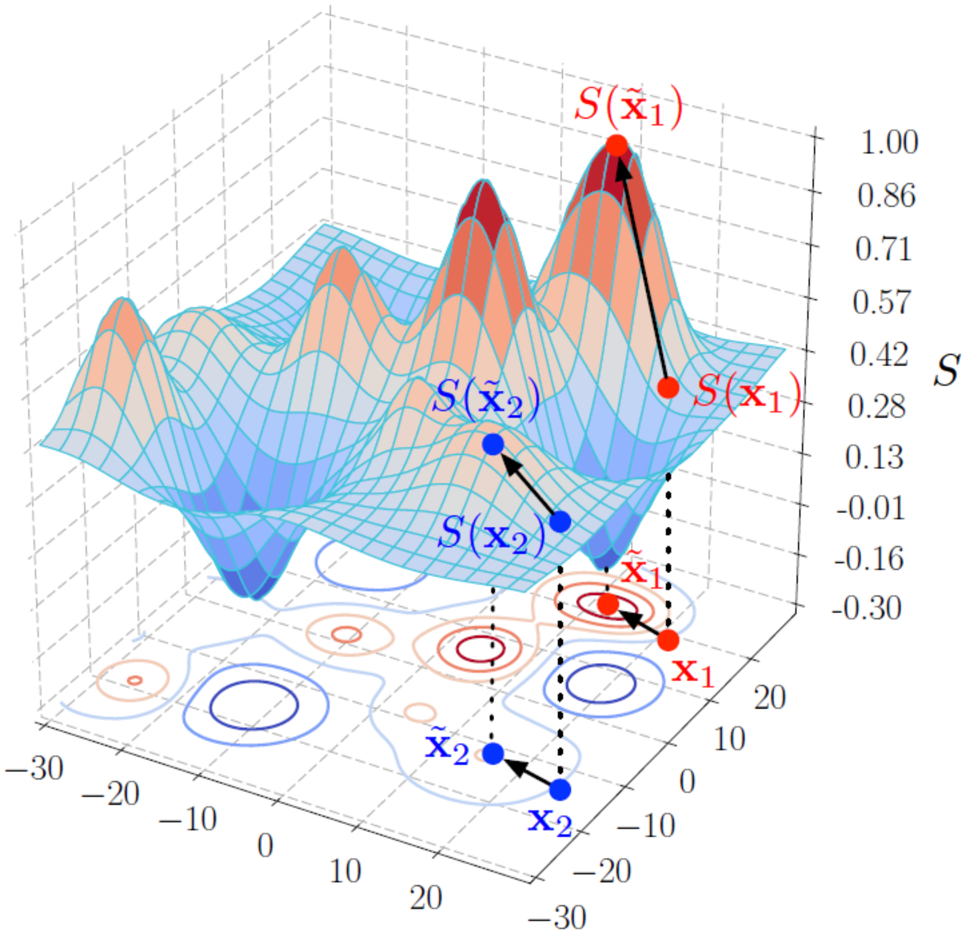
- **Use Penultimate Layer as a Latent Representation**
- Classifier is computed as
  - $z = E(x)$ "penultimate layer"
  - $P(y|x) = \mathrm{softmax}(Wz)$
- Learn a probability model $P(z)$
  - Gaussian: $P(z) = \mathrm{Normal}(z; \mu, \Sigma)$
  - Gaussian mixture model
  - $A(x) = -\log P\left(E\left(x_q\right)\right)$

Convolutional Neural Network Classifier



Methods (1): Classifier-Based Approaches: Perturbation

- **Score After Perturbation**
- Instead of scoring $A(x_q)$, score $A(x_q + \Delta x_q)$, where $\Delta \mathrm{x_q}$ is a fixed-step perturbation to move x_q "toward" the nearest class
- If x_q is near to an existing class, then the perturbation can reduce the anomaly score substantially
- For anomalies, the perturbation has little effect: $A(x_q) \approx A(x_q + \Delta x_q)$
- We can view this as approximately measuring distance from x_q to the peak of the nearest class

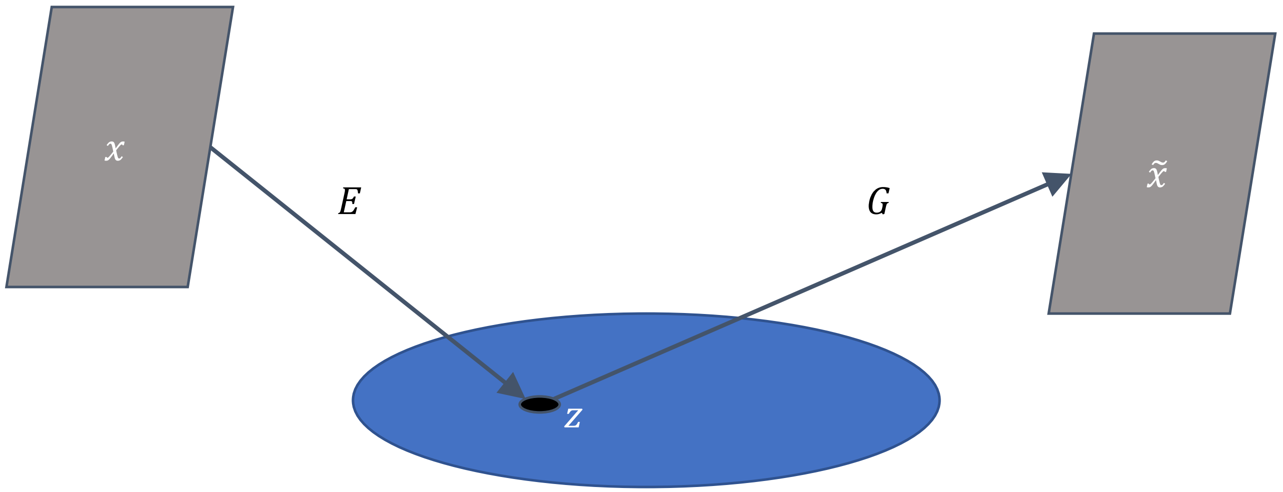


● In-distribution image
● Out-of-distribution image

Liang, Li, & Srikant. http://arxiv.org/abs/1706.02690

Methods (2): Autoencoders

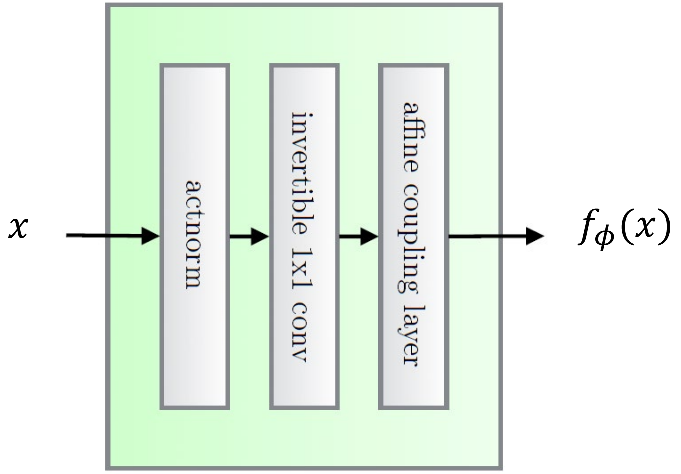
- Images lie in a lower-dimensional manifold. Can we discover it?
  - $z = E(x)$
  - $\tilde{x} = G(z)$
  - Autoencoder; VAE; GAN with Encoder
- Train an Auto-Encoder $z = E(x); \tilde{x} = G(x)$
  - $A(x_q) = \|x_q - \tilde{x}_q\|$ "reconstruction error"
  - $A(x_q) = -\log P(z_q)$ "latent probability"
- Results
  - Reconstruction error does not perform well
  - Latent probability models are mediocre



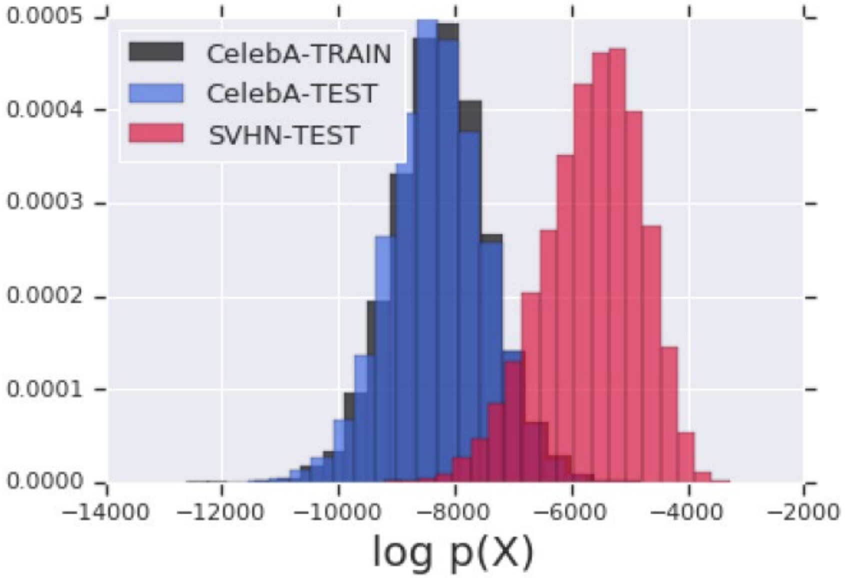
Methods (3): Normalizing Flow Models for $P(x)$

Normalizing Flows

- Define an invertible function $f_\phi(x)$. Several general, non-linear neural network-style functions have been developed
- Convert density over $f_\phi(x)$ into a density over x via Jacobian normalization
- $\log P(x) = \log P\left(f_\phi(x)\right) + \log \left|\frac{\partial f_\phi(x)}{\partial x}\right|$
- Anomaly score $A(x_q) = -\log P(x_q)$
- Example: Kingma & Dhariwal (2018). GLOW: Generative flow with invertible 1x1 convolutions



- Experiments show that these models often assign high probability density to images outside the training distribution, so they fail as anomaly detectors
    - Nalisnick, E., Matsukawa, A., Teh, Y. W., Gorur, D., Lakshminarayanan, B., To, N., Deep, N., & Inference, A. B. (2019). Do Deep Generative Models Know What They Don't Know?



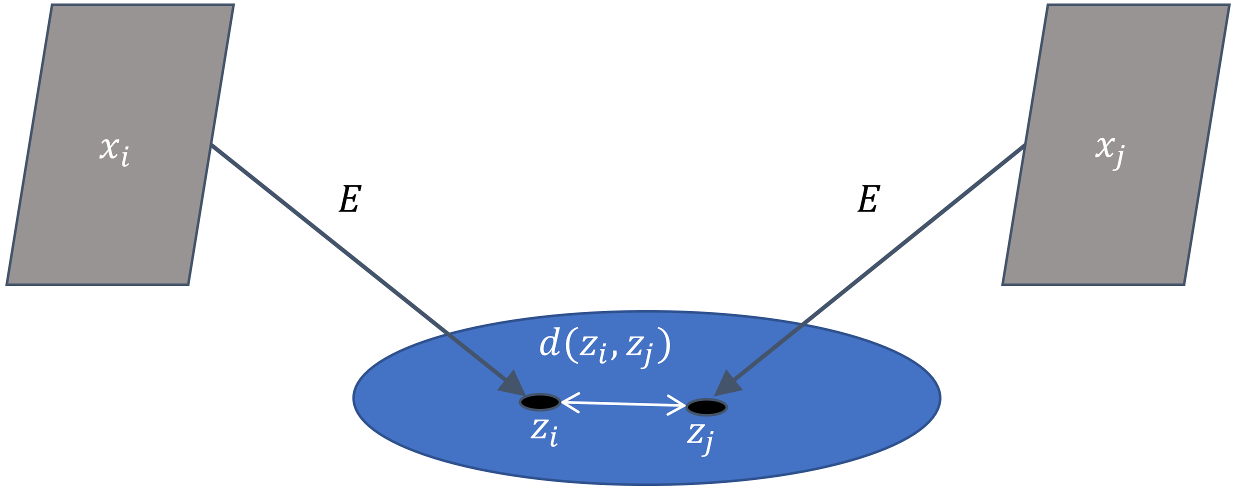
Nalisnick, Matsukawa, et al. (ICLR 2019)

Methods (4): Distance Functions

Learn an embedding $\boldsymbol{z} = \boldsymbol{E}(\boldsymbol{x})$ that has good distance properties

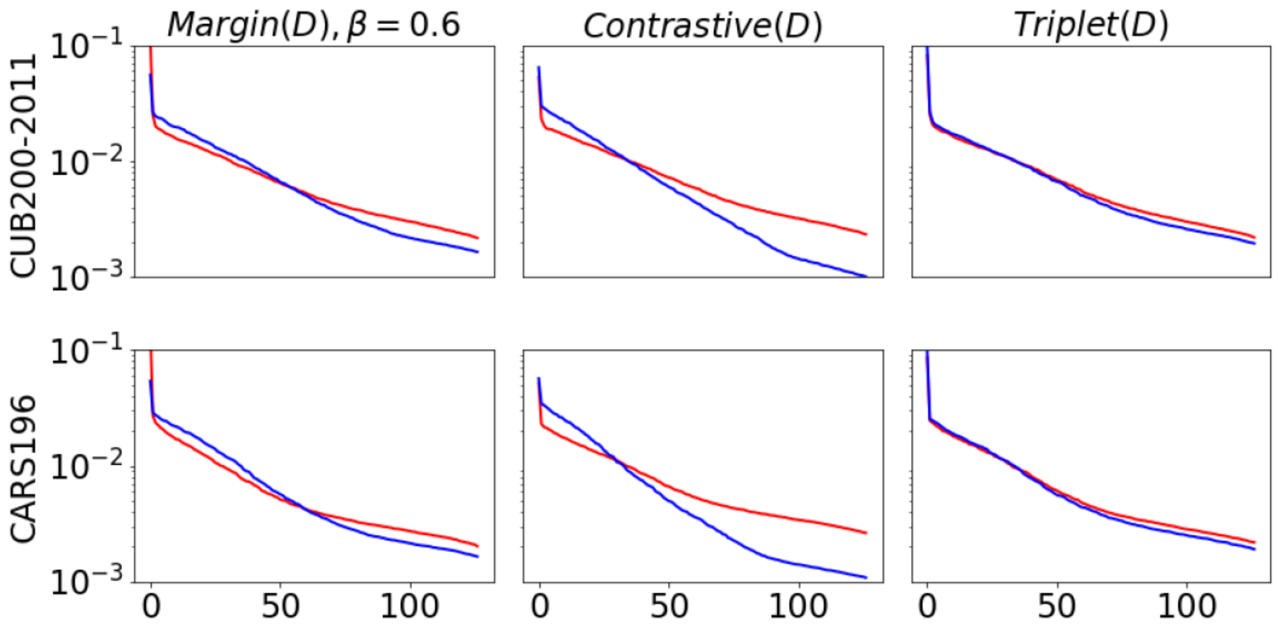
- $d(x_i, x_j) = \|E(x_i) - E(x_j)\|$
- Good nearest-neighbor classifier
- Captures some notion of perceptual distance
- Triplet Loss, N-pairs Loss, etc.
- $A(x_q) = \min_i d(x_q, x_i)$

- We have not been able to get this to work well



Distance Metric Learning with Spectral Regularization

- Roth, et al. (2020) "Revisiting Training Strategies and Generalization Performance in Deep Metric Learning"
- Optimize both accuracy and the entropy of the singular value spectrum
  - Compute SVD of the latent space embedding points
  - Sort the singular values in decreasing order
  - Encourage the curve to be flatter



Spectral Regularization Results

- Gives best classification results to date on three benchmark datasets
- Maybe it will work for anomaly detection?

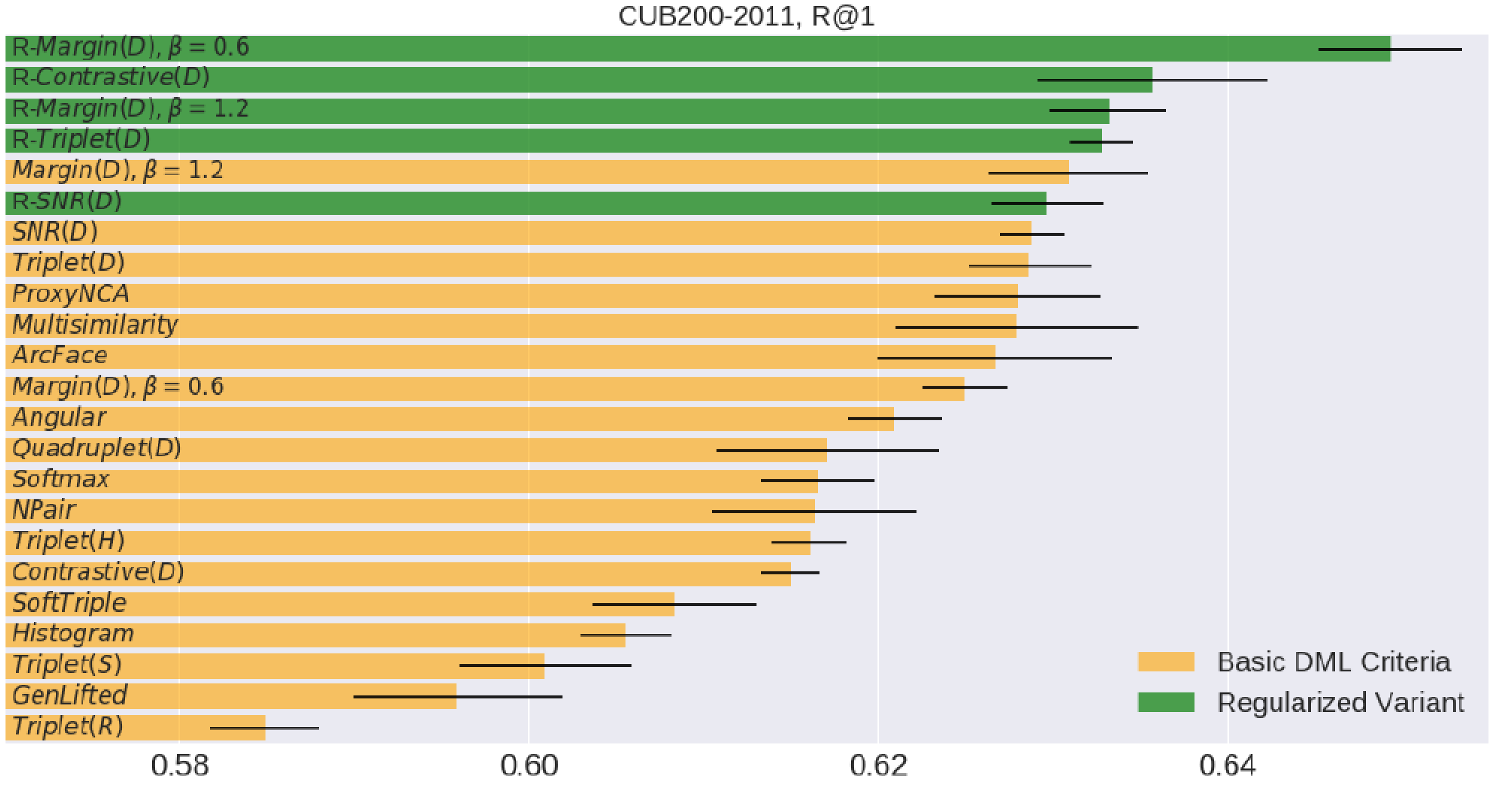
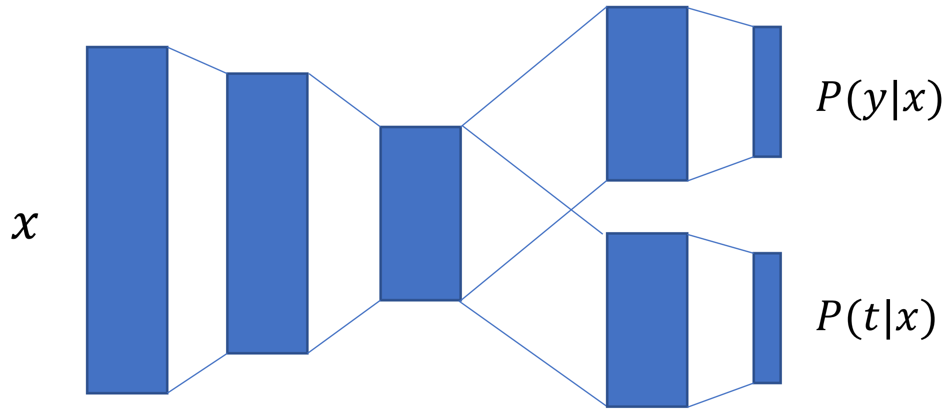
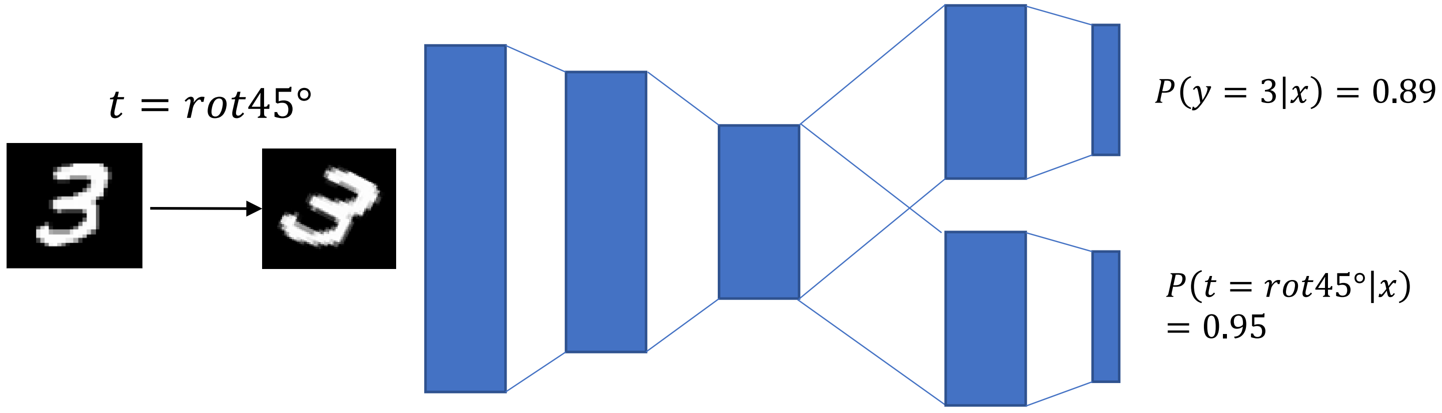


Figure 1. Mean recall performance and standard deviation of various DML objectives trained with (green) and without (orange) our proposed regularization.

Methods (5): Auxiliary Tasks

- **Define Auxiliary Self-Supervised Tasks**
- Train the network to perform the primary task and the auxiliary tasks
  - $t \in \{t_1, \ldots, t_k\}$
- Example tasks
  - Image rotation by $45^\circ, 90°, 135°, 180°$
  - Image flipping
  - Affine distortions
- Given a test query x_q, transform it according to each transformation and see whether $P(t|x_q)$ predicts the correct transformation
  - If Yes: trust the classifier
  - Else: declare an anomaly
- These tasks should require understanding the contents of the image
- Advantage: Avoids some nuisance novelty





Golan & El-Yaniv (2018). Deep Anomaly Detection Using Geometric Transformations. ArXiv: 1805.10917
Bergman & Hoshen (2020). Classification-Based Anomaly Detection for General Data. ICLR 2020

Concluding Remarks

- **Anomaly detection is important**
  - Critical for robust AI systems
  - Practical applications
- **Anomaly detection is difficult**
  - Moderately mature for tabular data sets
  - Fundamentally relies on some notion of distance
  - Very challenging for images where we need a notion of semantic distance
- **Research in this area is advancing rapidly with little theoretical understanding**