Lecture 7 – Supervised Learning

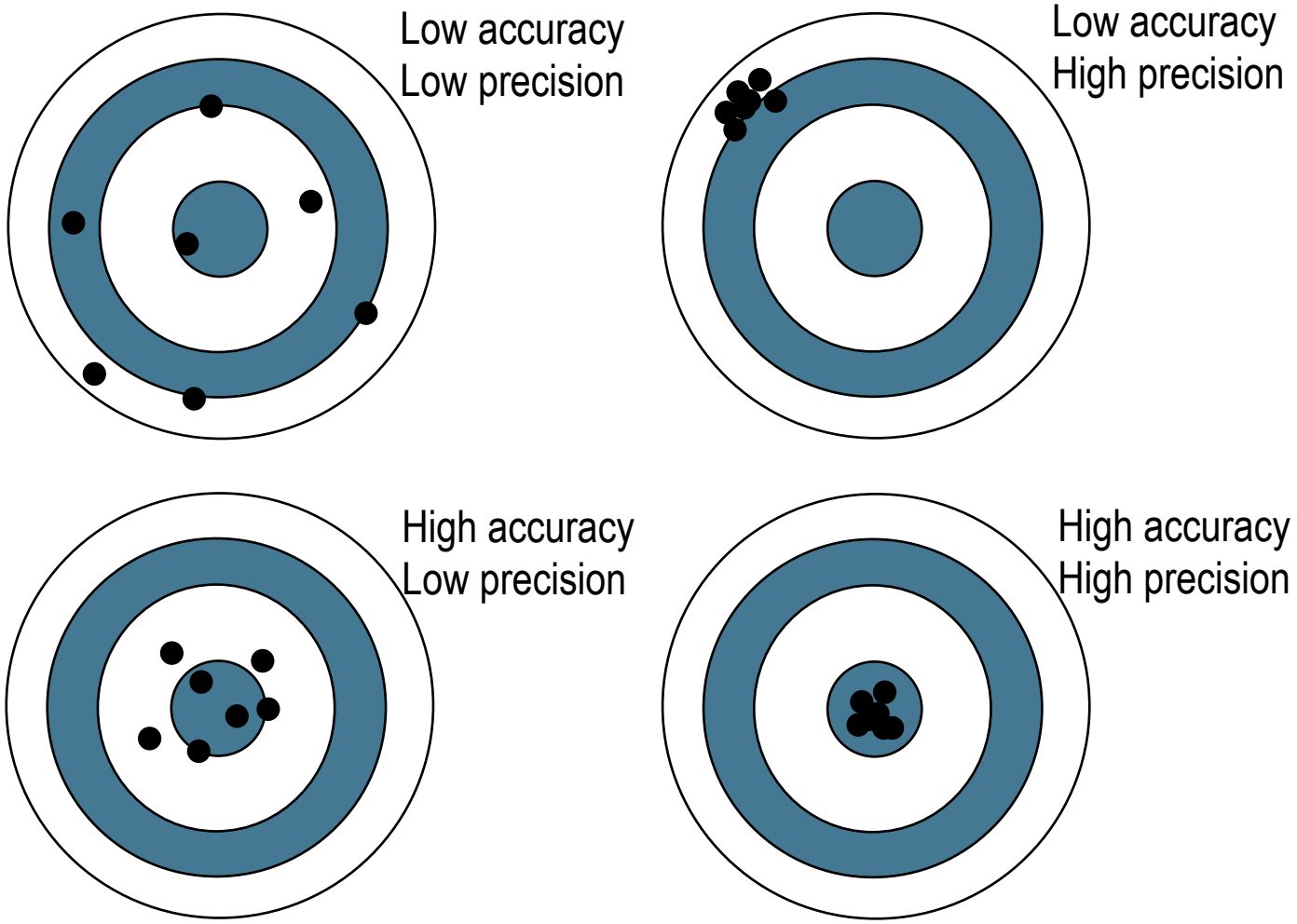
Classification models (continued), Ensemble methods

Information Systems for Sustainable Society (is3) | WiSo Faculty | Univ.-Prof. Dr. Wolfgang Ketter | 11/28/22

Agenda

- Evaluating Classification Models
- Classification and Regression Trees in Practice
- Non-linear Classification
- Ensemble Methods

Precision vs. Accuracy



- **Accuracy**: ratio of correctly classified samples over the full sample size – Measure of how well the classifier performs in general
- **Precision**: ratio of true positives versus all positives as classified by the classifier – Measure of how precisely the classifier identifies positives

Classification metrics

- So far, we have considered accuracy (0/1 loss) as the primary method for evaluating classifiers
- However, sometimes the benefits for correctly classifying positive and negative examples are different, as are the costs for predicting a positive example to be negative, and vice versa
- In cancer dataset, it is a very different thing (in terms of real-world effects) to predict that an actually malignant tumor is benign, versus predicting a benign tumor is malignant

Confusion matrix

- A **confusion matrix** explicitly lists the number of examples for each actual class and each prediction
- Can compute these (and all associated metrics) on training / holdout / testing sets, but we'll just show examples on training sets here

Confusion matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

Common classification error metrics derived from the confusion matrix

- Several common metrics are associated with entries of the confusion matrix (TP = true positive, FP = false positive, TN = true negative, FN = false negative)
  - TP Rate (also called Recall) = $\frac{TP}{TP+FN}$
  - FP Rate = $\frac{FP}{FP+TN}$
  - Precision = $\frac{TP}{TP+FP}$
  - Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- Different metrics can be standard for different domains

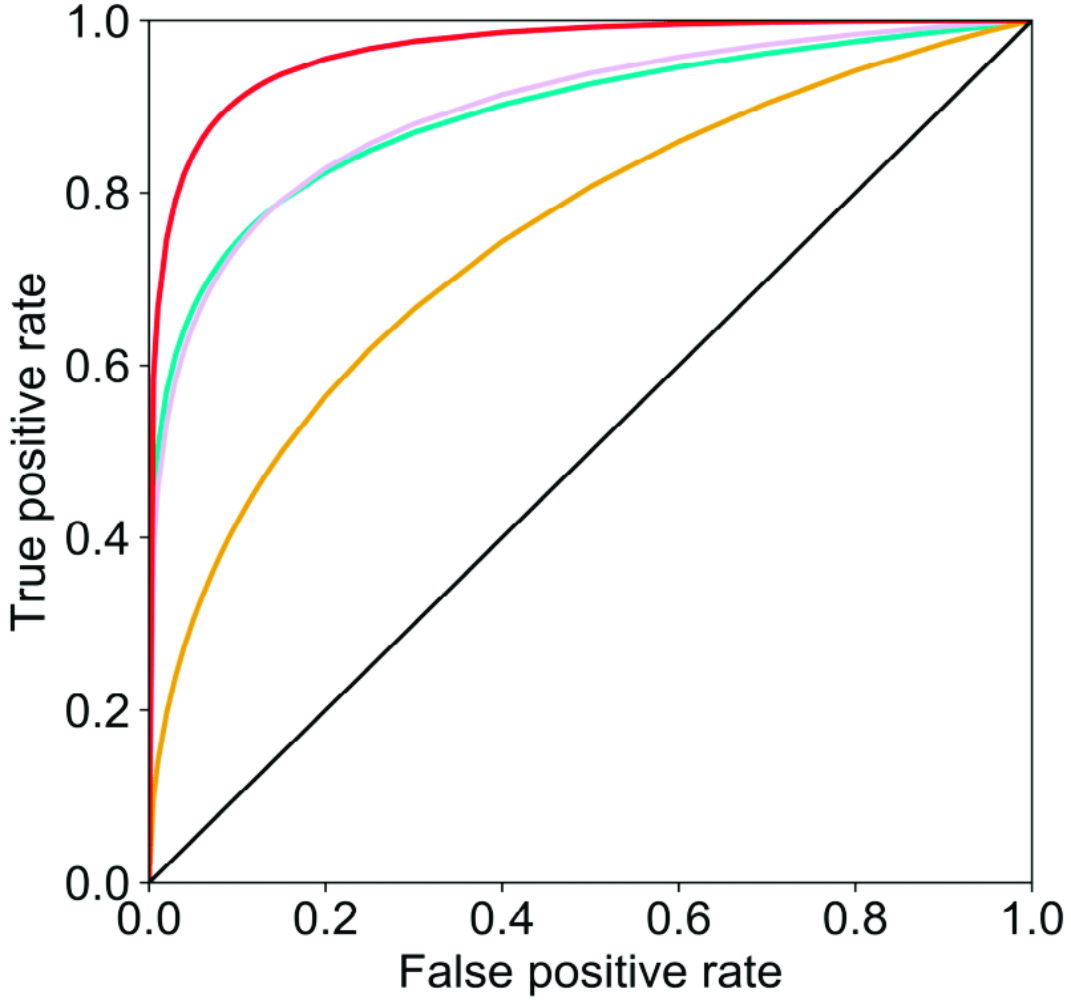
Confusion matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

Changing the prediction threshold

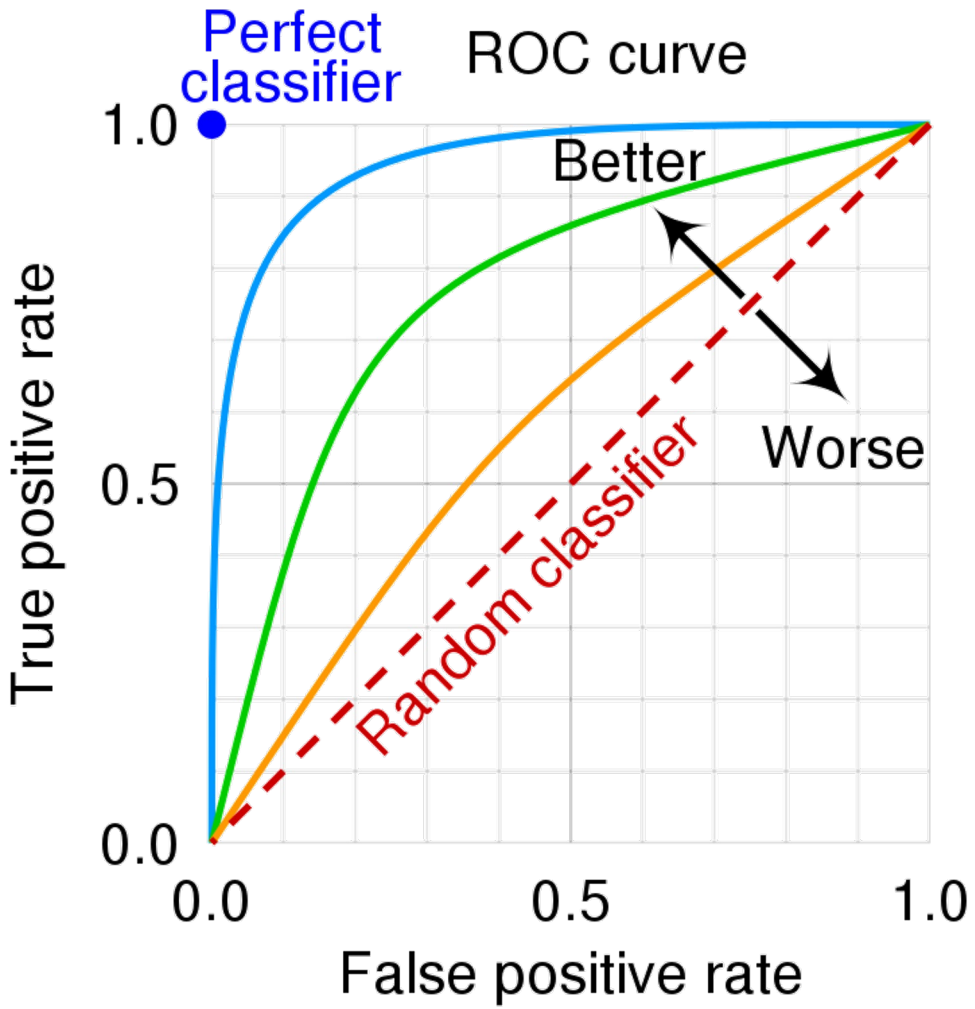
- Classifiers are implicitly trained around a “threshold” of zero (positive hypothesis means predict positive, negative means predict negative)
- But there is no reason to use only this threshold when we want to make predictions (may want to “overpredict” one class or the other)
- **Key idea:** by sorting the hypothesis function outputs, and adjusting the threshold at which we call something positive or negative, we can sweep out **all** possible classifications that a classifier can produce

ROC Curve



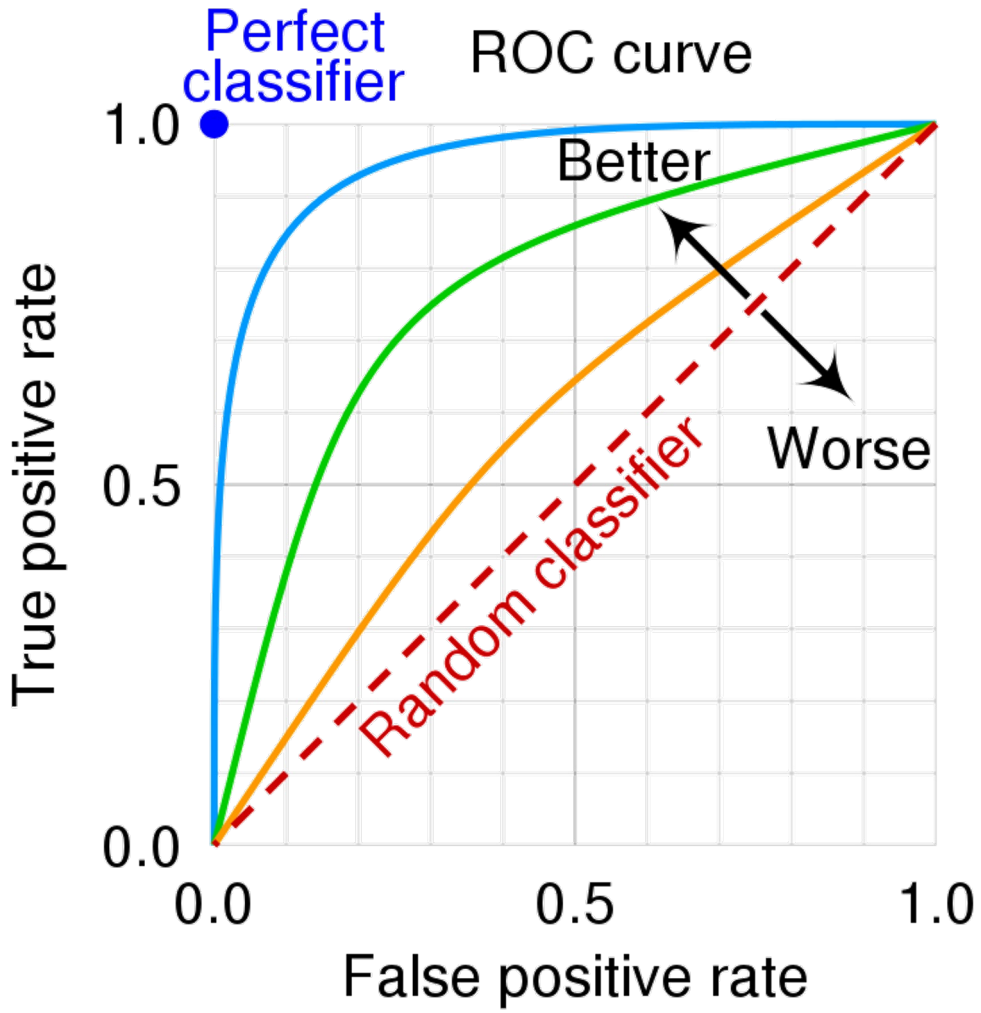
- A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
- A ROC curve is constructed by plotting the true positive rate (TP) against the false positive rate (FP).

ROC Curve



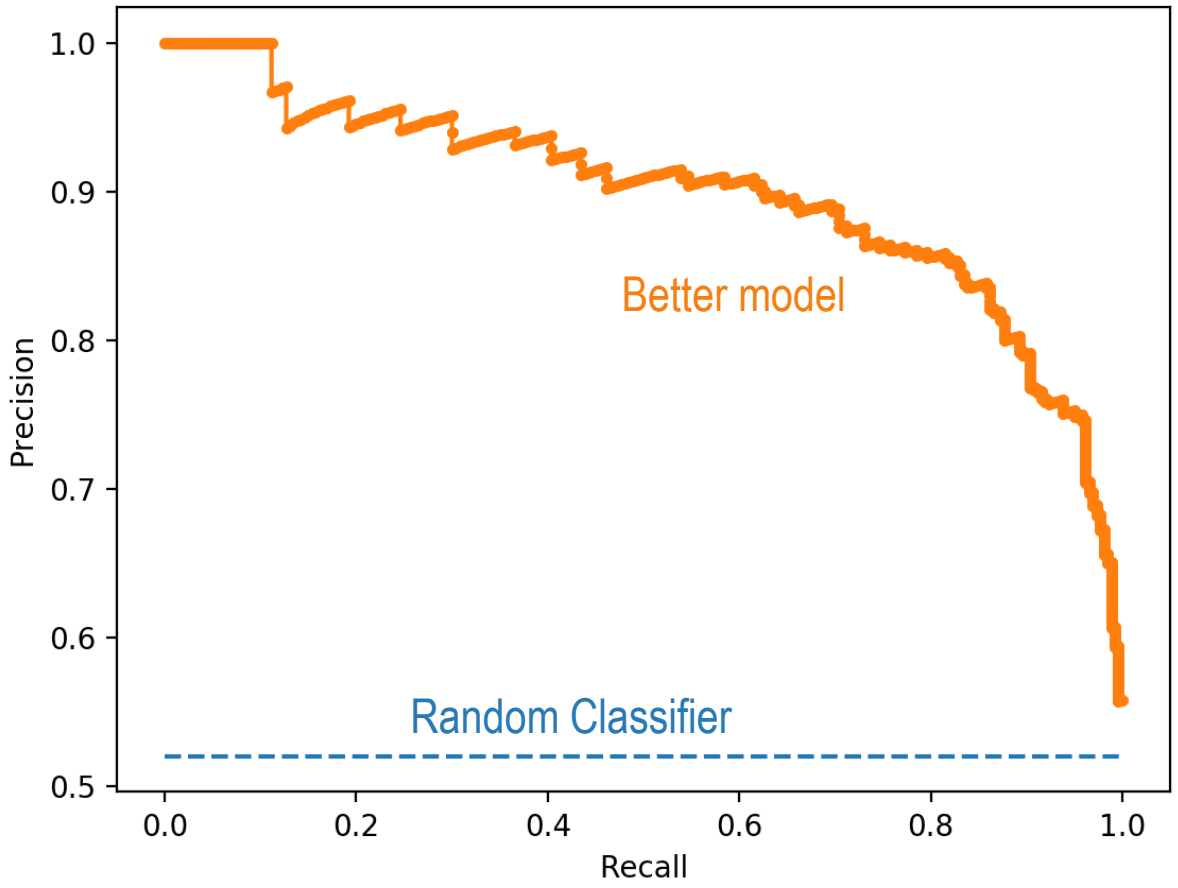
- A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.
- A ROC curve is constructed by plotting the true positive rate (TP) against the false positive rate (FP).

ROC Curve



- The ROC curve shows the trade-off between sensitivity (or TP) and specificity (1 – FP). Classifiers that give curves closer to the top-left corner indicate a better performance.
- As a baseline, a random classifier is expected to give points lying along the diagonal (FP = TP). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

Precision recall curves

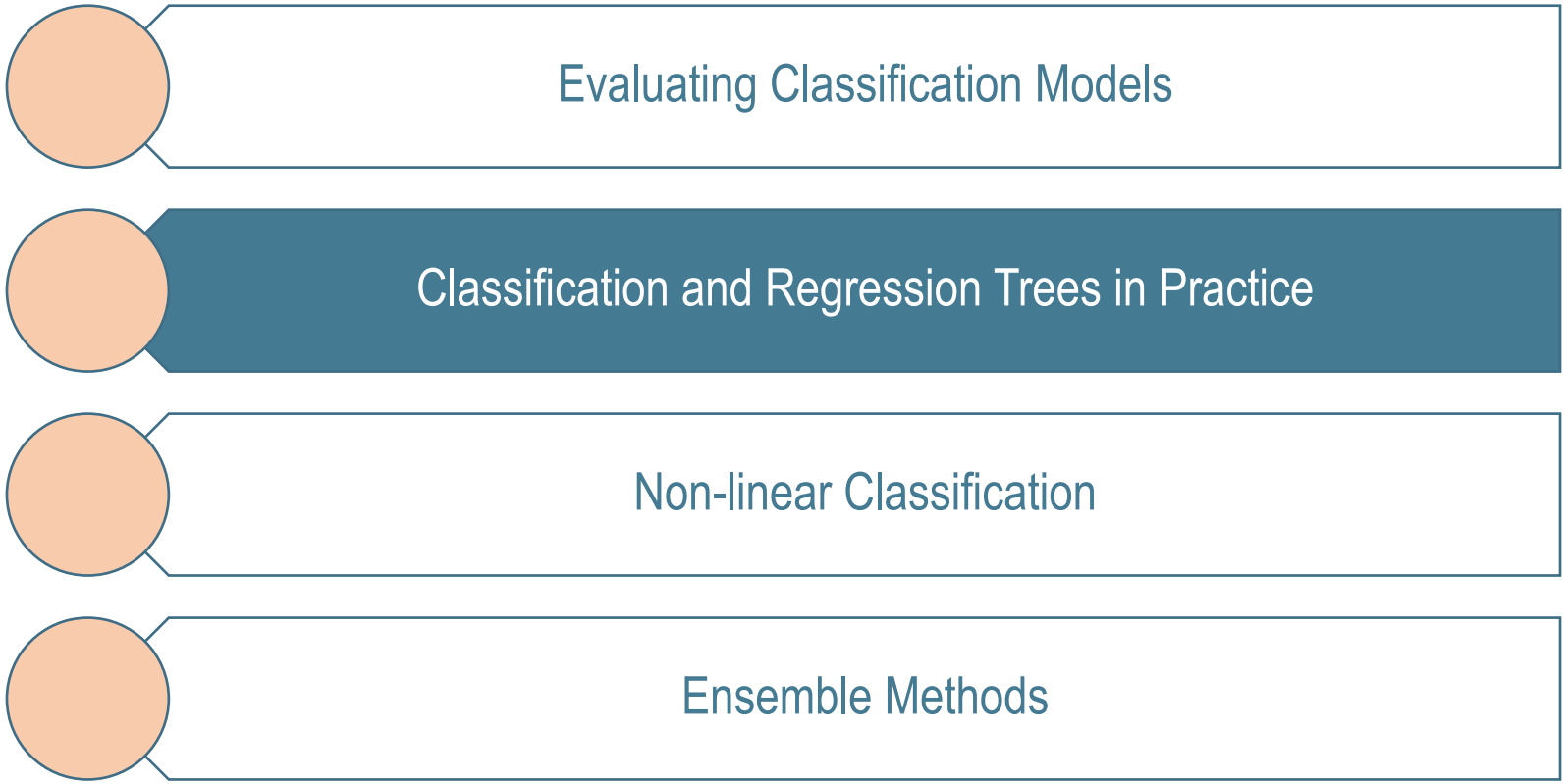


- The precision-recall curve shows the tradeoff between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate.
- Where
  - Recall = TP rate
  - Precision = TP/(TP+FP)

When to Use ROC vs. Precision-Recall Curves?

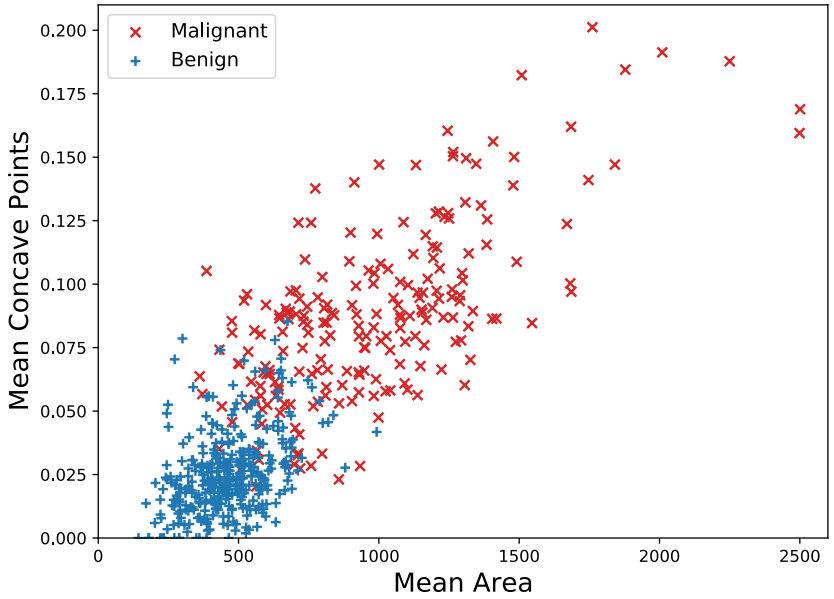
- Generally, the use of ROC curves and precision-recall curves are as follows:
  - ROC curves should be used when there are roughly equal numbers of observations for each class.
  - Precision-Recall curves should be used when there is a moderate to large class imbalance.
- The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance.
- Some go further and suggest that using a ROC curve with an imbalanced dataset might be deceptive and lead to incorrect interpretations of the model performance.
- The main reason for this optimistic picture is because of the use of true negatives in the False Positive Rate in the ROC Curve and the careful avoidance of this rate in the Precision-Recall curve.

Agenda

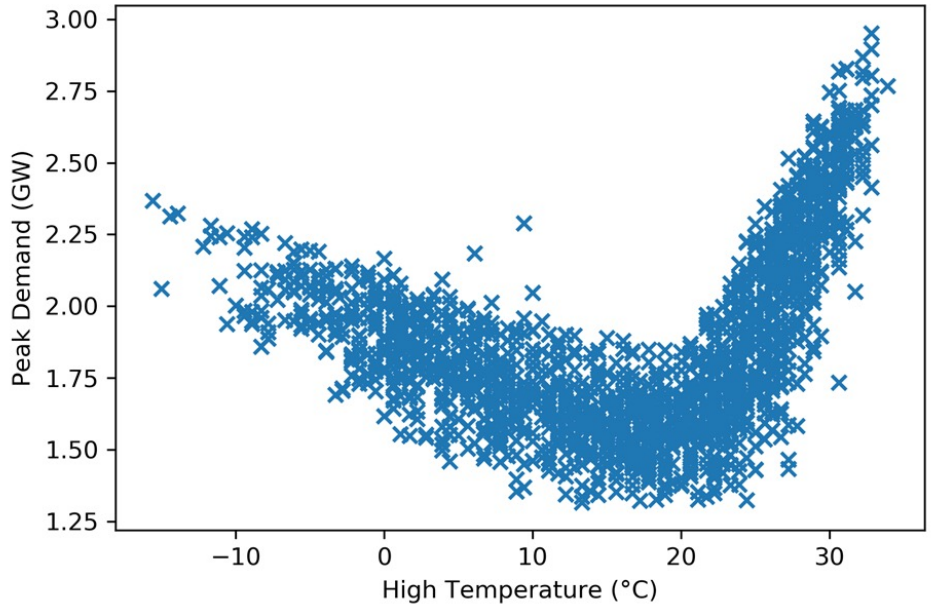

Evaluating Classification Models
Classification and Regression Trees in Practice
Non-linear Classification
Ensemble Methods

Let us return to our two running examples for classification and regression to see how tree-based methods would work in these cases

Classification – Classifying Breast Cancer Cells

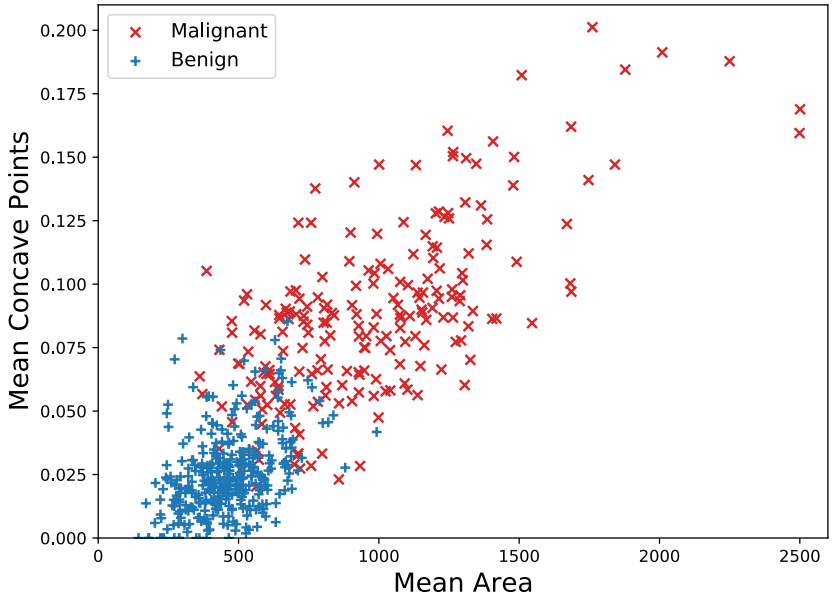


Regression – Predicting Peak Electricity Consumption

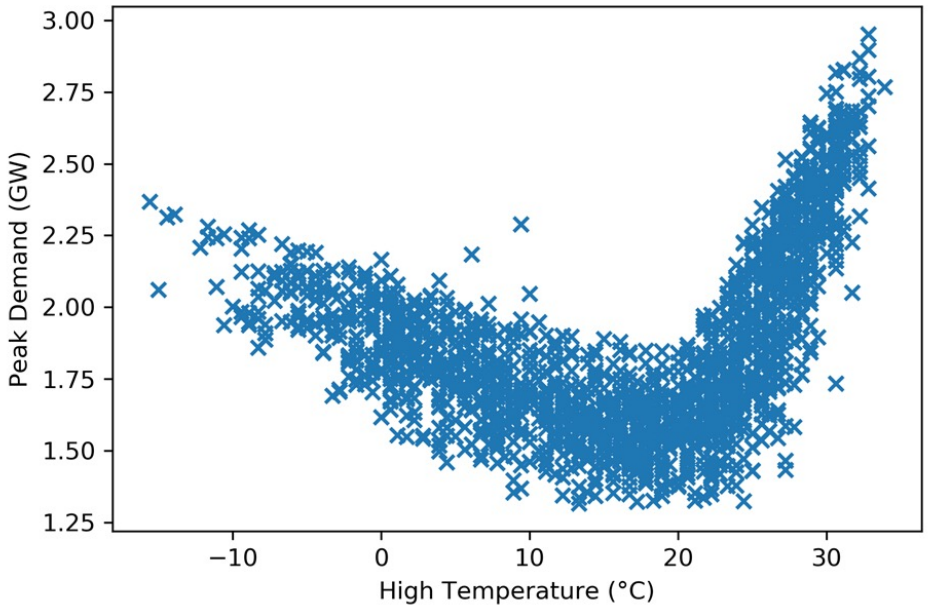


Let us return to our two running examples for classification and regression to see how tree-based methods would work in these cases

Classification – Classifying Breast Cancer Cells

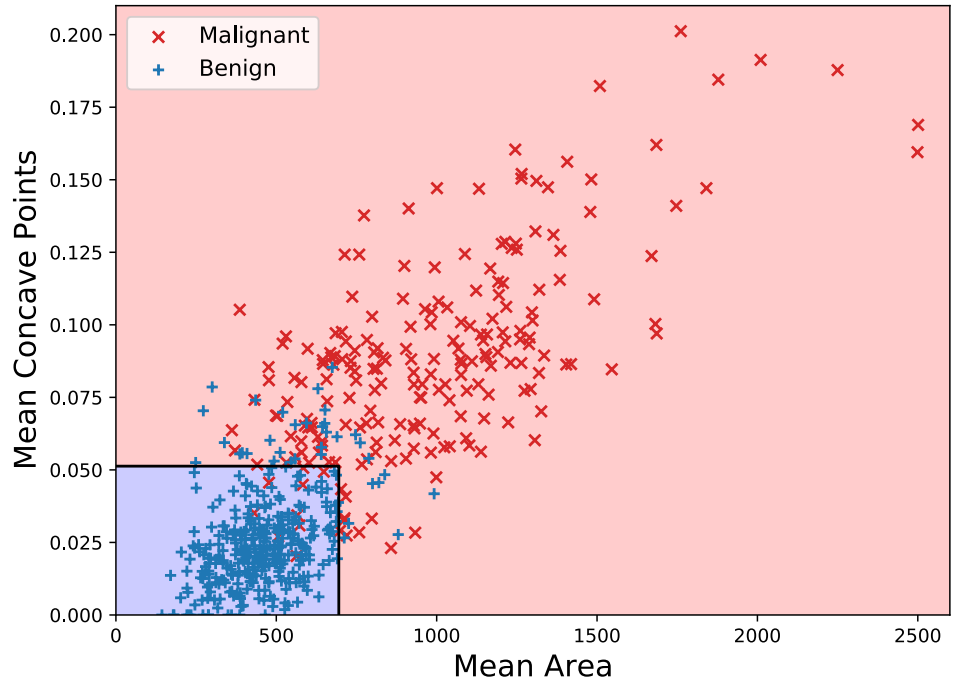


Regression – Predicting Peak Electricity Consumption

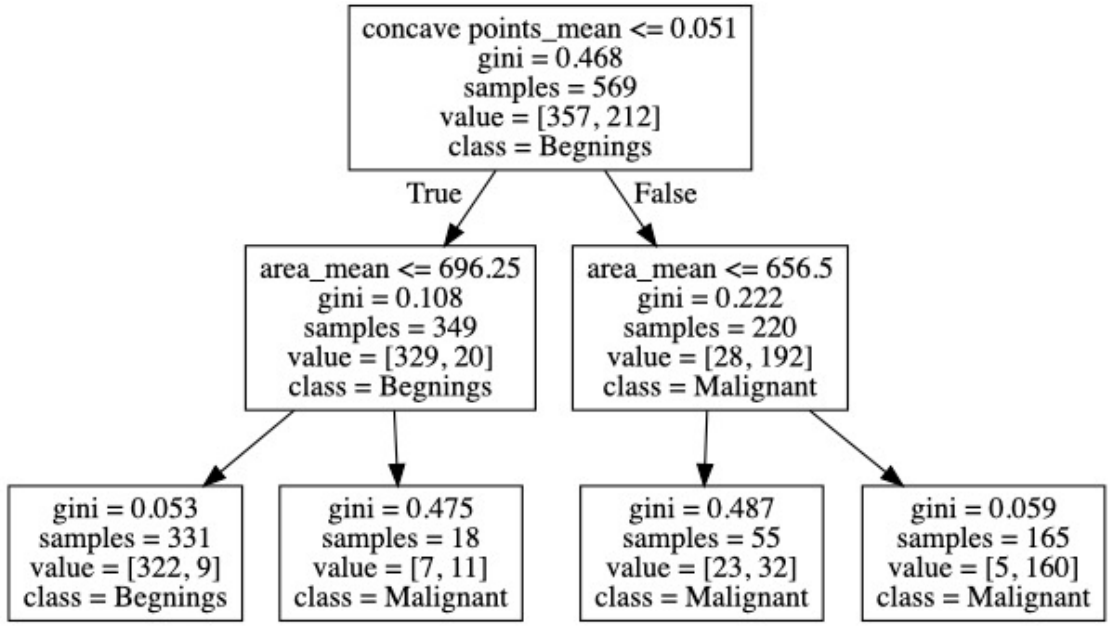


Let's look at a very simple tree architecture for the classification of breast cancer data (tree_depths=2, impurity_criterion=gini)

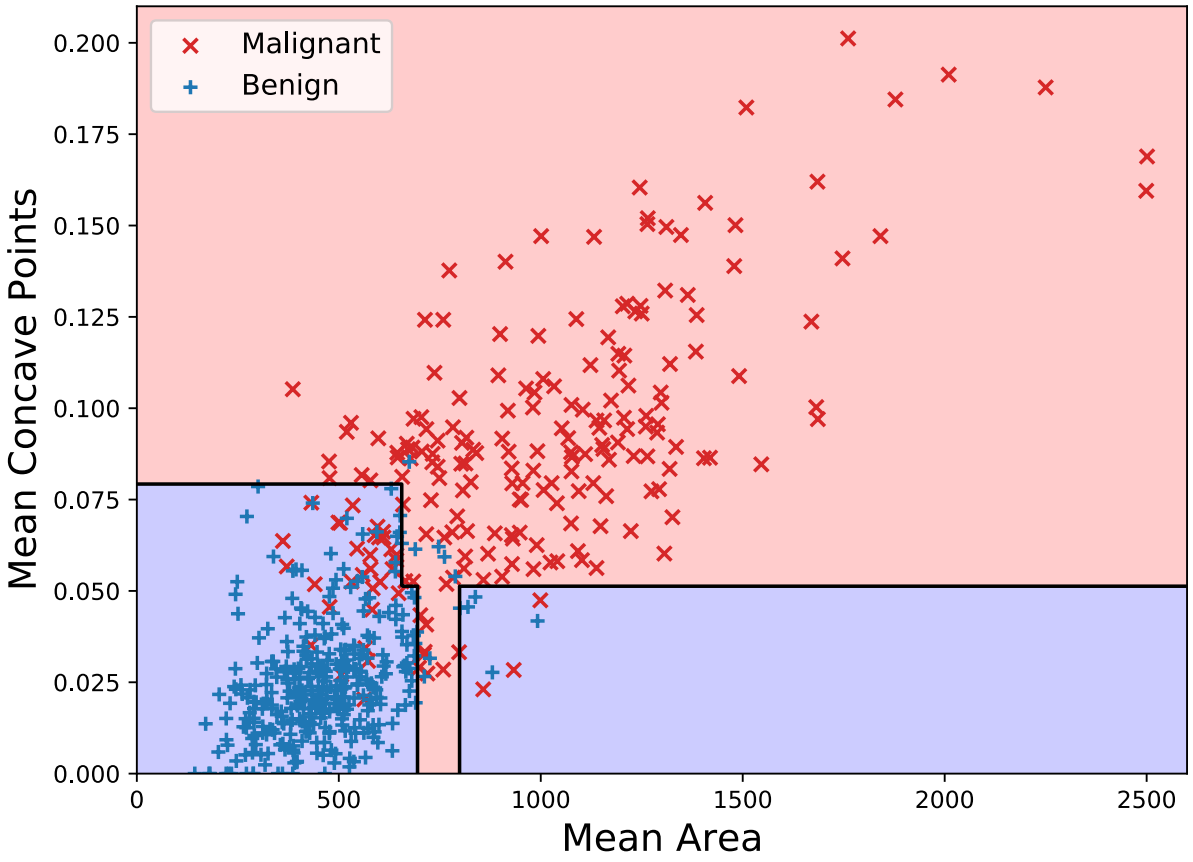
Decision surfaces



Decision Tree

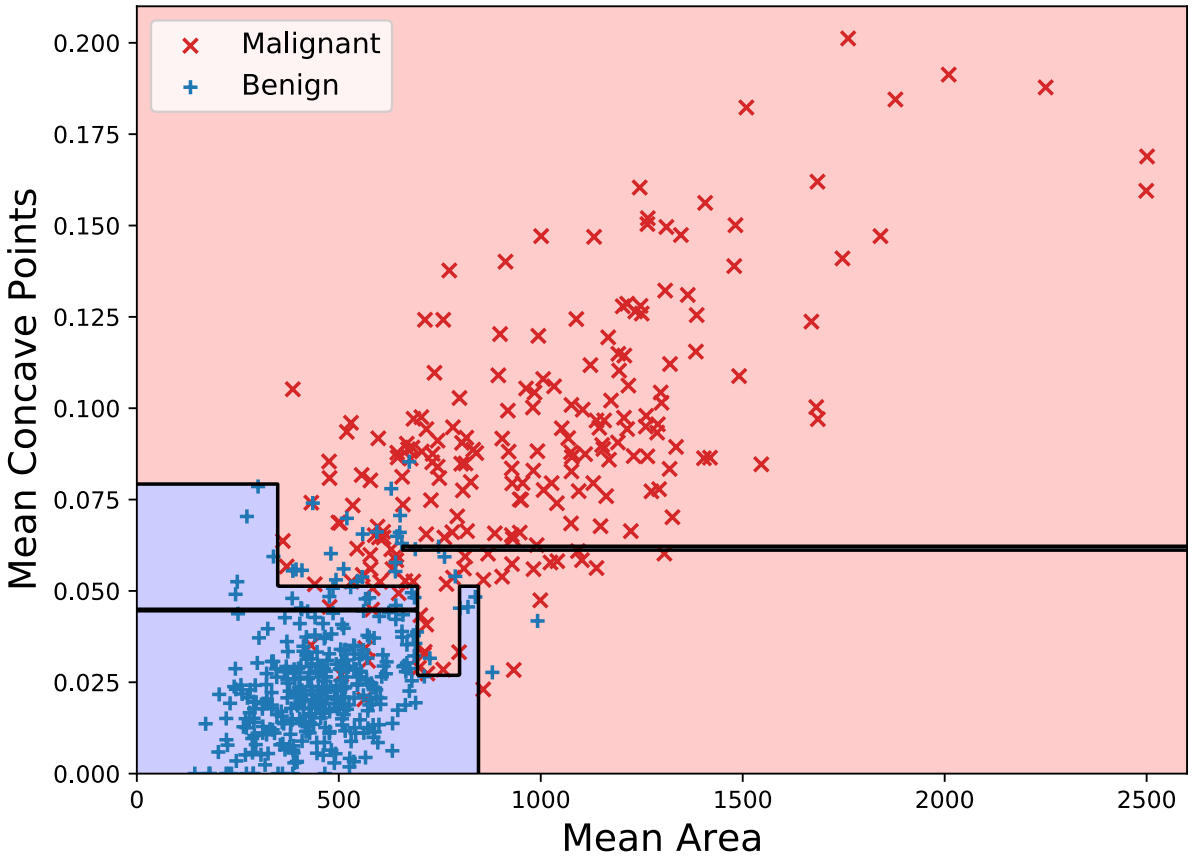


Fully grown trees overfit easily – Let's increase tree branching depth iteratively



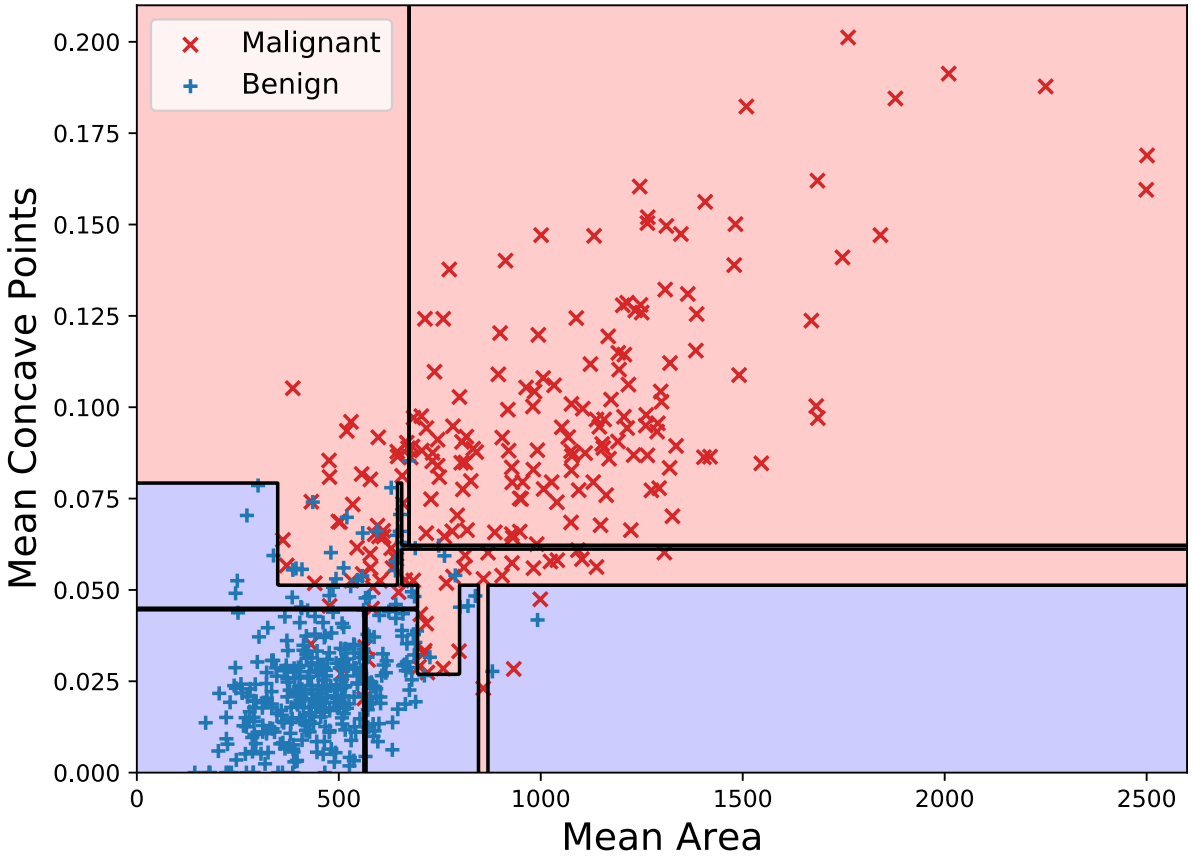
- **Tree depths = 3**
- Num nodes = 15
- Num leaves = 8

Fully grown trees overfit easily – Let's increase tree branching depth iteratively



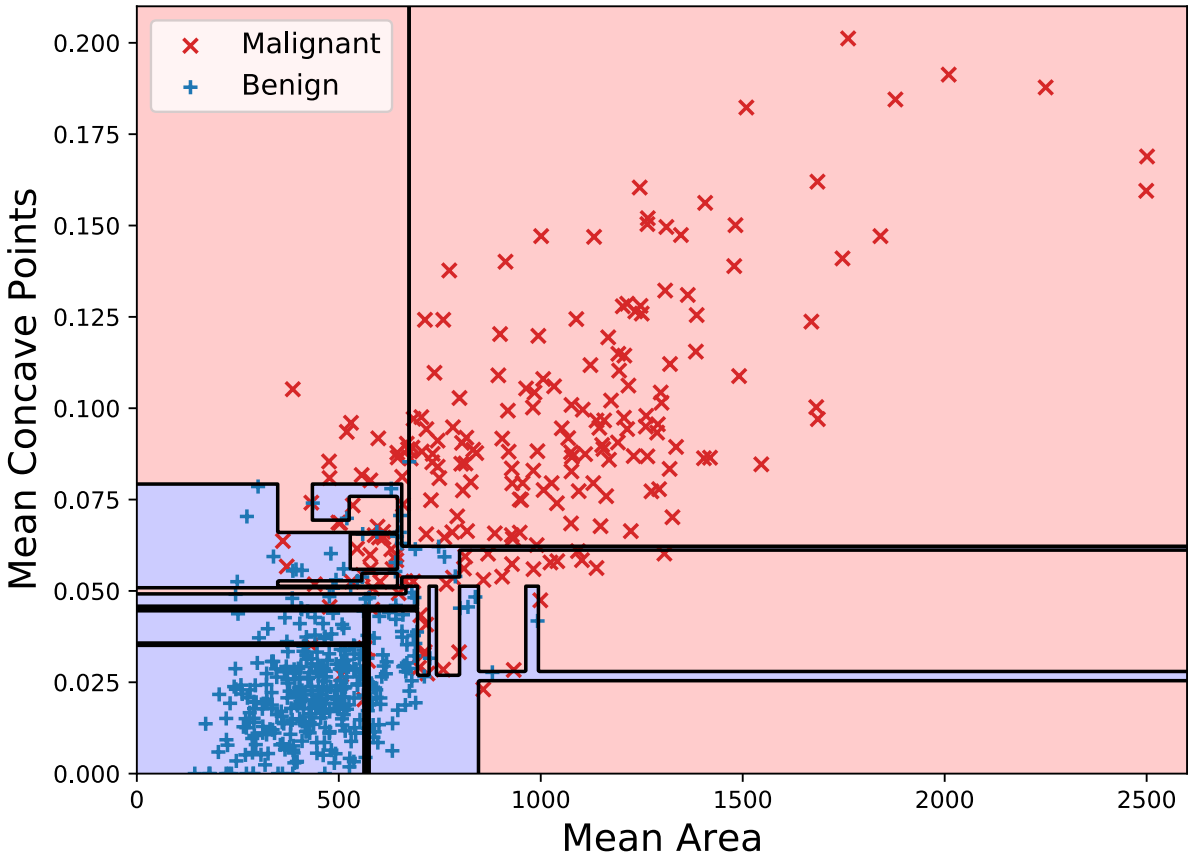
- **Tree depths = 4**
- Num nodes = 29
- Num leaves = 15

Fully grown trees overfit easily – Let's increase tree branching depth iteratively



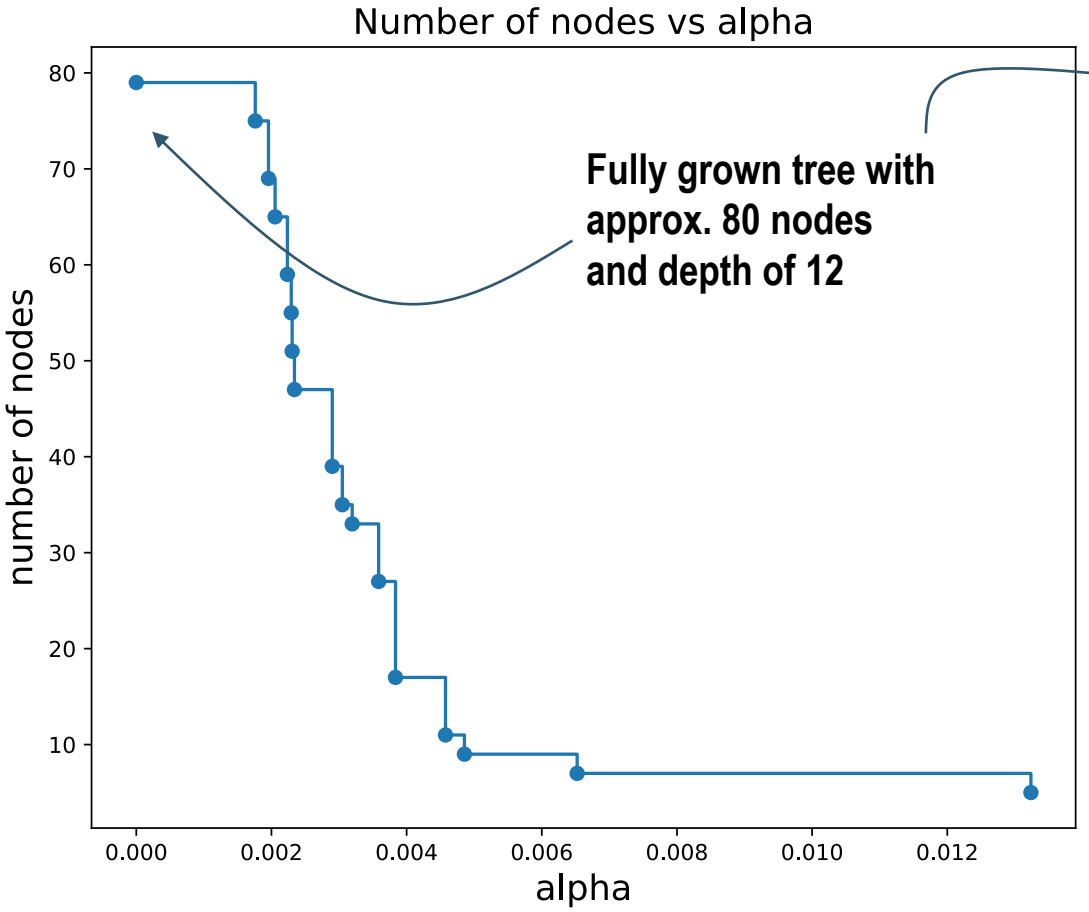
- **Tree depths = 5**
- Num nodes = 45
- Num leaves = 23

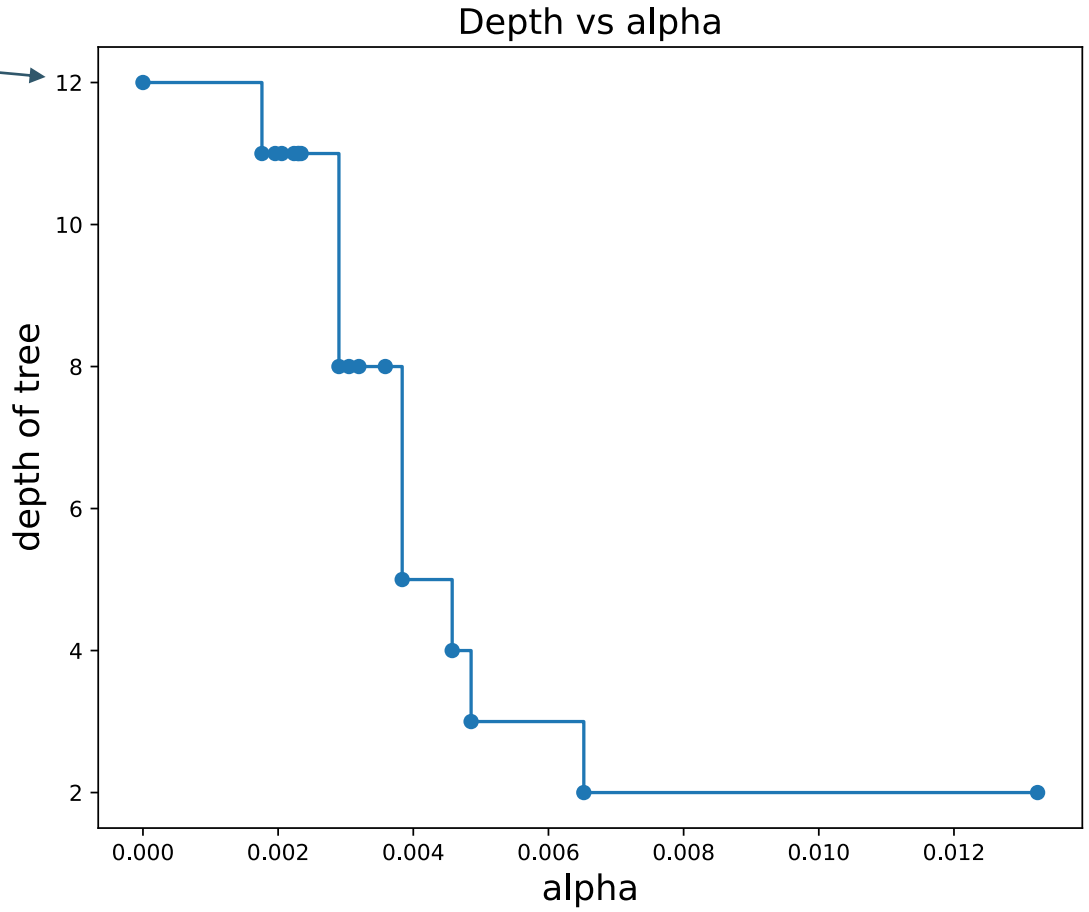
Fully grown trees overfit easily – Let's increase tree branching depth iteratively



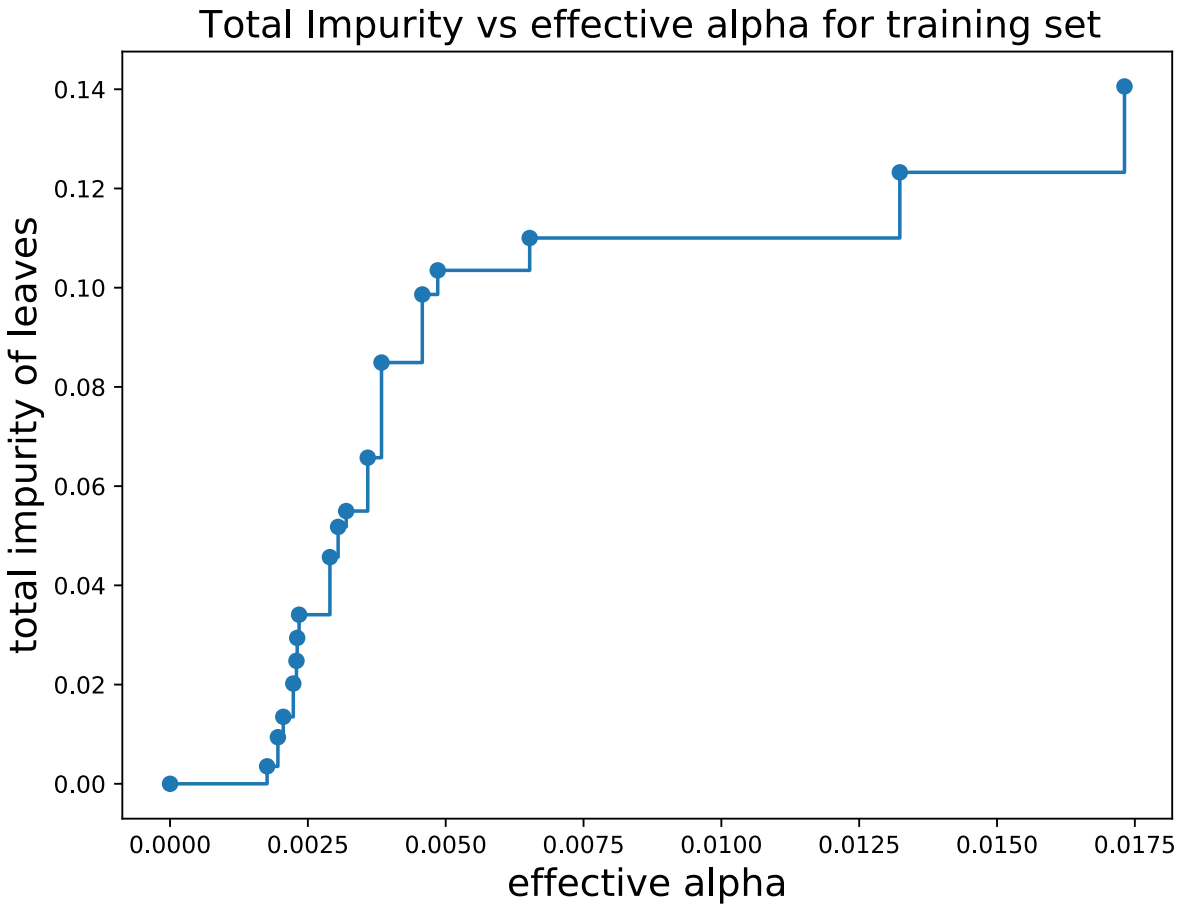
- **Tree depths = 10**
- Num nodes = 97
- Num leaves = 49
- Mechanisms such as **pruning**, **setting the minimum number of samples** required **at a leaf node** or setting the **maximum depth** of the tree are necessary to avoid the encountered overfitting

Pruning is effective in controlling tree complexity – Alpha is a parameter that controls the cost complexity (CC) metric (comp. regularization)

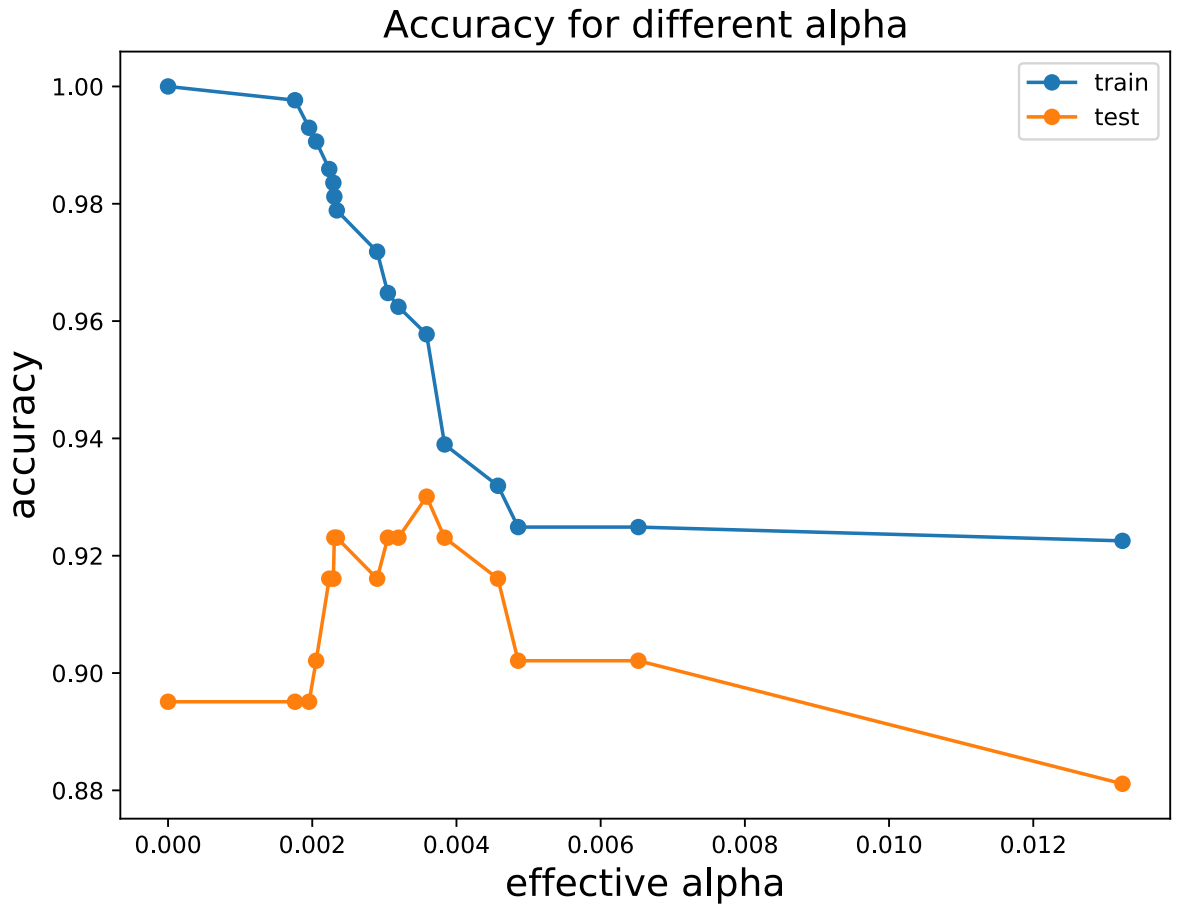




Naturally, pruning will induce impurity in the training data – Challenge is to find the sweet spot where generalization is optimized

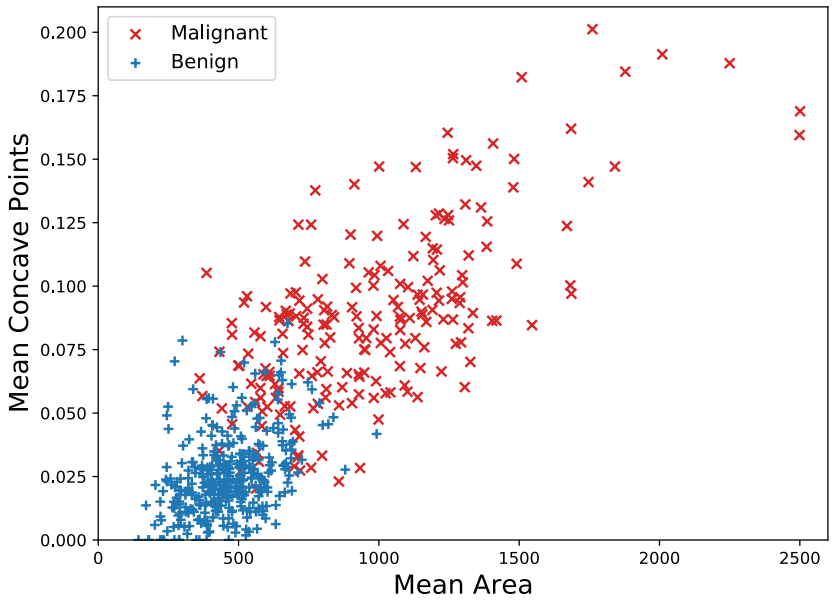


The sweet spot where generalization is optimized can be found via a grid search – Here we find the effective alpha of approx. 0.004 to yield the best results

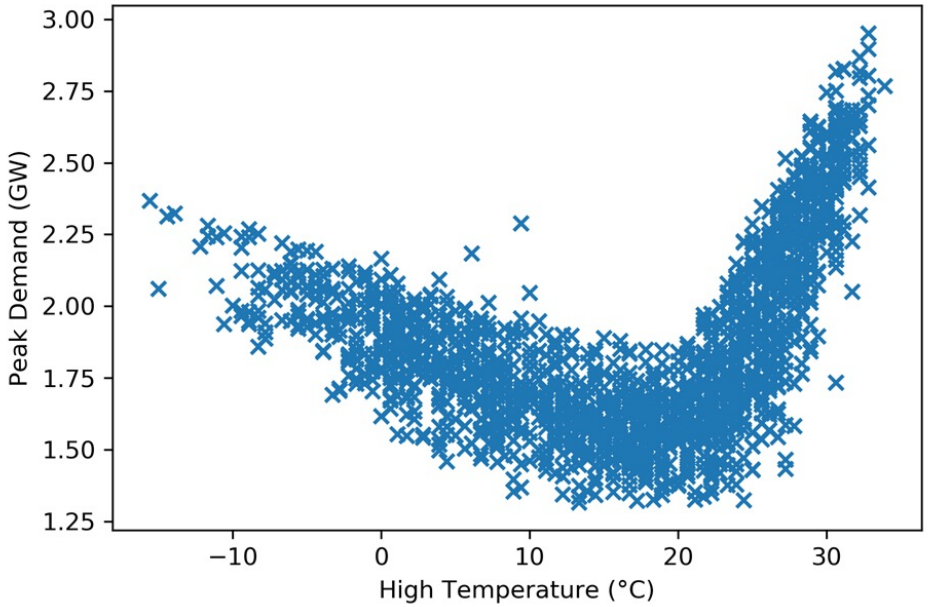


Let us return to our two running examples for classification and regression to see how tree-based methods would work in these cases

Classification – Classifying Breast Cancer Cells

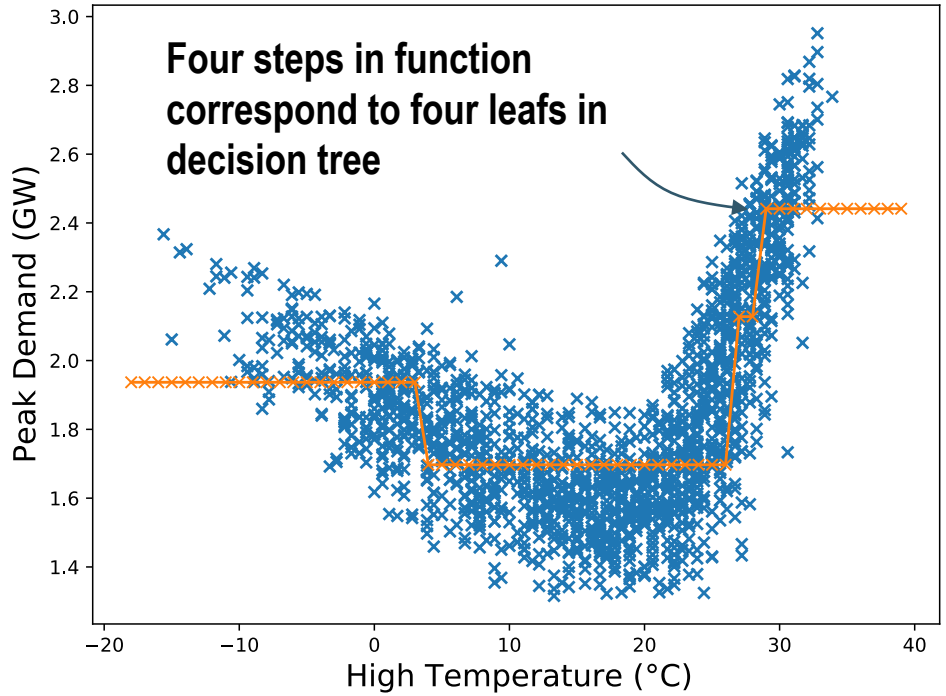


Regression – Predicting Peak Electricity Consumption

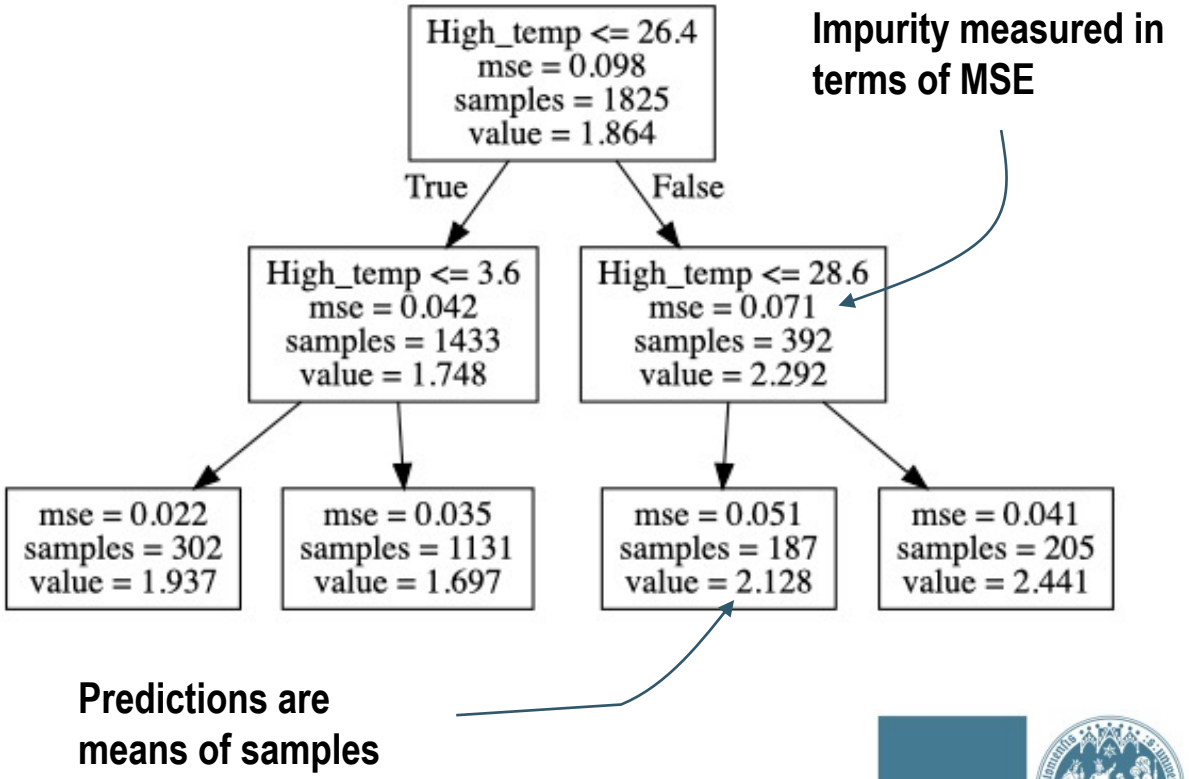


Let's look at a very simple tree architecture for the **prediction of peak electricity demand** (tree_depths=2, impurity_criterion=mse)

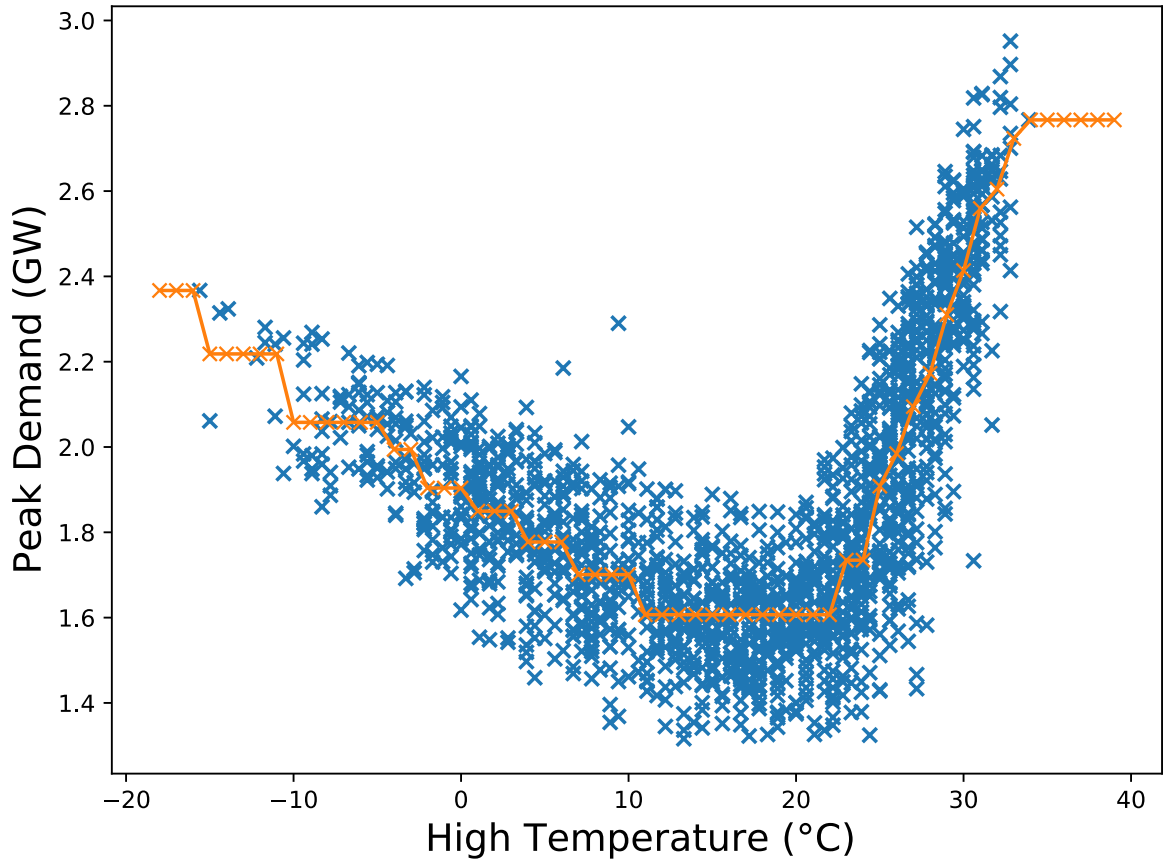
Regression Line



Decision Tree

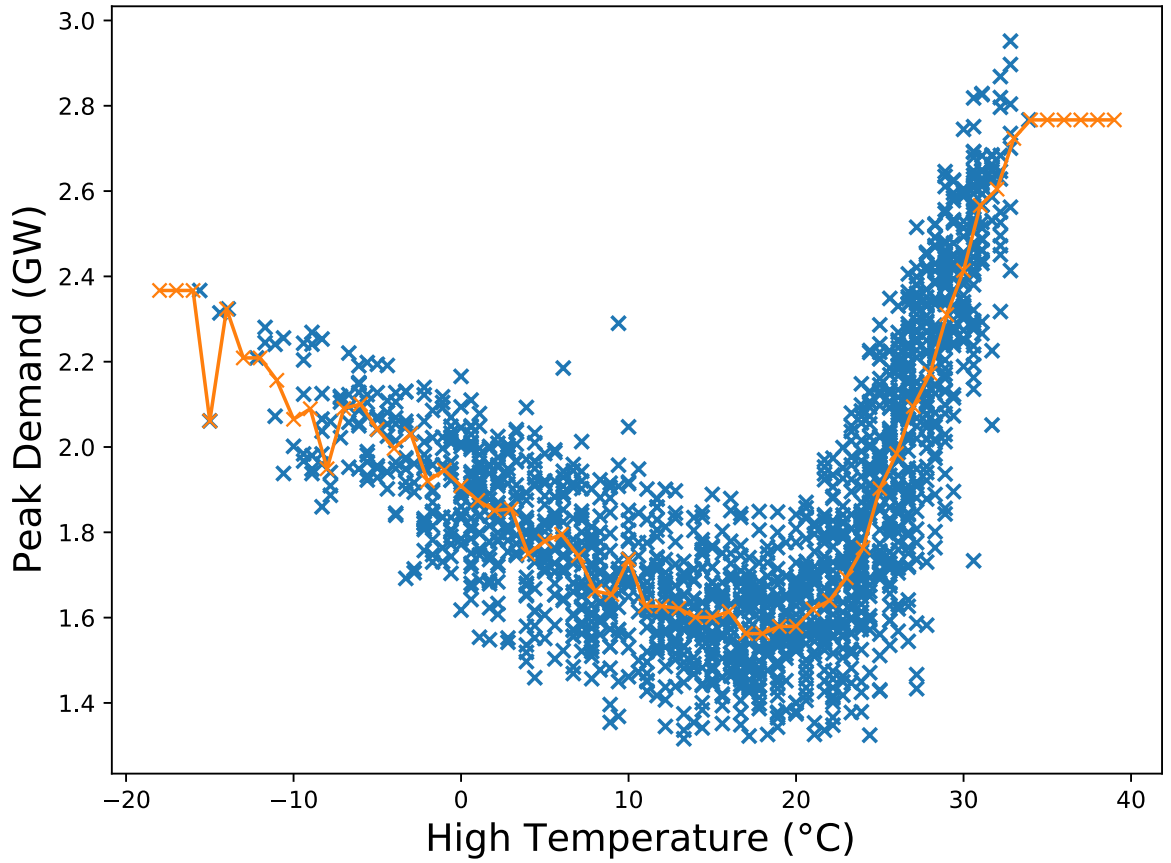


By increasing tree depth we can fit a considerably smoother line (1/3)



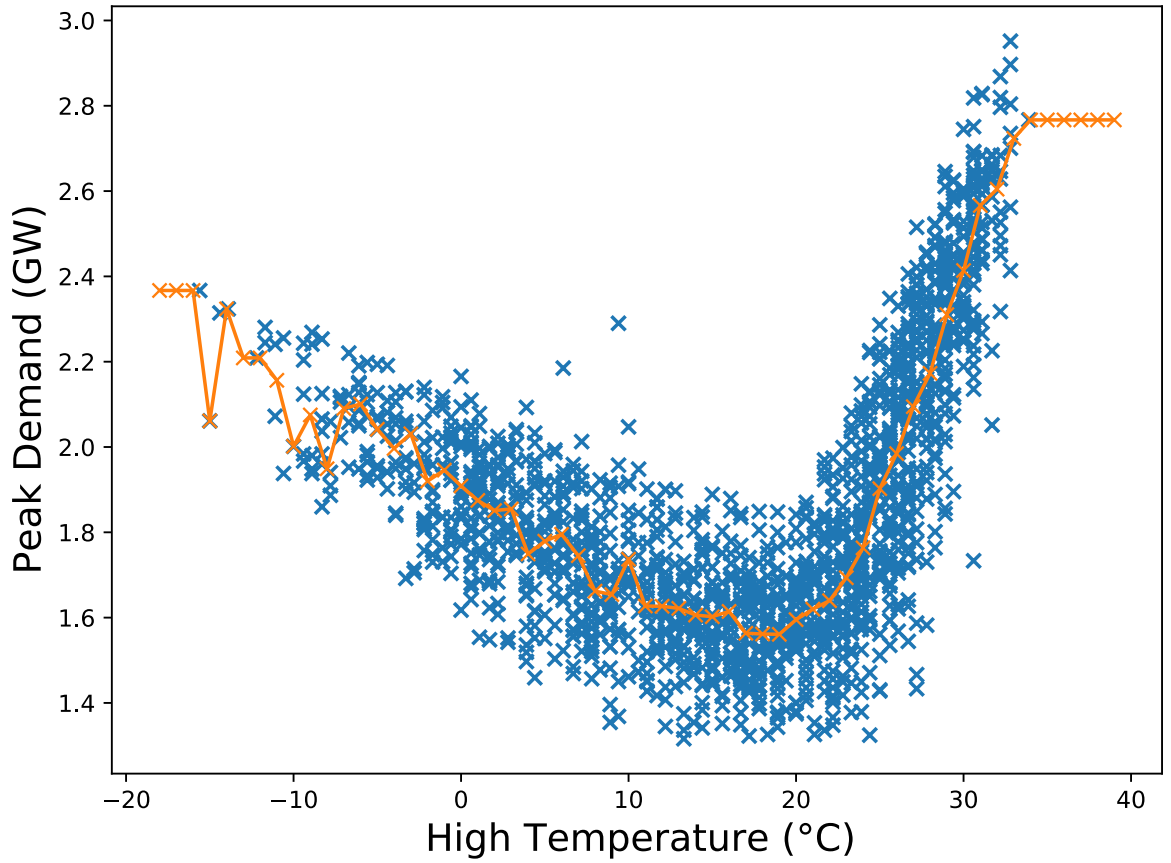
- **Tree depths = 5**
- Num nodes = 53
- Num leaves = 27

By increasing tree depth we can fit a considerably smoother line (2/3)



- **Tree depths = 10**
- Num nodes = 159
- Num leaves = 80

By increasing tree depth we can fit a considerably smoother line (3/3)



- **Tree depths = 100**
- Num nodes = 175
- Num leaves = 88
- Note that due to the non-parametric nature of trees, **highly non-linear functional forms are possible**
- Again, overfitting is likely here

Agenda

- Evaluating Classification Models
- Classification and Regression Trees in Practice
- **Non-linear Classification**
- Ensemble Methods

Nonlinear classification

- Just like linear regression, the nice thing about using nonlinear features for classification is that our algorithms remain exactly the same as before
  - I.e., for an SVM, we just solve (using gradient descent)

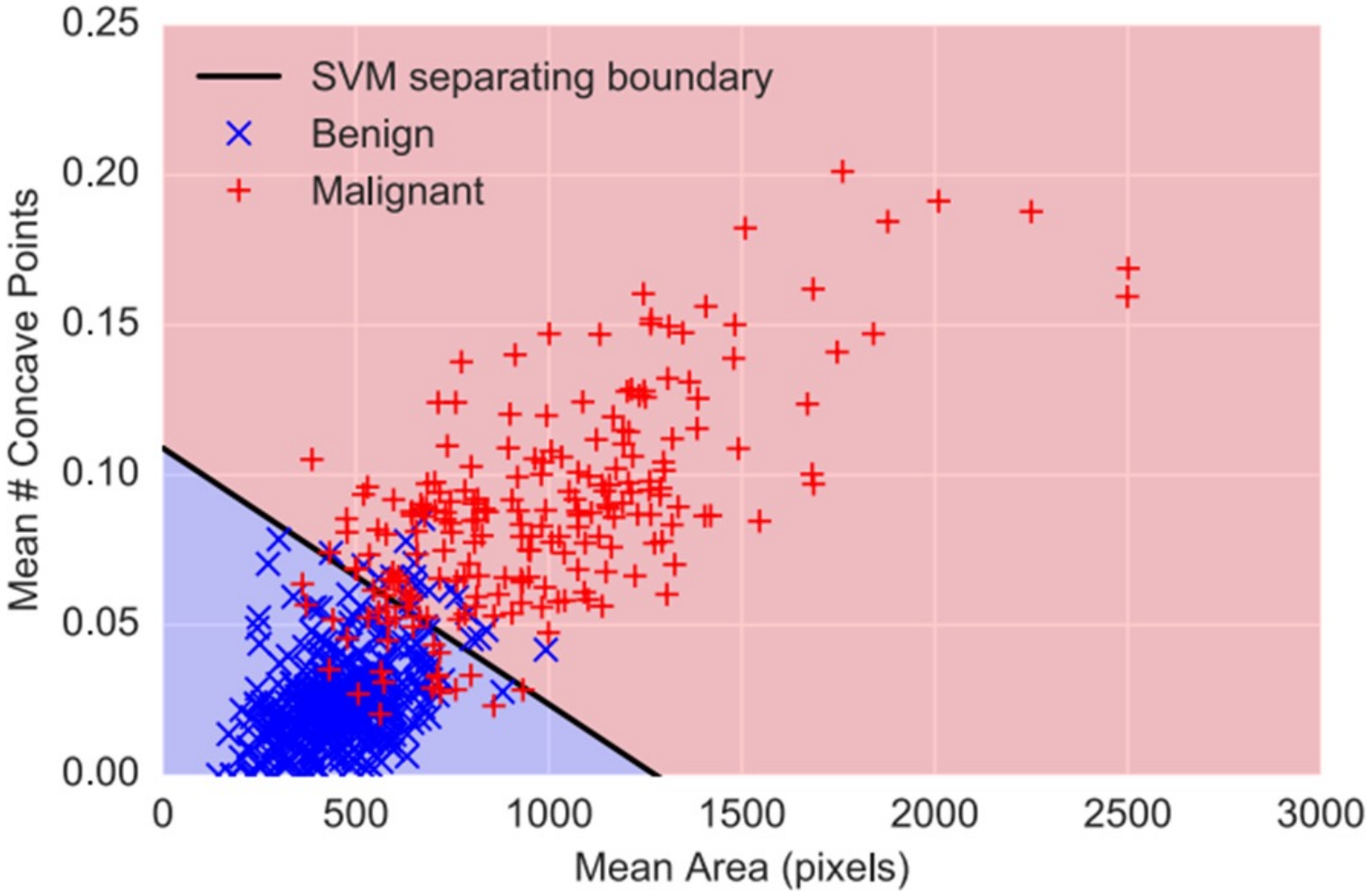
$$\underset{\theta}{Minimize} \sum_{i=1}^{m} \max\{1 - y^{(i)}.\theta^T x^{(i)}, 0\} + \frac{\lambda}{2}\|\theta\|_2^2$$

- Only difference is that $x^{(i)}$ now contains non-linear functions of the input data

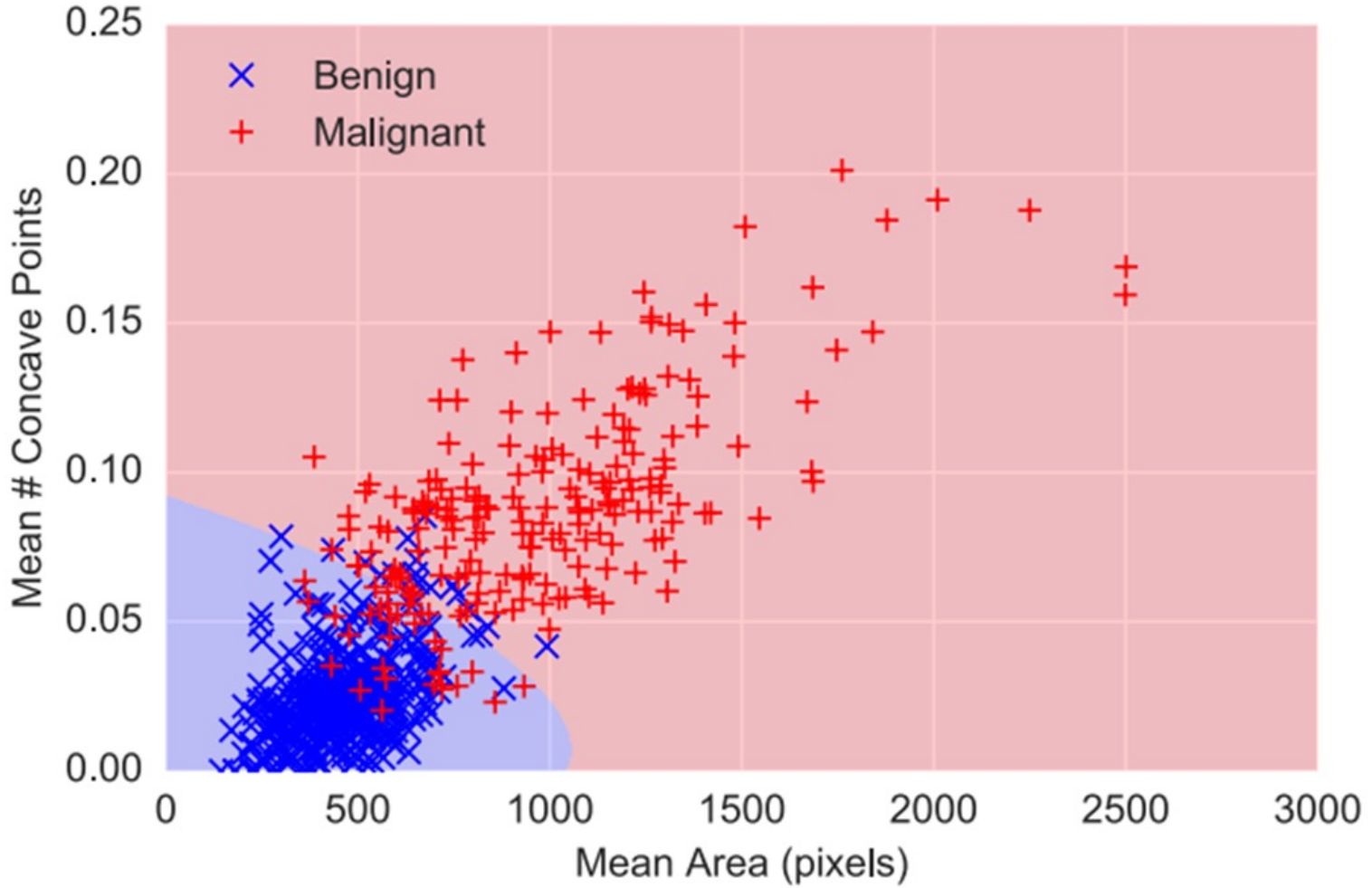
Notation for more general features

- We previously described polynomial features for a **single** raw input, but if our raw input is itself multi-variate, how do we define polynomial features?
- Deviating a bit from past notion, for precision here we're going to use $x^{(i)} \epsilon\ \mathbb{R}^m$ to denote the **raw inputs**, and $\phi^{(i)}\ \epsilon\ \mathbb{R}^m$ to denote the input features we construct (also common to use notation $\phi(x^{(i)})$)
- We'll also drop (i) superscripts, but important to understand is that we are transforming each feature this way.
  - E.g., for a 2-degree polynomial: $x = [Mean\_Area]\ , \phi = \begin{bmatrix} x^2 \\ x \\ 1 \end{bmatrix}$

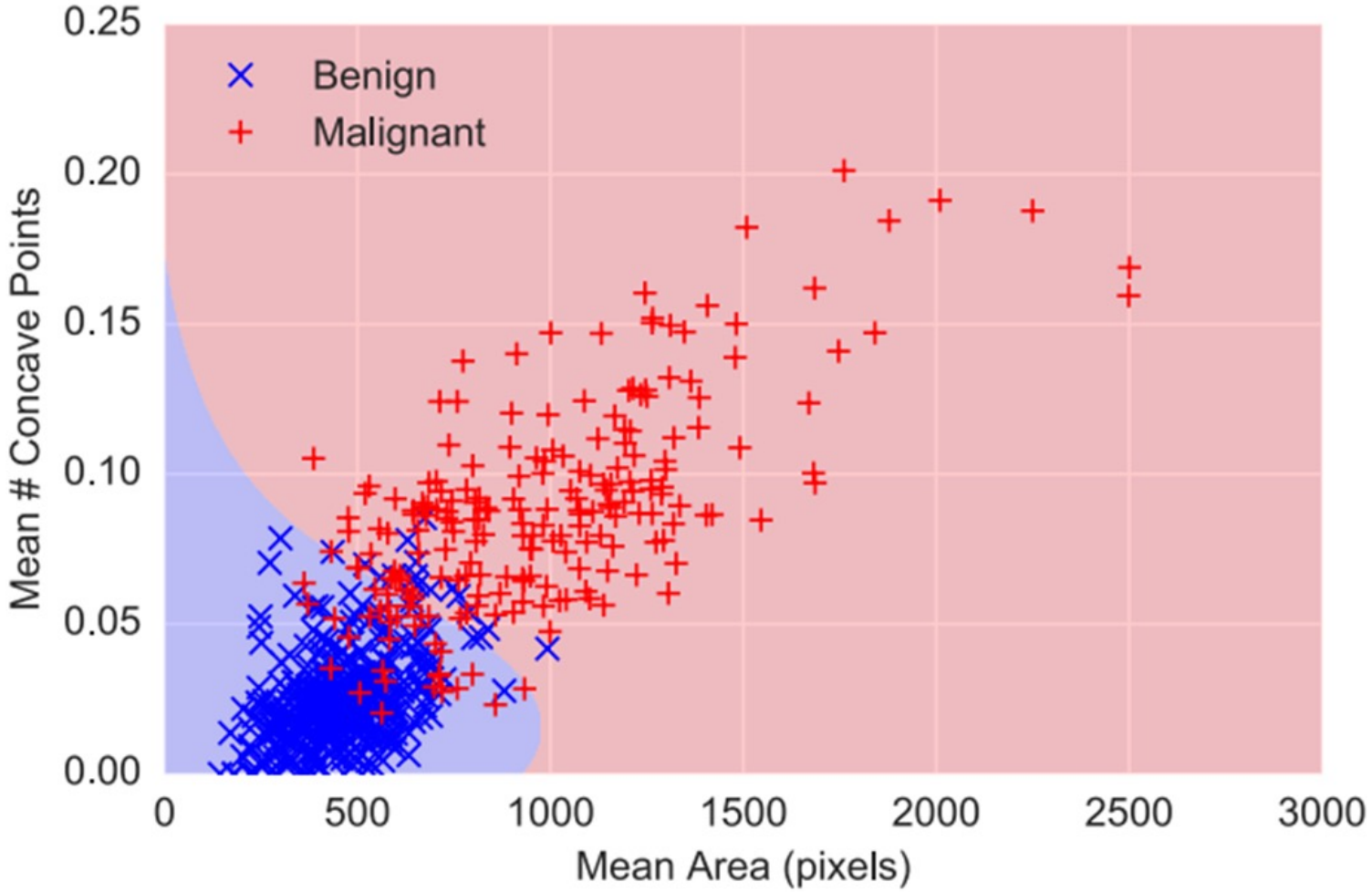
Linear SVM on cancer data set



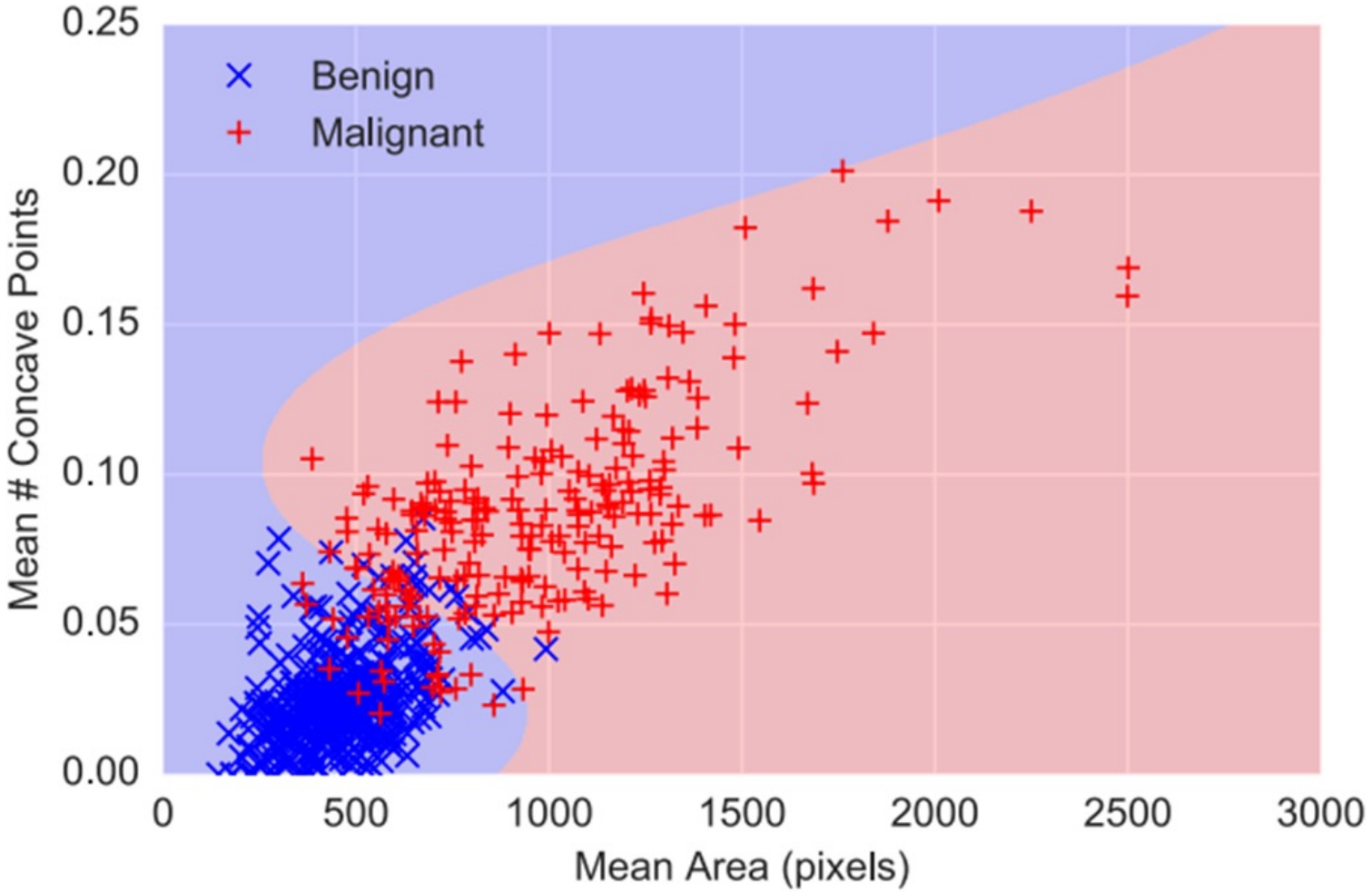
Polynomial features d = 2



Polynomial features d = 3



Polynomial features d = 10



Radial basis functions are another very common way of non-linear classification

Radial Basis Function (RBF)

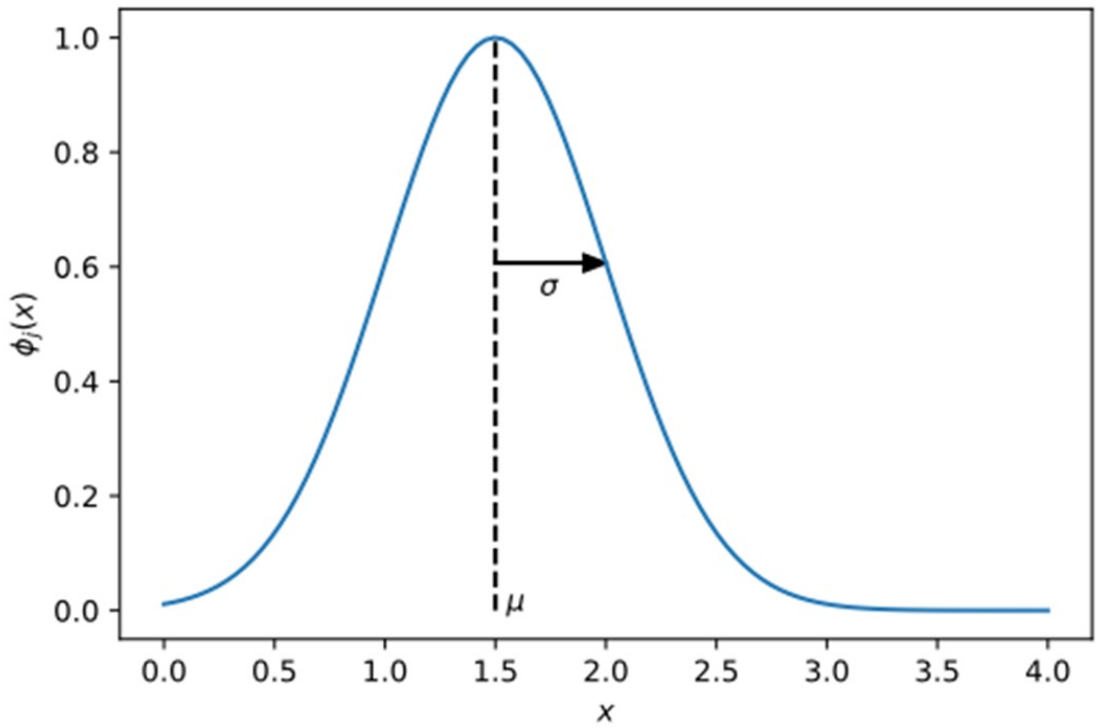
$$\phi : \mathbb{R} \to \mathbb{R}^k = \begin{bmatrix} \exp\left(\frac{-(x-\mu^{(1)})^2}{2\sigma^2}\right) \\ \exp\left(\frac{-(x-\mu^{(2)})^2}{2\sigma^2}\right) \\ \vdots \\ \exp\left(\frac{-(x-\mu_{(k-1)})^2}{2\sigma^2}\right) \\ 1 \end{bmatrix}$$

where $\mu_{(1)},\ldots,\mu_{(k-1)} \in \mathbb{R}$ (called the means) and $\sigma \in \mathbb{R}$ (called the bandwidth) are the hyperparameters of this feature vector

- Arguably the most frequently used type of non-linear feature is not the polynomial, but the *radial basis function* , often abreviated as RBF
- RBFs are non-linear functions in that they are *local* features: the value of any particular feature is close to zero for most of the input space, but non-zero in a small regions around a “center” parametr (μ)

Let us consider a single $\phi_j(x)$ component of the RBF feature vector in more detail

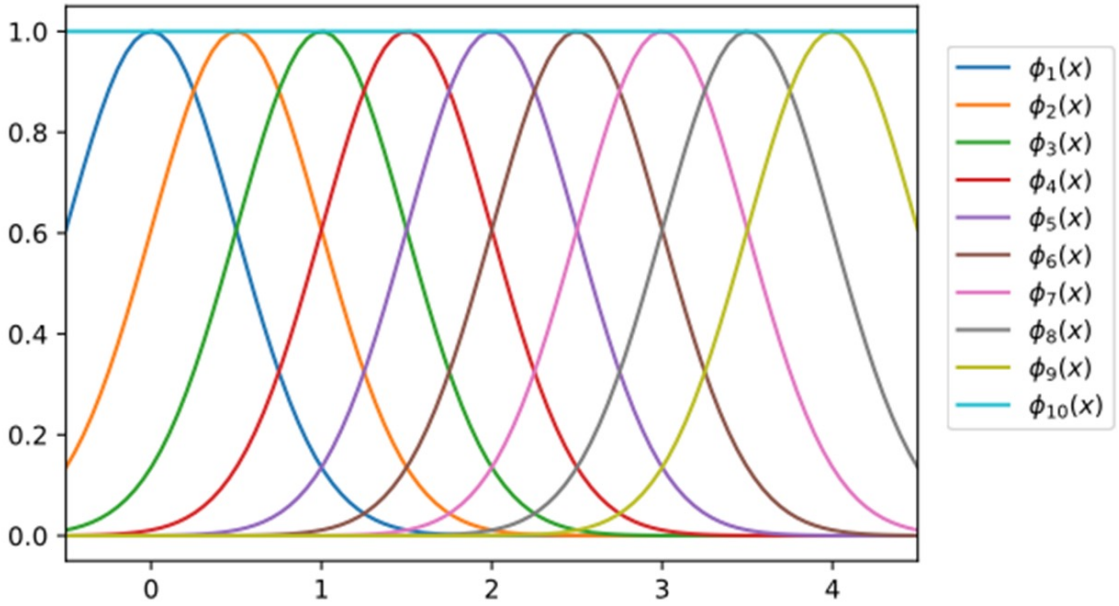
RBFs



- One single dimension of this feature (for varying inputs x, and here assuming mean $\mu_{(j)}$=1.5 and σ=0.5) looks like the graph shown on the left
- You may recognize this $\phi_j(x)$ as looking similar to the density function of the Gaussian distribution (although without the normalizing constant that sets it to mean zero)
- The feauture is largest (equal to one) when x is equal to μ, but falls off very rapidly as x moves away from μ

How do we now make predictions – To get an intuition, consider a feature vector with k=10 RBF centers (i.e. 9 RBF features plus a constant term)

10 RBF centers

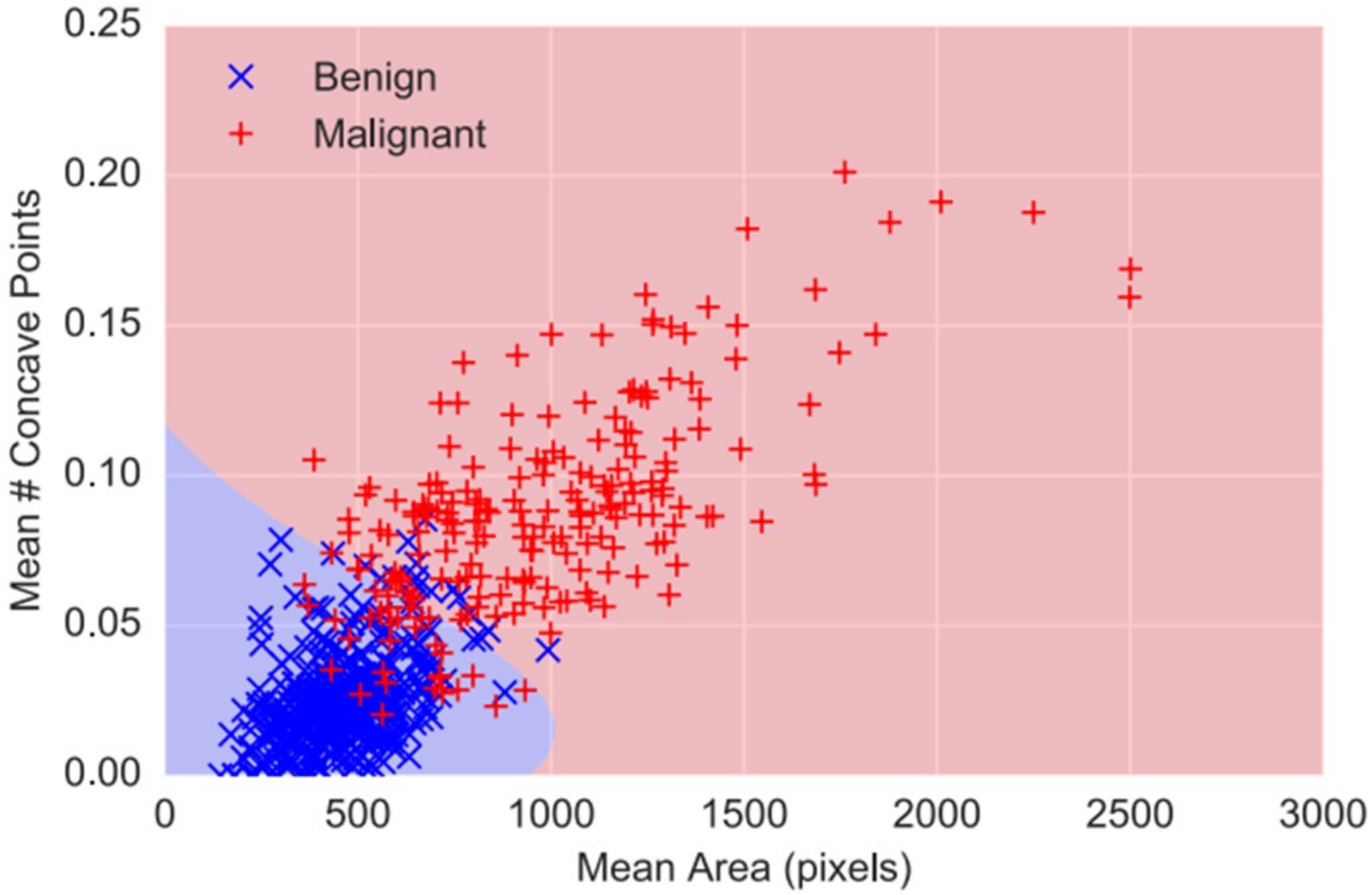


- The goal of **nonlinear fitting with RBFs** is to approximate the underlying function with a **linear combination of the RBF features,** as we did previously with polynomial features
- By combining them in the proper manner, it is **possible to approximate very general functions**

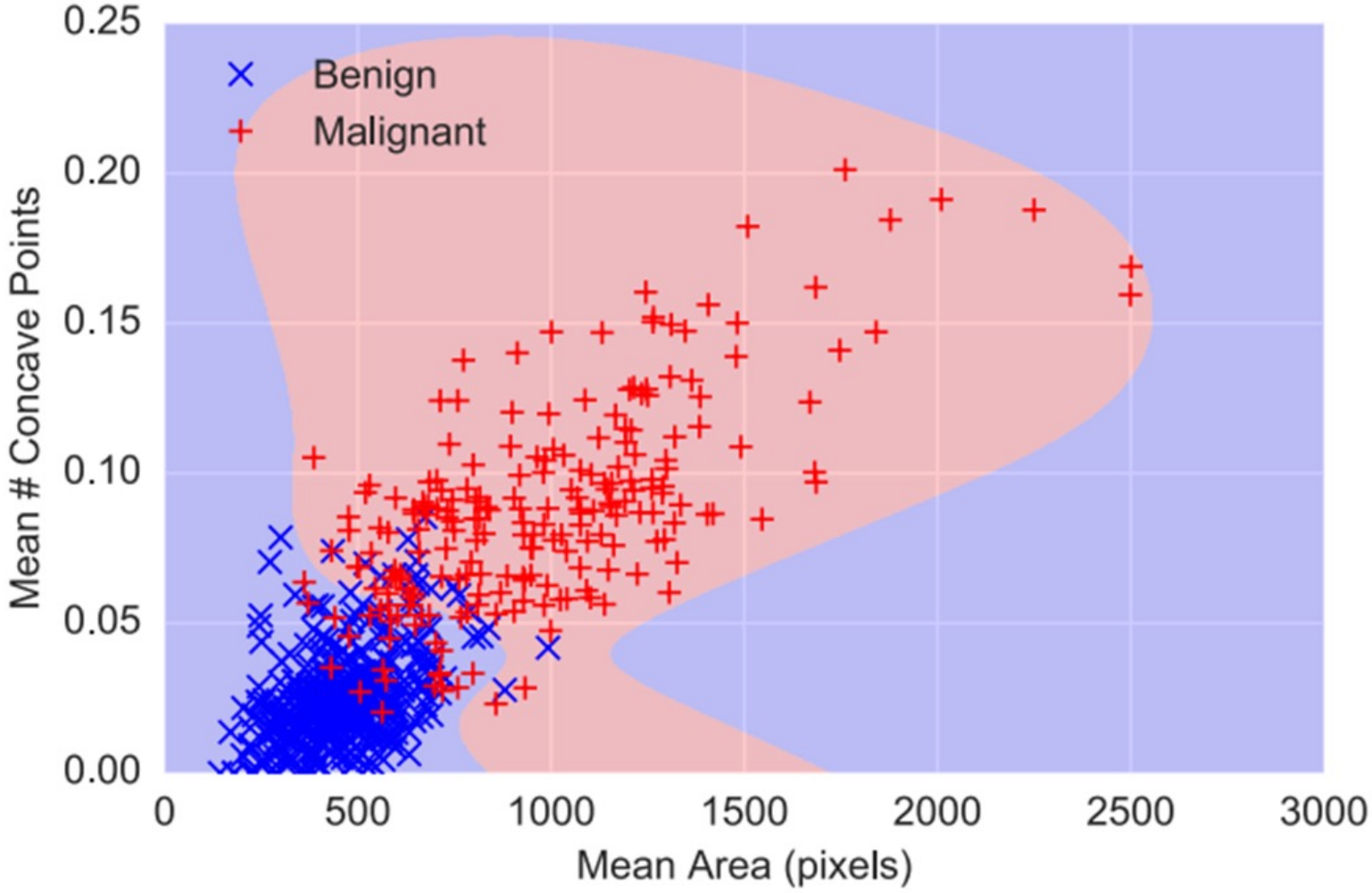
RBF features

- In the following, we assume that X has been normalized so that each feature lies between $[-1, +1]$ (same as we did for polynomial features)
- We observe how the classifier changes as we change different parameters of the RBFs
- p will refer to total number of centers, *k* will refer to the number of centers along each dimensions, assuming centers form a regular grid (so since we have two raw inputs, $p = k^2$)

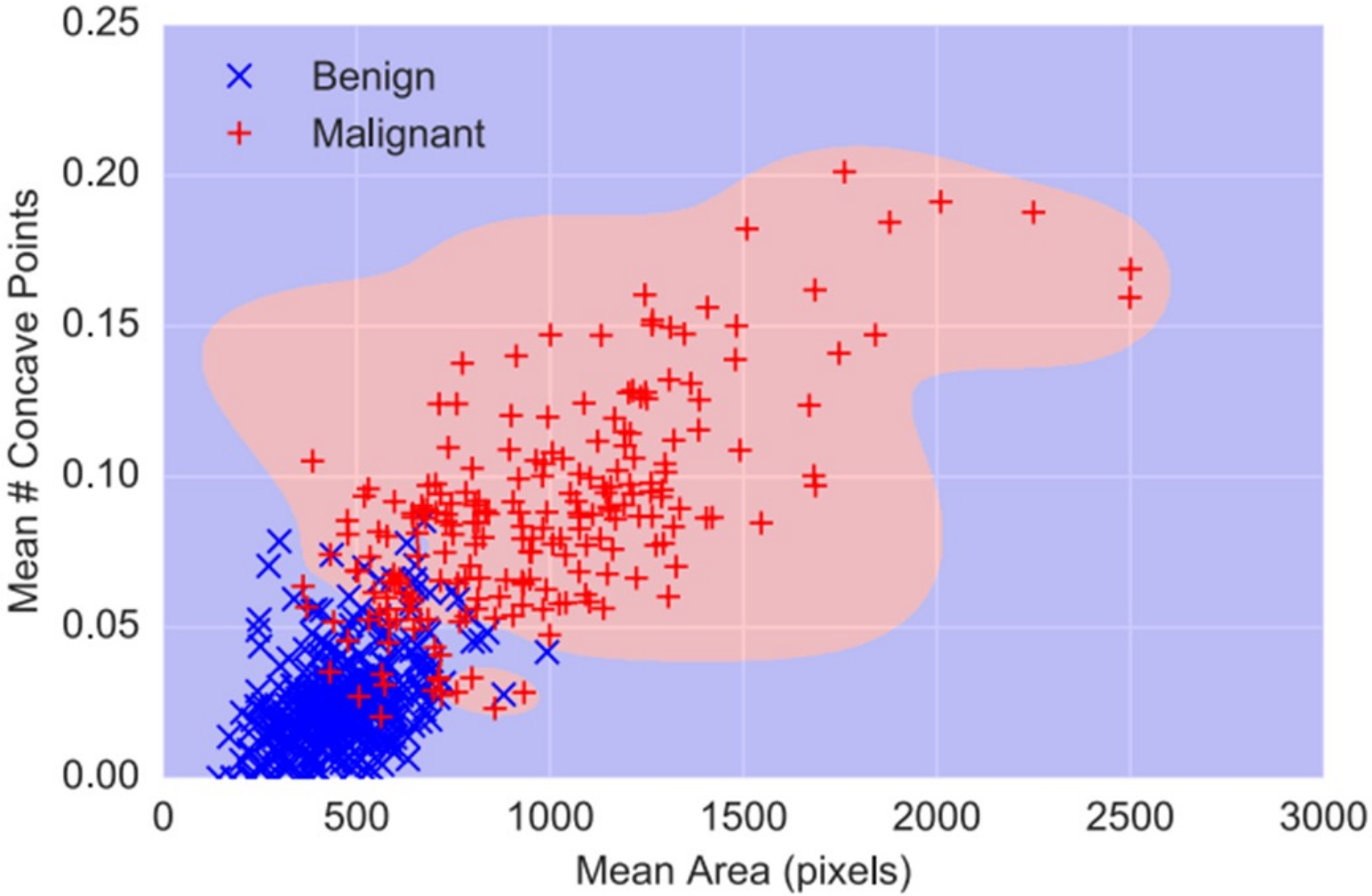
RBF features, k = 3, σ = 2/d



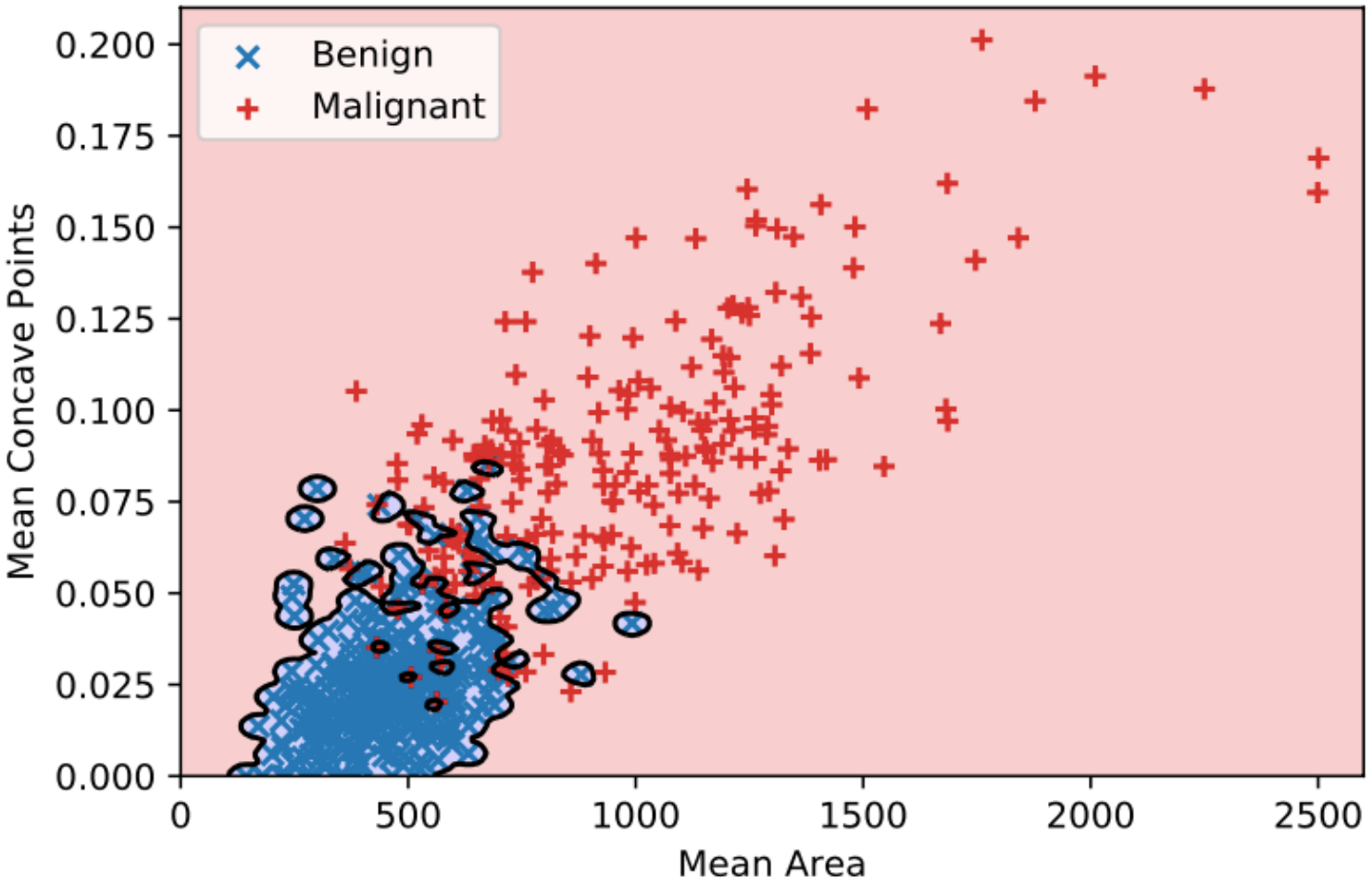
RBF features, k = 10, σ = 2/d



RBF features, k = 20, σ = 2/d



Overfitting taken to the extreme

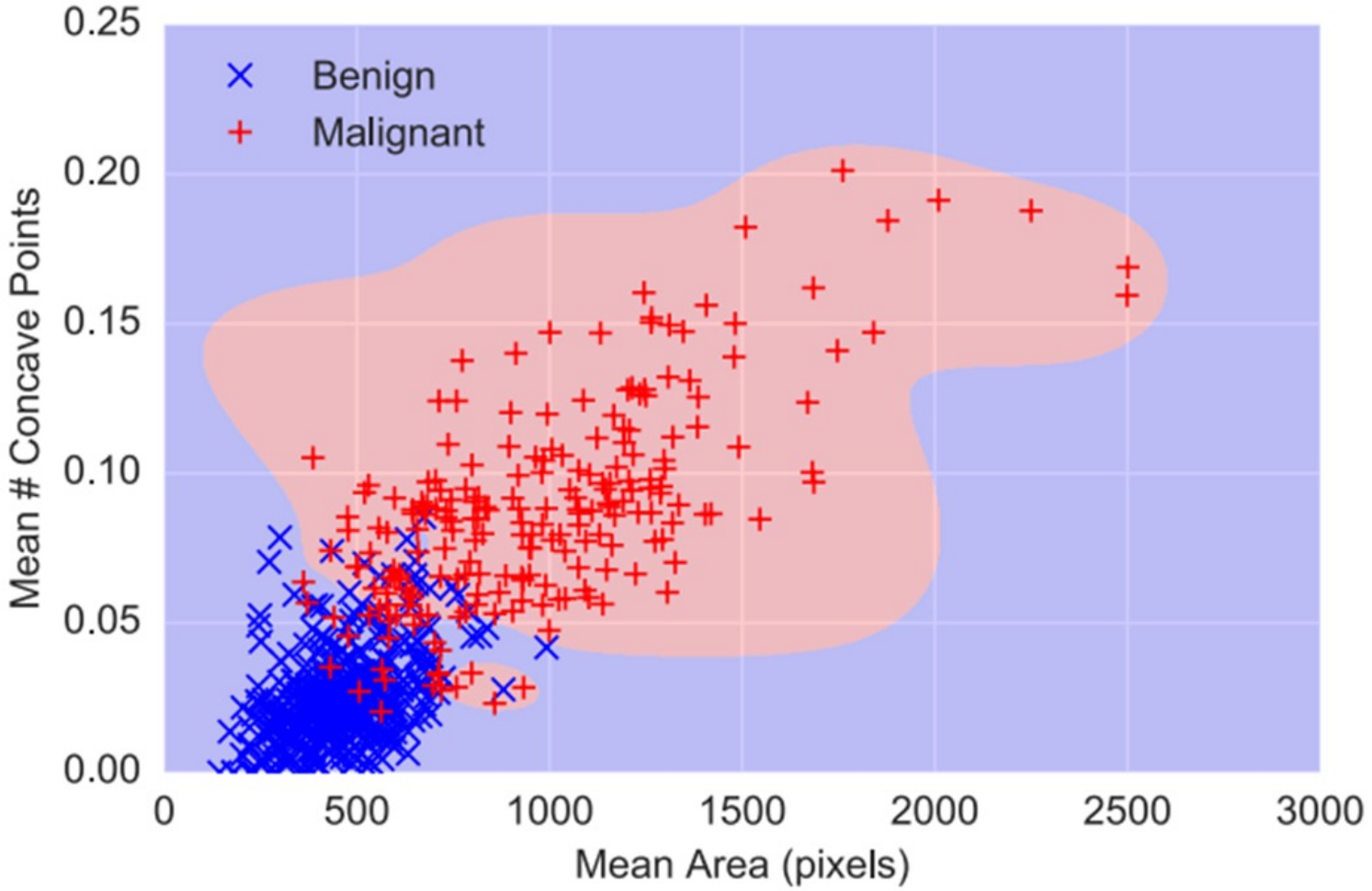


- Because the preceding examples are all quite similar (the decision boundary is roughly linear) let's introduce overfitting for illustrative purposes
- We make the bandwidth very small, and the regularization small, so that the classifier actually manages to get 100% accuracy on the training data

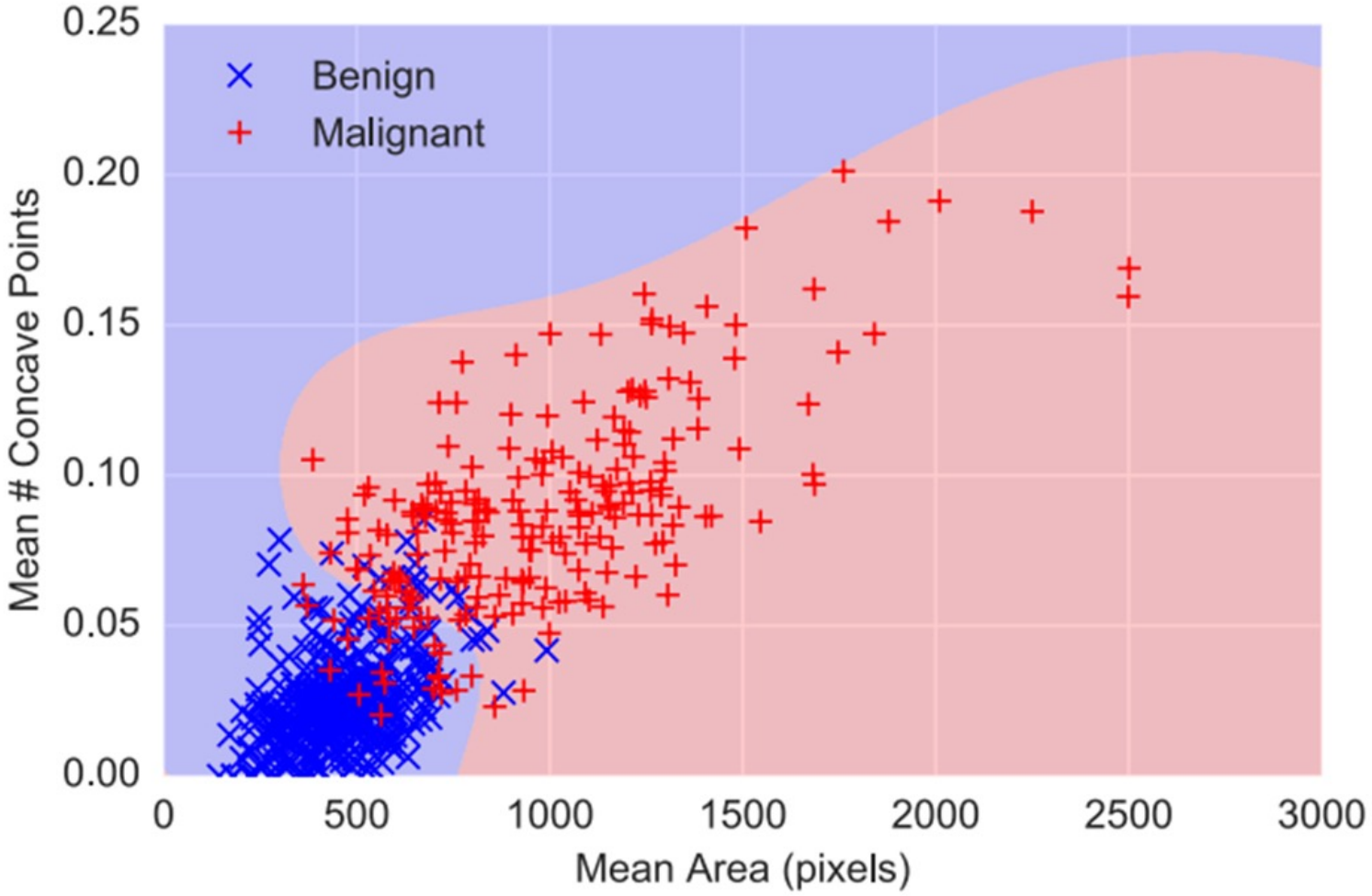
Model complexity and bandwidth

- As seen previously in a regression setting, we can control model complexity with RBFs in three ways: two of which we have already seen
  1. Choose number of RBF centers
  2. Increase/decrease regularization parameter
  3. Increase/decrease bandwidth

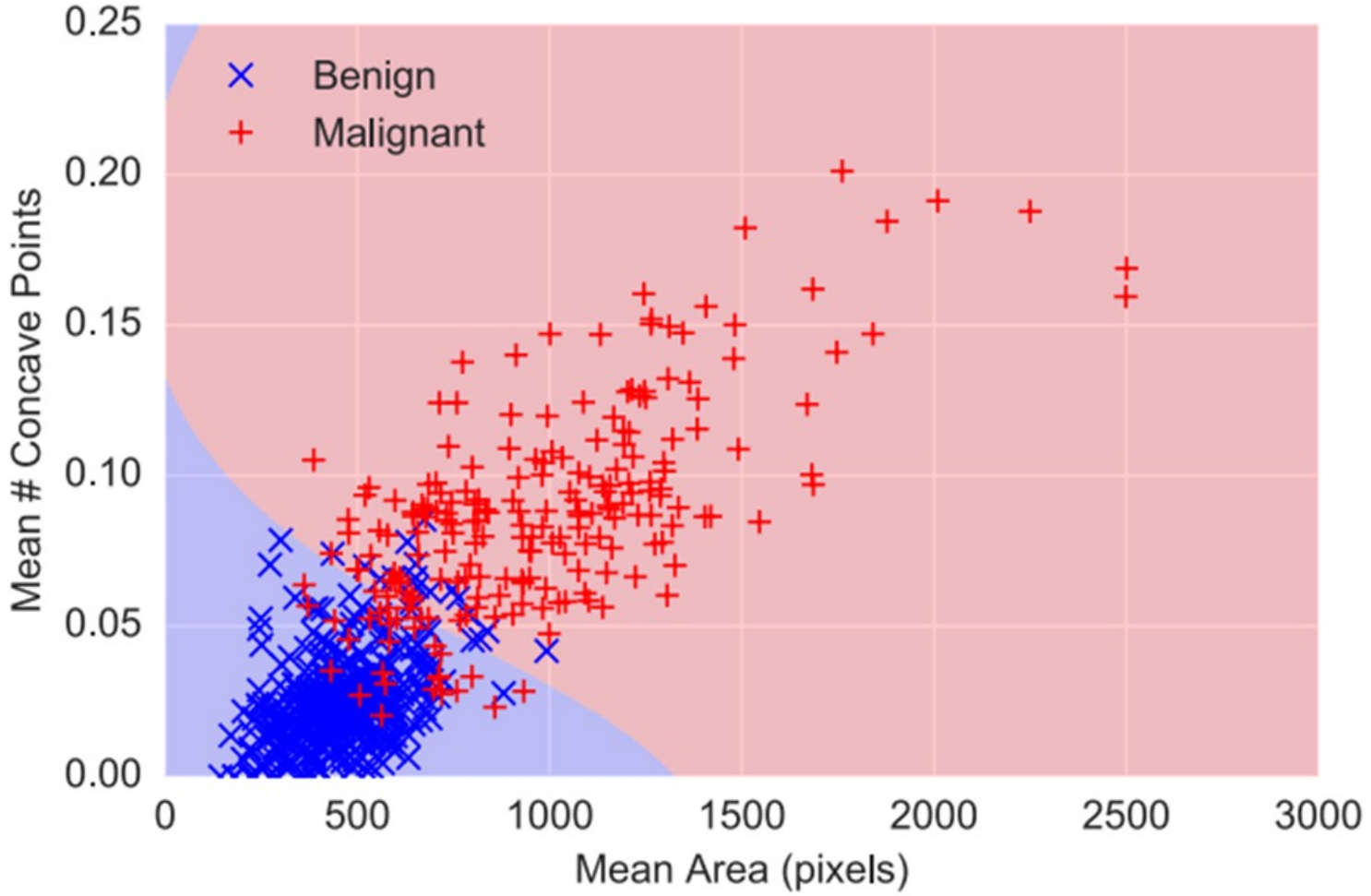
RBF features, k = 20, σ = 0.1



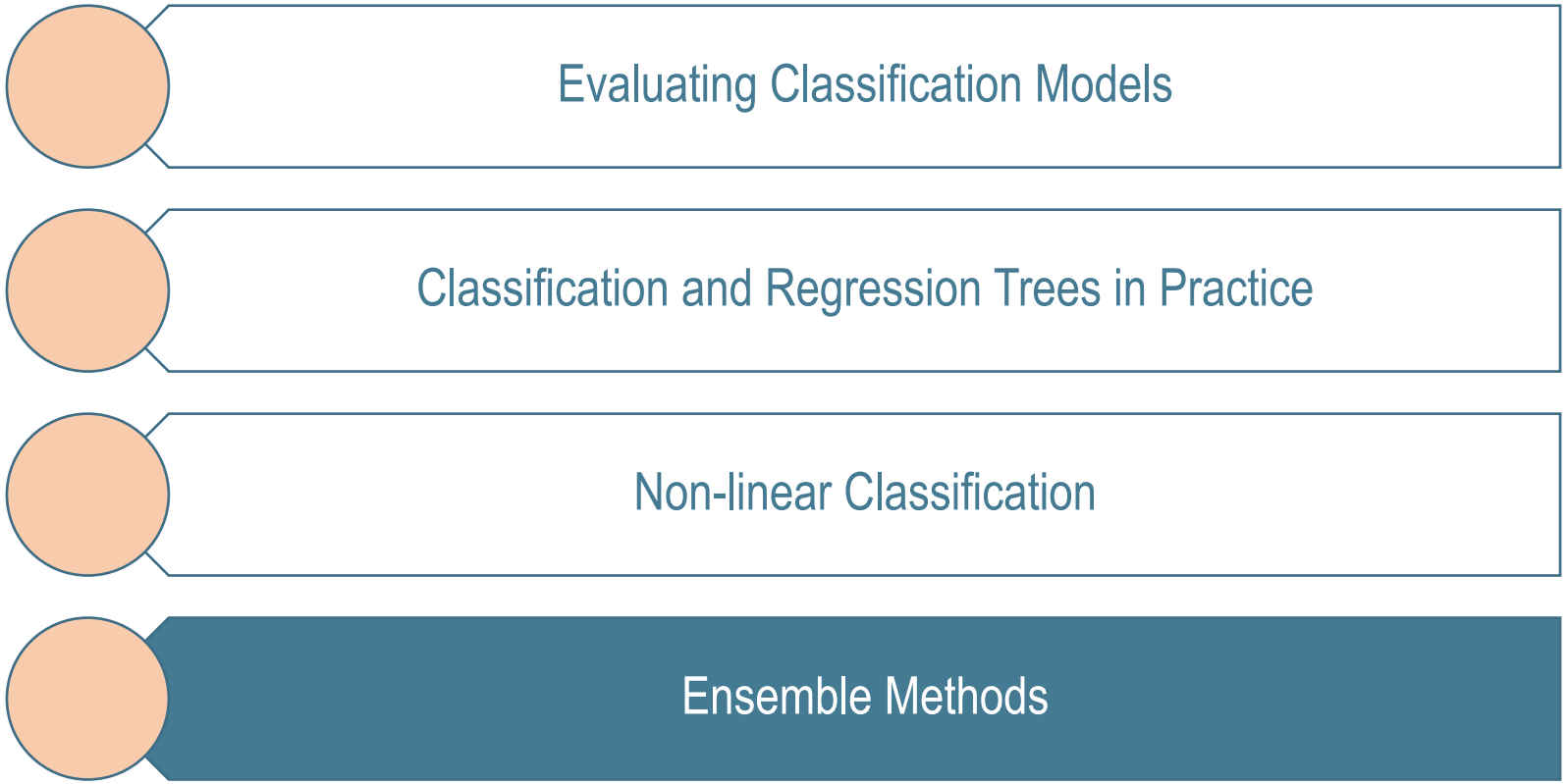
RBF features, k = 20, σ = 0.5



RBF features, k = 20, σ = 1.07



Agenda


Evaluating Classification Models
Classification and Regression Trees in Practice
Non-linear Classification
Ensemble Methods

Ensemble learning – Harnessing the Wisdom of the Crowds

What is the weight of the ox?

- **Francis Galton**, a prominent statistician from the 19th century, **watched a contest at a county fair in England**
- The contest's objective was to **guess the weight of an ox**
- Individual contest **entries were highly variable**, but **the mean of all the estimates was surprisingly accurate – within 1% of the true weight** of the ox.
- On balance, the **errors** from multiple guesses tended to **cancel one another out**
- **Ensemble learning works in much the same way!**

Source: Shmueli et al.

Ensemble learning – Basic intuition

- In predictive modeling, “risk” is equivalent to variation (i.e. variance) in prediction error
- By **combining** individual models **variance may be reduced** – To see this let‘s consider an example:
  - Let‘s consider two models with zero avg. error: $E(e_{1,i}) = E(e_{2,i}) = 0$:
  - Combining both by taking the average we will still get zero avg. error:
    - $E(\frac{e_{1,i}+e_{2,i}}{2}) = 0$
  - However, the variance of the ensemble may be lower than each individual variance, if the covariance (i.e., correlation) is small or negative:
    - $Var(\frac{e_{1,i}+e_{2,i}}{2}) = ¼\ (var(e_{1,i})+var(e_{2,i})) + ¼\ 2cov(e_{1,i}, e_{2,i})$
- Thus, using an **average of two predictions can** potentially **lead to smaller error variance**, and therefore better predictive power.
- These results generalize to more than two methods; you **can combine results from multiple prediction methods** or classifiers.

Which error correlation in ensemble models is best for reducing variance of the combined learner?

a) Combine models with positively correlated errors
b) Combine models with uncorrelated errors
c) Combine models with negatively correlated errors
d) I am not sure

Which error correlation in ensemble models is best for reducing variance?

a) Combine models with positively correlated errors
b) Combine models with uncorrelated errors
c) Combine models with negatively correlated errors
d) I am not sure

The respective errors per model should ideally be negatively correlated to achieve the highest reduction in overall variance

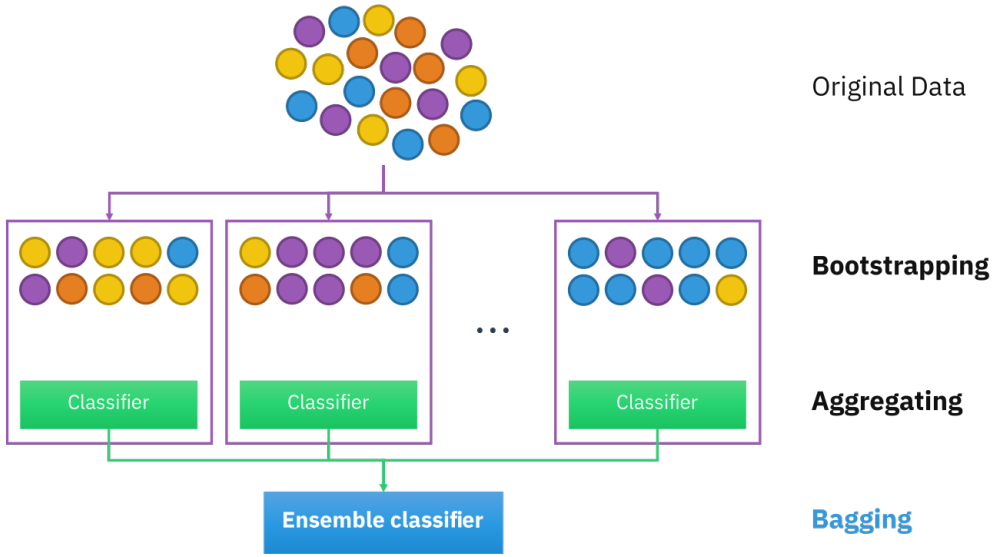
Basic Ensemble Methods for classification and regression

We will discuss three forms of ensemble learning

- **Bagging** – Voting between N learners trained on bootstrapped data of the same dataset and evaluated on a validation set
- **Boosting** – Applying N learners in sequence while applying different weights for wrongly predicted data points in the subsequent step (e.g. XGBoost)
- **Random Forests** – Building multiple trees of different architecture (i.e., a forest) and making predictions bases on majority vote (classification) or mean computation (regression)

Bagging – Intuition

Illustration of Bagging (classification setting)

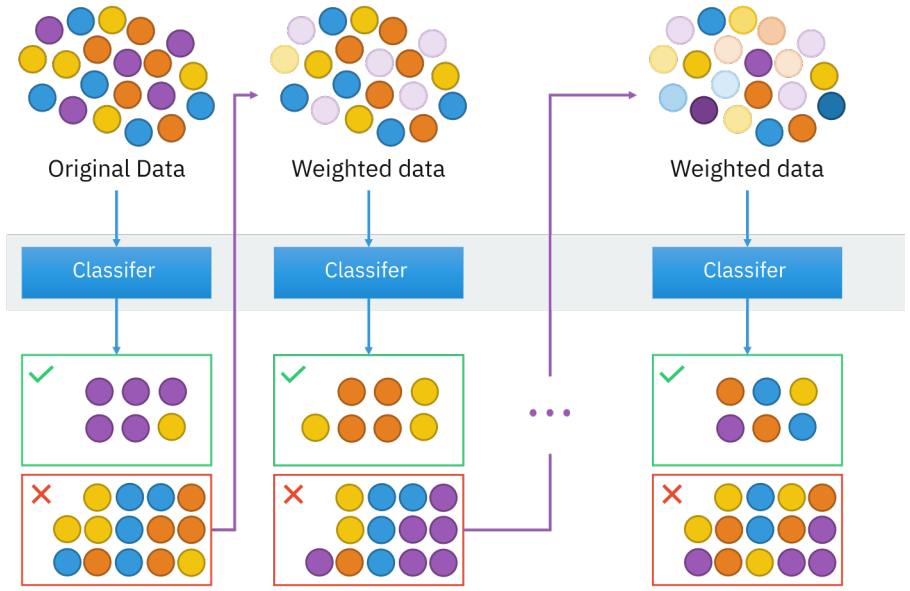


- **Bagging** (or bootstrap aggregation) creates multiple data sets from the original training data by bootstrapping
- Bootstrapping is **re-sampling with repetition** (i.e. entries can be chosen multiple times per bootstrap sample and across samples)
- **The algorithm** runs several models and aggregates output with a voting system

Source: wikipedia

Boosting – Boosting is related to bootstrapping but applies learners in sequence without bootstrapping

Illustration of Boosting (classification setting)

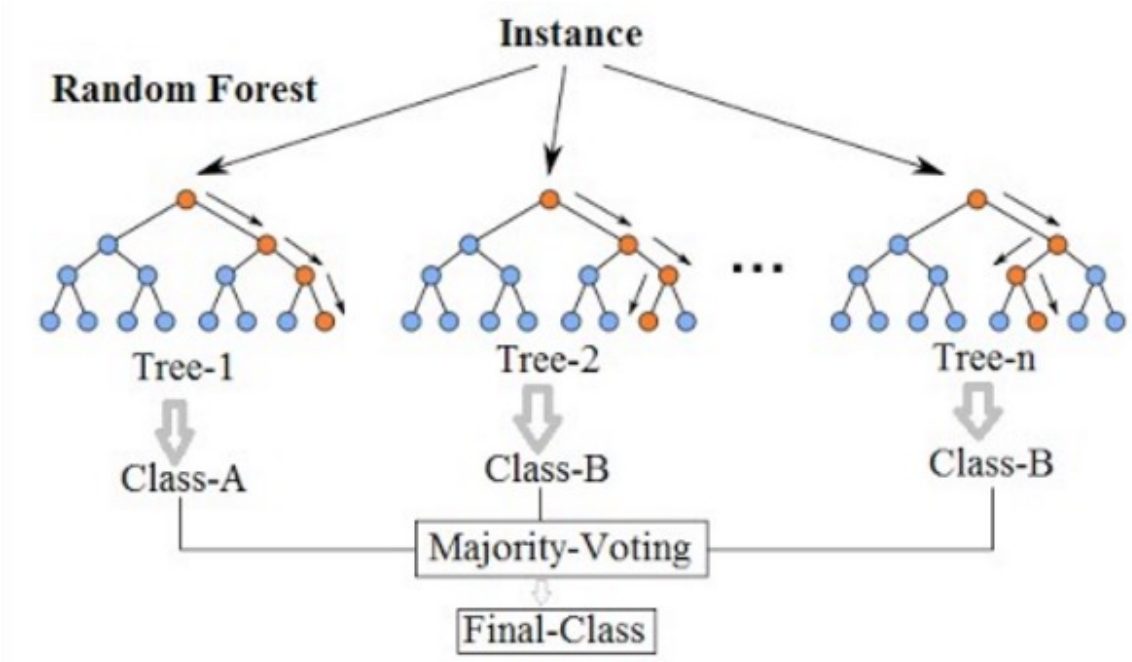


- Boosting **applies learners sequentially**
- **Higher weights** are **assigned** to **observations** that have been **misclassified by** the **previous learner**
- Thus, boosting **combines multiple weaker learners** sequentially into a single stronger learner
- Boosting is **often combined with decision trees** as learners of choice, such as the popular **XGBoost framework**

Source: wikipedia

Random Forests are a very popular type of tree-based ensemble – Training N trees in parallel and predicting via majority vote between tree outputs

Illustration of Random Forests (classification setting)



- Random forests are an ensemble method that combines multiple decision trees for classification or regression, where the term “forest” refers to multiple trees, i.e., multiple learners (typically N = 100 to 1000)
- Predictions are made via a majority vote (classification) or a mean computation (regression)
- Strengths: RFs tackle the bias problem with single decision tree and correct for the tendency to overfit
- Drawbacks: RFs are no longer easy to interpret and do not perform well on small data sets
- Random forests have become a highly popular method in forecasting competitions (e.g., knowledge Discovery in Databases (KDD), Kaggle)

Source: Medium

Contact

For general questions and enquiries on **research**, **teaching**, **job openings** and new **projects** refer to our website at www.is3.uni-koeln.de

For specific enquiries regarding this course contact us by sending an email to the **IS3 teaching** address at is3-teaching@wiso.uni-koeln.de