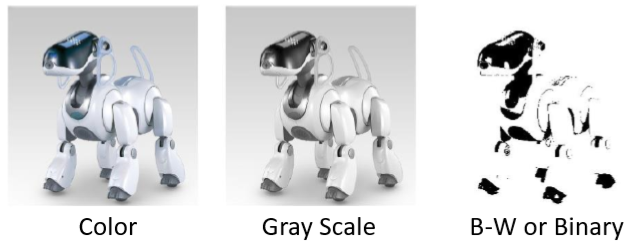
CS 3630: Intro to Perception and Robotics – Notes

# Lecture 1: …

# Lecture 2: Image Processing

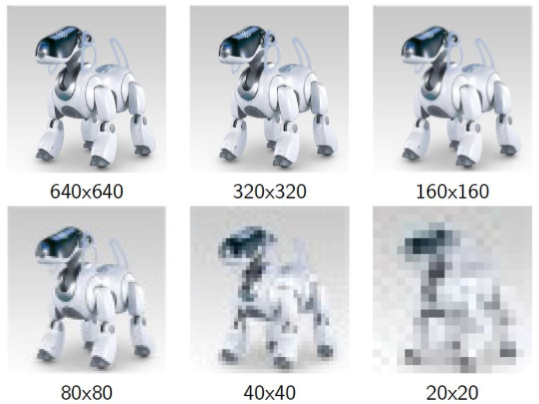
* Cameras are the primary sensor for many robotics platforms
* Common camera types:
  + Single view, RGB image (2D data)
  + Stereo, RGB-D image (3D data)
  + Structured light or time of flight, RGB-D image (3D data)
* Images: A 2D array of intensity/color values
  + You can have color, grayscale, or B-W (binary) images:

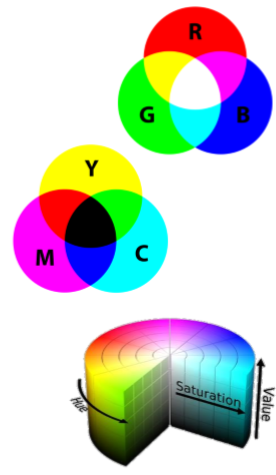


* **Grayscale Images:** A grid (matrix) of intensity values between 0 (black) and 255 (white)
* **Image Processing in the Context of Image Classification:**
  + Vision in robotics is primarily to recognize entities in the images: people, cars, signs, etc.
  + For now we discuss image processing in the context of image classification: does an image belong to a group among a set of categories
* Using Scikit-image and scikit-learn, we fill focus on:
  + **Image adjustment:**
    - <http://scikit-image.org/docs/stable/user_guide/transforming_image_data.html>
    - <http://scikit-image.org/docs/stable/api/skimage.exposure.html#module-skimage.exposure>
  + **Image Filtering:**
    - <http://scikit-image.org/docs/stable/api/skimage.filters.html?highlight=filter#module-skimage.filters>
  + **Image Features:**
    - <http://scikit-image.org/docs/stable/api/skimage.feature.html?highlight=feature#module-skimage.feature>

**Image Adjustments**

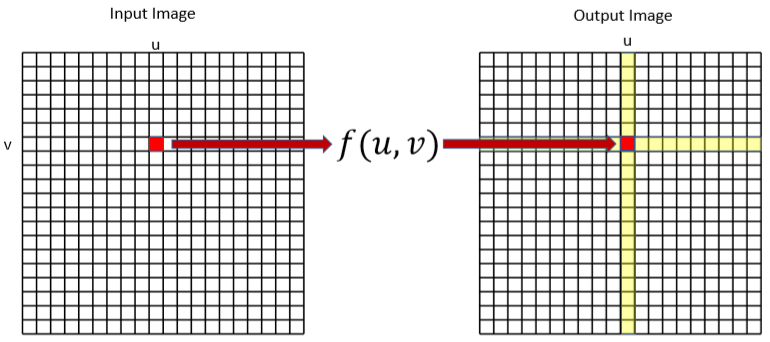
* Some types of image Adjustments:
  + Scaling
  + Color manipulation (color space, color -> grayscale, etc.)
  + Contrast
  + Exposure
* **Image Sampling/Resolution:**



* **Color Models:** An abstract mathematical model describing the way colors can be represented as tuples of numbers (usually in three or four values). Some models:
  + **RGB** – Red, Green, Blue
  + **CMYK** – Cyan, Magenta, Yellow, Black
  + **HSV** – Hue, Saturation, Value
* No color model is “better” than another but, for specific applications, one model might be more suitable than another
  + E.g. HSV might work better when looking for a dark object on a light background
* Adjusting image contrast and/or exposure levels can help normalize images and remove differences that occur due to lighting

**Image Filtering**

* Converts an input image into an output image by performing operations on a pixel-by-pixel basis
* Can be used for:
  + Noise removal
  + Edge detection
  + Multi-scale algorithms
  + Feature detection
  + Matched filters
  + …
* **Monadic Operators for Filtering**
  + Monadic operations take a single pixel as input and give a single pixel output
  + They do not consider neighboring pixel values



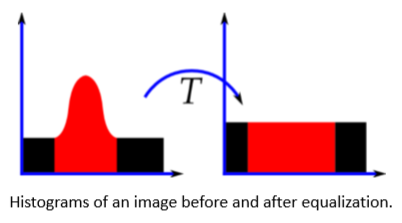
* The images below are examples of monadic image operations:
  + Original Image
  + **Thresholding**: All values greater than some threshold

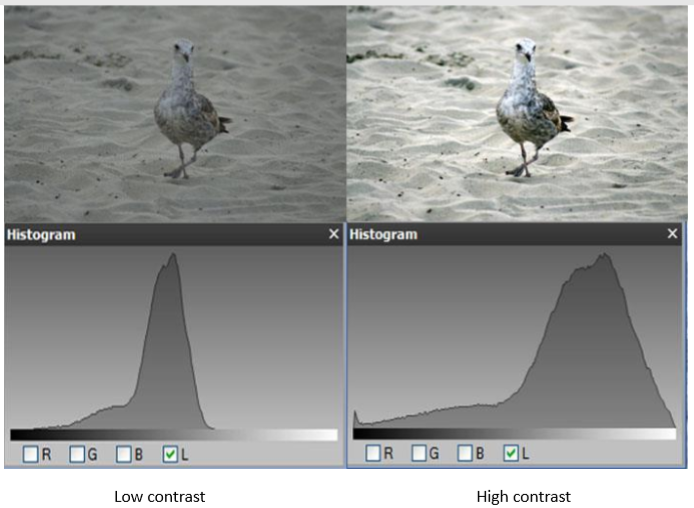
are mapped to white, and the ones smaller are mapped to black

* + **Histogram Normalized:** More detail below
  + **Posterization:** Converts continuous gradients of a tone into defined regions of fewer tones, creating abrupt changes from one tone to another
* **Color Histogram:** A histogram array represents the distribution of colors in image
  + If the image is grayscale, it represents the *intensities*
  + For the color , the value gives the number of pixels of color in image
* An simple algorithm to create a histogram
  + If is the color label of pixel

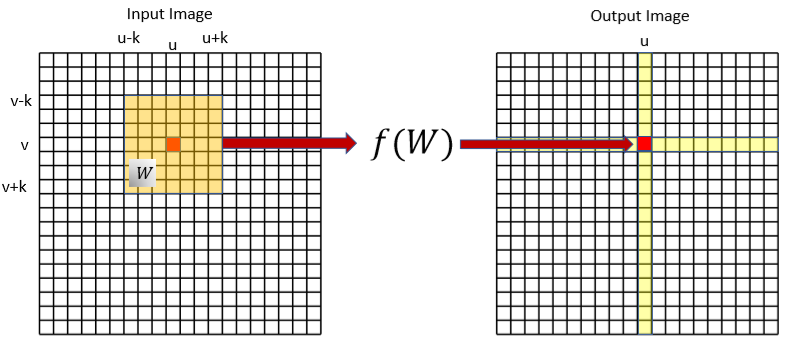
Initialize:

For pixel

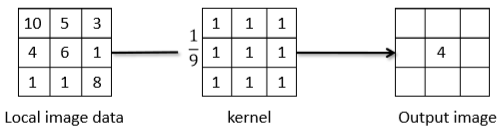
* **Histogram Filter:**
  + You can use a histogram to redistribute the intensity or color distribution of an image
  + This usually **increases the contrast** of an image by spreading out the most frequent intensity values
  + Which is useful for images with backgrounds and foregrounds that are similar (both bright or both dark)
* **Contrast:** The difference between a maximum and minimum pixel intensity in an image (or in a region of an image)
  + ***You can do that with skimage.exposure***



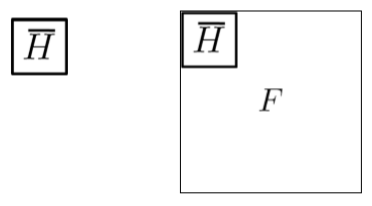
* **Local Operators:** Use region of pixels in the input image to determine the value of a single pixel in the output image



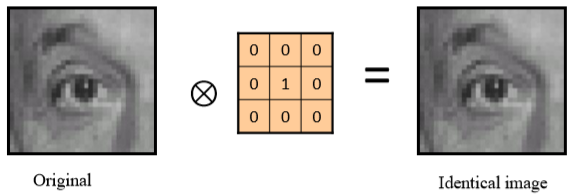
* E.g. *Averaging using a linear filter*
  + Replace each pixel by a linear combination of its neighbors
  + The matrix of the linear combination is called the “kernel”, “mask”, or “filter”

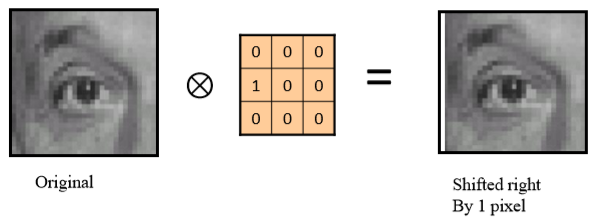


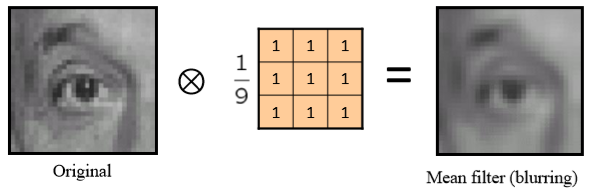
* + I think the example above is summing every element with weight one, then dividing the sum by 9 to get the center pixel
  + If *F* is the image, *H* is the kernel (with size by ), and *G* is the output image, then:
* **Convolution:** Applying a kernel to an image like this (going through all the image using regions

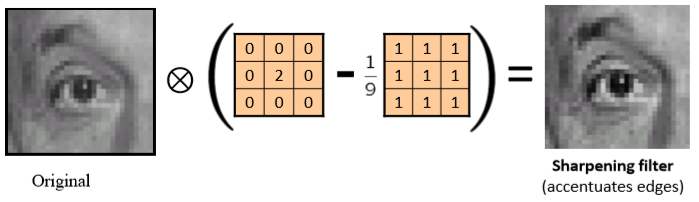


* + Imagine the box sliding around all of
* **Linear filters**: Examples

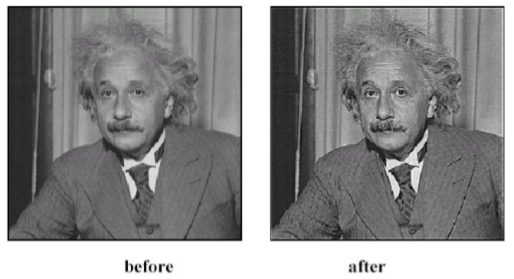




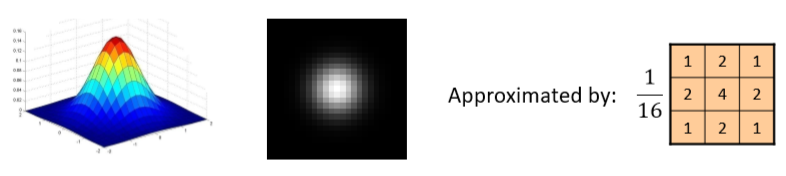


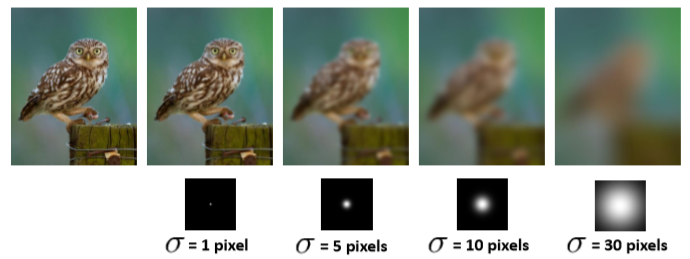


* + Another example of Sharpening:

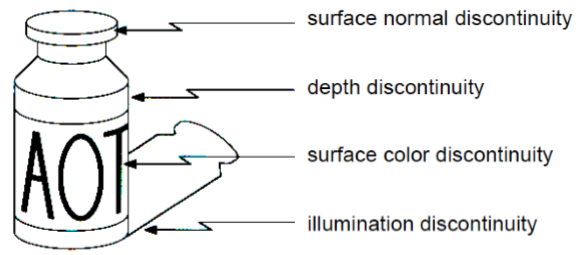


* **Gaussian Kernel**

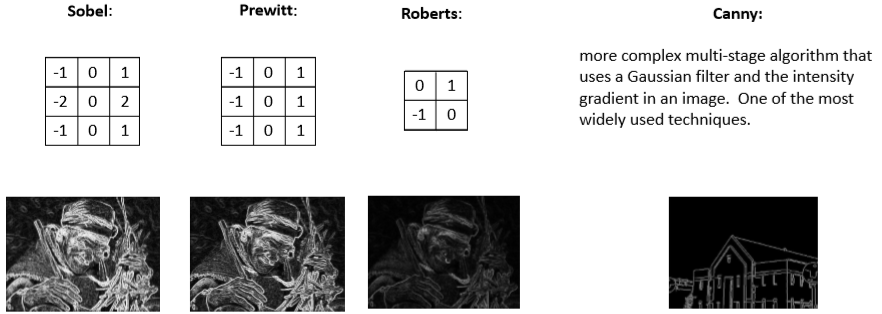




* **Edge Detection:** Origins of edges include



* One way to do edge detection is through Convolution



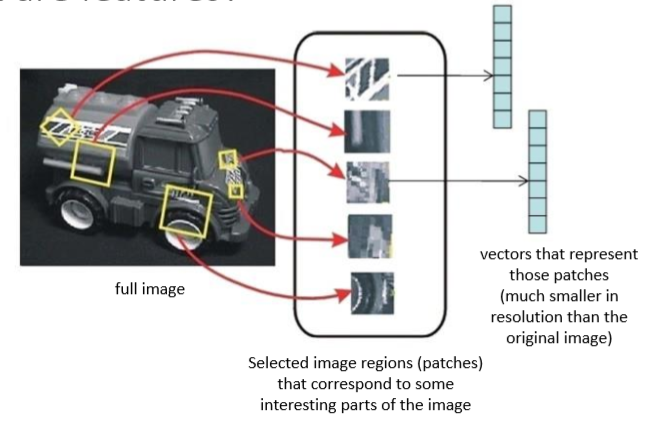
* ***Notes for Lab 1:***
  + **Blurring the image slightly (Gaussian filter) can improve classification performance by smoothing away random noise in the image**
  + **Edge detection can help with some approaches to image classification, but it’s not always necessary**

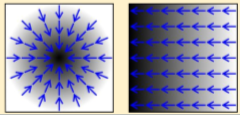
**Image Features**

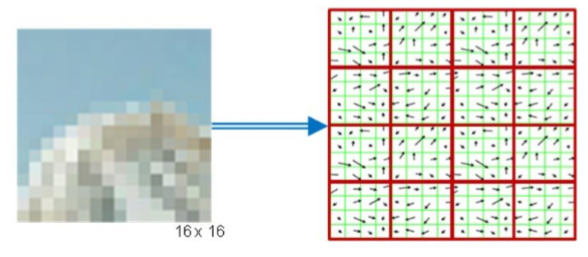
* An image can be thought of as a sequence of variables, where is the number of pixels
* Feature extraction is a type of dimensionality reduction
  + *Reducing* the number of variables under consideration
* The purpose of feature extraction is to represent only the *interesting* aspects of an image
  + Giving you a more concise representation consisting of fewer variables (*features*)
* This is useful when images are large and a reduced feature representation can be used to quickly complete tasks
  + E.g. image matching and retrieval
* **Image Processing Pipeline**

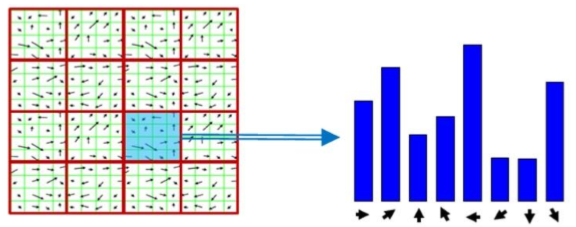


* **Available Feature Extractors**
  + **Histogram of Oriented Gradients (HOG)** feature descriptor: <http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html?highlight=sift>
* **What are Features?**



* **HOG Descriptor:** I think this is the general process
  + Select regions of interest and scale each one to 16x16 pixels
  + Compute the *gradient* for each pixel (orientation and magnitude) and divide the region further into 4x4 pixel squares
    - **Gradient:** A directional change in the intensity or color of an image
  + Concatenate the histograms to obtain a 16\*8 = 128 dimensional feature vector for each region of the image being considered







* **Why HOG features?**
  + For recognition tasks of 2-20 object classes (small-scale recognition tasks), HOG geatures combined with certain machine learning techniques perform very well
  + HOG-based approaches lead performance in many CV benchmarks until recent advances in Deep Learning, they are still the model of choice for small-sclae classification tasks
* **HOG Parameters:**
  + You can tweak many of the variables we discussed: number of gradient orientations (8 in the example above), pixels per cell (16 above), and cells per block (16 above)
  + *The performance of HOG is sensitive to these values, adjust them for good performance*

**Textbook References:**

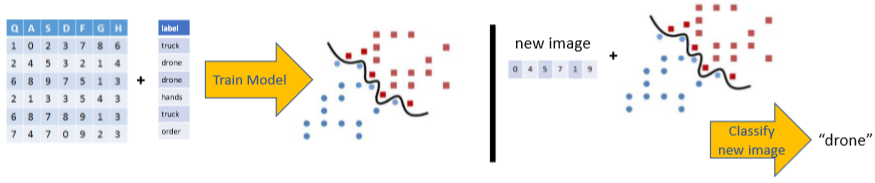
* **Autonomous Mobile Robots: 4.3 – 4.4**
* **Robotics, Vision and Control: 12.2 – 12.4**
* **Scikit-image examples and documentation**

# Lecture 3: Supervised Learning

* We already saw the **Image Processing Pipeline**

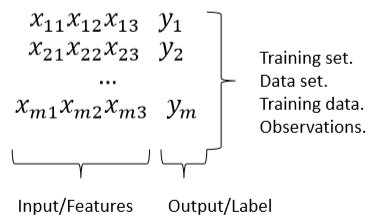


* Now for **Image Classification**
  + In the leftmost table below imagine you just have two columns since you only have two dimensions in the points plotted



## Supervised Learning

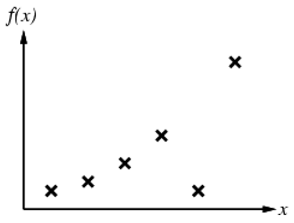
* Given a set of data points with an outcome, supervised learning creates a model to describe them
* Two ways:
  + **Classification:** Outcome is a discrete variable (with <10 outcomes usually)
  + **Regression:** Outcome is continuous
* **Training Data**:
  + In the data you collect, every was generated by some equation
  + Supervised learning tries to discover a function that approximates , we call the hypothesis



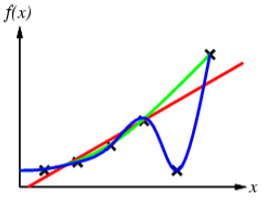
* **What else do we have?**
  + In real life, you usually have background knowledge (theory about the underlying process, etc.) in addition to the set of data
* We will use **just the data** (which makes this **inductive learning**)
  + It’s a better base case and “cleaner”
  + Also, you need more complex mechanisms to reason with prior knowledge

**Inductive Learning**

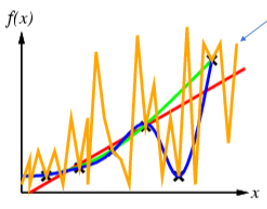
* Given a set of observations, come up with a model *h* that describes them
  + *h* models the observations well, and *is likely to predict future obervations well*
* For example, Given the points below from *f*:



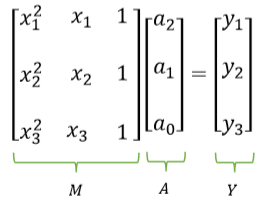
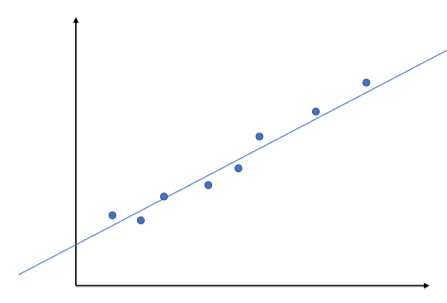
* + You could have the following hypotheses:



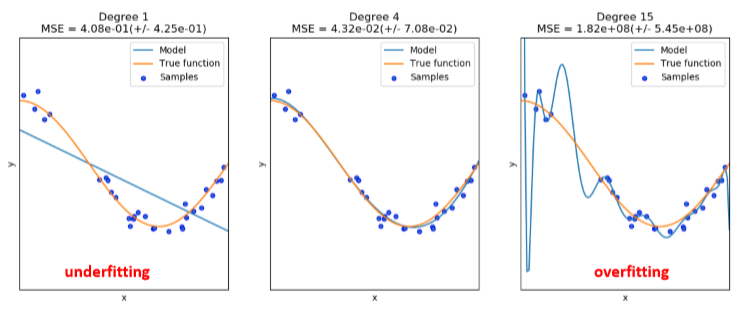
* + I think the blue curve is already overfitting
* **Overfitting:** The generated model is overly complex, and too closely tries to capture the idiosyncrasies in the data under study instead of capturing the overall pattern.
  + The model will then fail to accurately predict future (previously unseen) observations.
  + Here’s a more extreme example of overfitting:



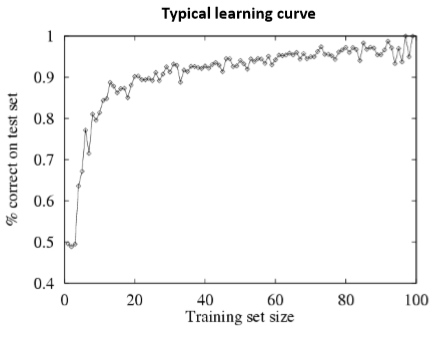
* **Occam’s razor:** Prefer the simplest hypothesis consistent with the data (e.g. green)
* **Polynomial Models:** You can *always* find a polynomial that *exactly* fits the data
  + Say you have *n* data points
  + Consider the following hypothesis, for *n* data points, this model will give an exact fit to the data
  + For example, say , you can make three equations, one for each data point:
  + Which you can rewrite as a matrix equation:



* + **This is almost never a good idea**
  + E.g. the example to the right has 8 data points but a linear model captures the data very well

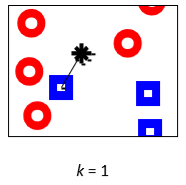


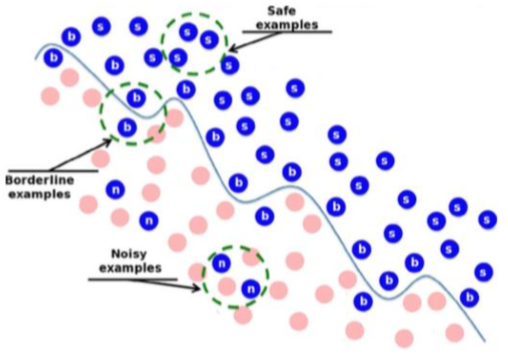
* **Avoiding Overfitting the Model**
  1. Divide the data that you have into a distinct ***training set*** and ***test set***
  2. Use only the training set to train your model
  3. Verify performance using the test set
     + Do that using **error rate, accuracy, F1 score**
  + **Drawback**: The data withheld for the test set is not used for training. E.g.
    - 50-50 split: You didn’t train on half of the data (waste)
    - 90-10 split: You might not get good measure of accuracy
* **K-fold Cross-Validation**
  1. Divide the data into *k* equal subsets
  2. Run the learning algorithm *k* times, each time leaving of the data out to test on, and use the rest for training
  3. The average error rate of all *k* rounds is a better estimate of the model accuracy
  + is usually 5 or 10
  + (number of samples) is **leave-one-out cross-validation**
* **More Data is Usually Better:**

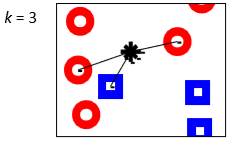


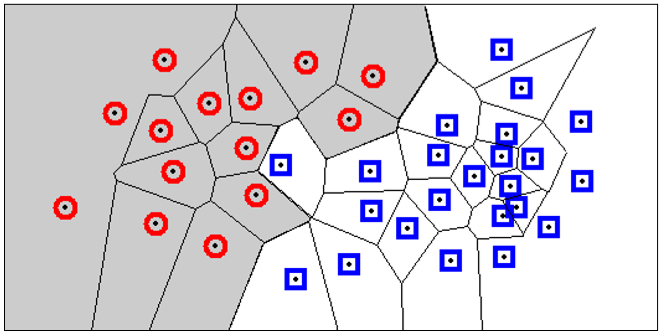
* Some Classification Algorithms
  + K-Nearest Neighbors
  + Support Vector Machines
  + Decision Trees
  + Neural Networks
  + Logistic Regression
  + …
* Types of classification:
  + **Binary:** There are only two classes
  + **Multiclass:** There are three or more classes (e.g. “pigeons”, “sparrows”, “crows”, “eagle”, …)
* Some algorithms are designed for more than two classes, others are inherently binary algorithms
  + You can turn binary classifiers into multinomial classifiers by a variety of strategies

**K-Nearest Neighbors**

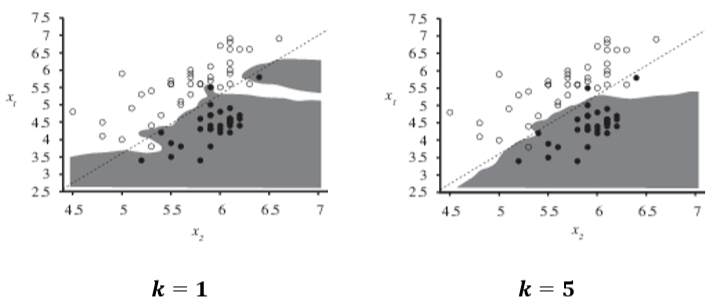
* **Learning Algorithm:**
  + Store all training examples
* **Prediction Algorithm:**
  + Classify the new example by finding the training example that is most similar
  + Return the corresponding label:
  + So you just store everything and when you’re queried, you return the data point you have that is most similar to the one asked for
* To classify new data , you need to solve:
* **1-NN:** If you’re only using one neighbor, you assign the query point *q* to the class of the nearest neighbor
  + Most data contains random, meaningless information (**noise**), which makes this inaccurate
  + There are also large regions that don’t contain training samples, so the nearest neighbor might be far away, and not be a good predictor
* Examples of noisy data:



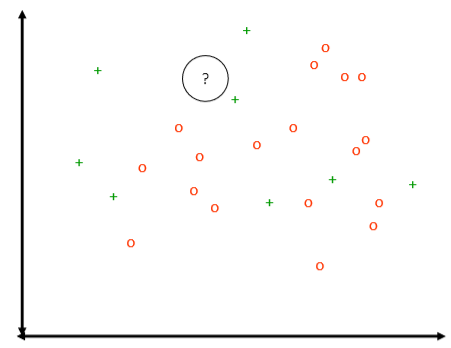
* **K-NN:**
  + Compute the *k* nearest neighbors and assign the class by majority vote
  + Increasing *k* makes the algorithm more resilient to noise in the data but can have some other side effects
* For 1-NN, here’s a **Voronoi Diagram**, which shows which class a point would be assigned to based on where it falls:

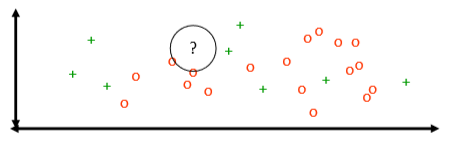


* The effect of *k* on the decision boundary:



* **Calculating the closes neighbors:**
  + You can use the Minkowski Distance ( norm):
  + *i* is the dimensionality of the data
  + **When , then is just the Euclidean distance**
* **Distance-Weighted k-NN:**
  + Should points closer to the query have more influence than those far away?
  + Weigh the value that each one of the *k* neighbors contributes to voting by a weight value corresponding to the distance
  + I.e. for neighbor
* **Disadvantages of k-NN:**
  + The scaling of different features can make it difficult to find a good distance function
    - Solution: Normalize all feature values
  + This doesn’t perform well when the dimensionality of the feature space is very high
  + E.g.:





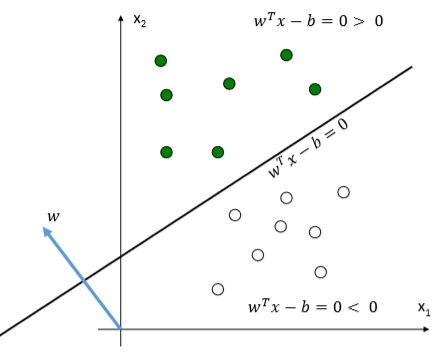


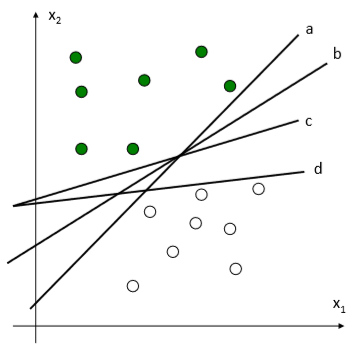
* + This also allows you to remove irrelevant features and to reduce noise, thus improving classification

# Support Vector Machines

* The supervised classification problem
  + Given a labeled data set, determine a decision rule that gives the correct class for each sample
  + Geometrically, you can interpret this rule as a curve (or surface) that partitions the feature space
  + Simpler decision surfaces are preferable
  + The simplest possible decision surface is a line (hyper-plane in higher dimensions) 🡪 **linear classifier**

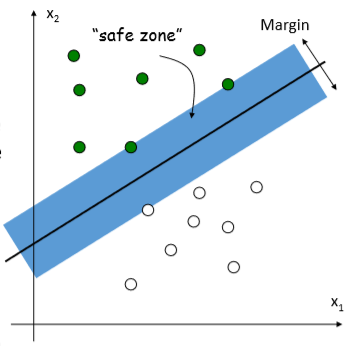
**Linear Discriminant Function**

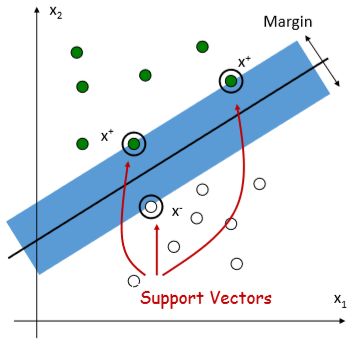
* + Classes are **linearly separable** if they can be separated by a single line
  + Lines, planes, and hyper-planes can be described by the equation
  + is orthogonal to the line, plane or hyperplane
* In the case to the right, the decision rule is:
  + If 🡪 Green
  + If 🡪 White
* To see how this works, recall that the projection of a point onto is given by:
  + So we can define a line as the set of all that satisfy the equation
  + Which can be written as:
  + With:
* Take a look at the green and white points we’ve been showing, how would you classigy them using a linear discriminant function such that error rate is minimized?
  + There’s an infinite number of answers:



**Maximum Margin Linear Classifier**

* The linear function with the **maximum margin** is the best
* **Margin:** the width that the boundary could be increased by before hitting a data point
* **Having a large margin is good** because it is the most robust to outliers, so it is the most generalizable

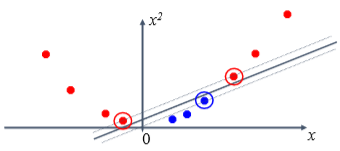


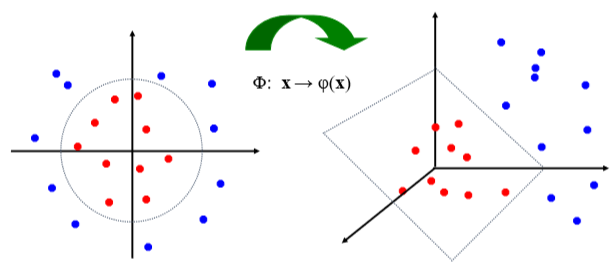
* + To maximize the margin, **minimize** subject to the constraint that all points are correctly classified
* **Linear *Support Vector Machine (SVM)* classifier**
  + The support vectors satisfy the equations for the lines that are on the margin (they *define* the margin):
* Datasets that are linearly separable work out great this way:

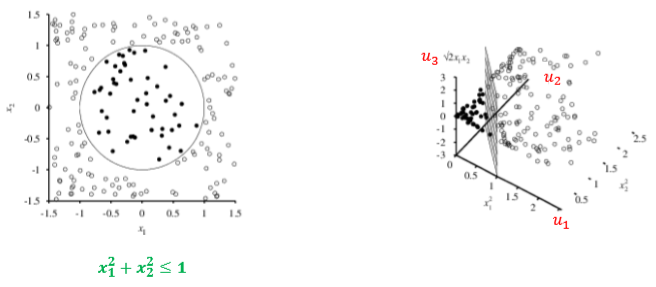


* + But what about datasets that are not linearly separable? Can SVMs help?

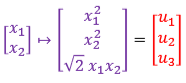


* + Yes! You have to map the data to a higher-dimensional space to make it linearly separable
  + Think of it as a **nonlinear change of coordinates:**
* [x] is the original feature, [u1, u2]T is the transformed vector
* **The Kernel Trick:** The original input space can be mapped to a higher-dimensional feature space where the training set is separable

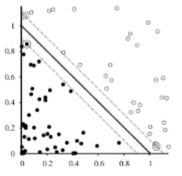




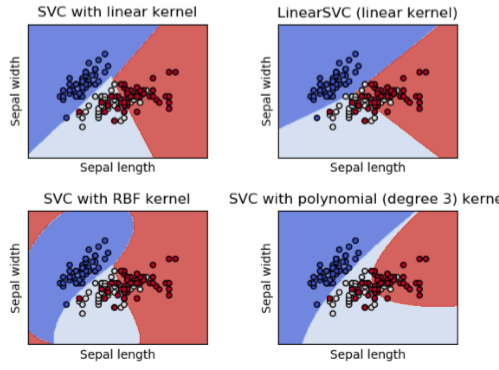
* + Nonlinear change of coordinates



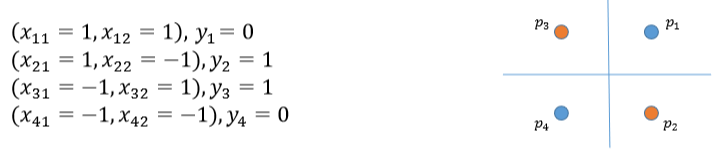
* + Closeup of the decision boundary:



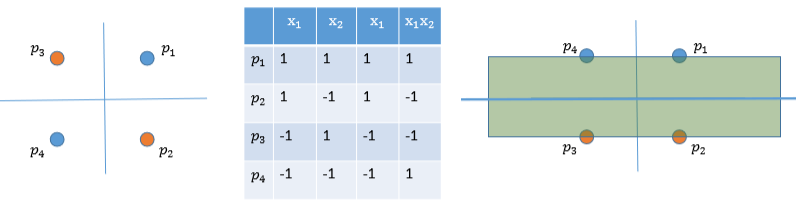
* Different kernels result in different classification boundaries



* Trying SVM by hand:
  + Construct an SVM that computes the XOR function, consider:



* + I thin means blue and means red
* To define a transformation that will make these points linearly separable:
  + We need some mapping that makes the given points linearly separable
  + There are multiple answers (the third answer below adds a dimension):
  + Note that adding a third dimension that is a constant does not help, e.g.
* What is the margin for the answers above?
  + It depends on the formula, but the margin is the distance between the decision line and the support vectors
  + For both of the first two transformations above, the margin is 1



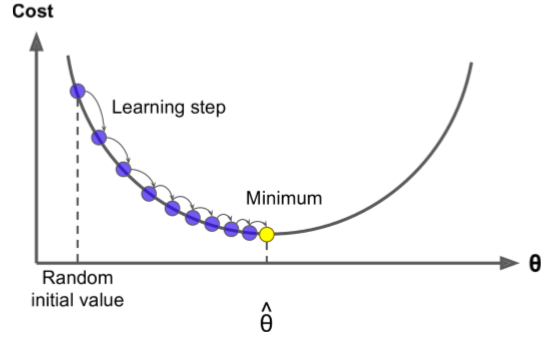
* **Multiclass SVMs**
  + **One-against-all classification:**
    - Train a single classifier per class, using the samples of that class as positive samples and all other samples as negatives (*N* total classifiers for *N* classes)
    - This requires the base classifiers to produce a real-valued confidence score for its decision of that class. Then I think that when you try to classify a point, you apply all classifiers and the one with highest confidence is chosen
  + **One-against-one classification:**
    - Train binary classifiers for a -way multiclass problem
    - Each receives the samples of a pair of classes from the original training set and must learn to distinguish the two classes
    - At prediction, all classifiers are applied to the new sample and the class that got the highest number of “+1” predictions gets predicted by the combined classifier

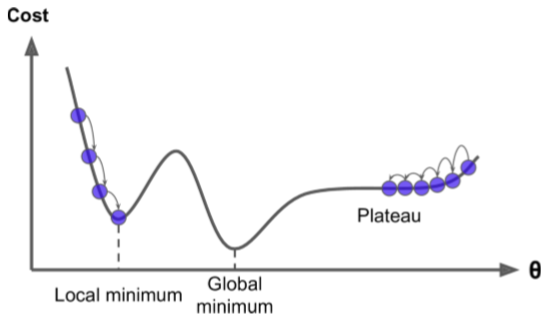
**Additional Resources:**

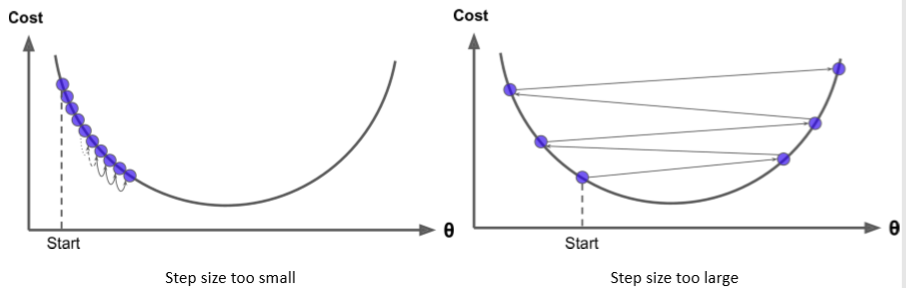
* <https://brohrer.github.io/blog.html>
* <http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
* <http://scikit-learn.org/stable/modules/svm.html>

**Parameter Optimization**

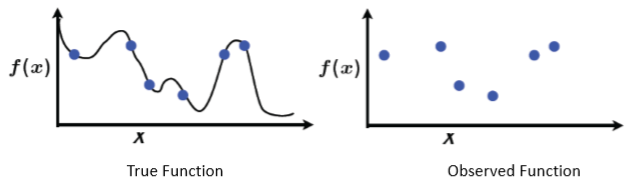
* Robotic systems have many parameters
  + Control parameters
  + Odometry parameters
  + Kalman filter parameters
  + …
* How can we optimize for these efficiently?
  + **Grid search (not efficient):** Create nested for loops that iterate over all possible combinations of parameters.
  + **Gradient Descent/Hill Climbing:** Find optimum by iteratively following the gradient/slope of the parameter space
* **Gradient Descent:**
  + Keeps track of the current state and changes the state to (hopefully) improve performance
  + This is used for:
    - Optimization problems
    - Scheduling
    - Task assignment
    - … Many other problems that seek to find the best state according to some objective function







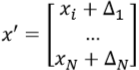
* **Overcoming Local Optima**
  + Try again: Run the algorithm several times and return the best solution, change the initial start location randomly
  + Sideways moves: If stuck on a plateau, we can allow flat moves for a while and eventually become unstuck
  + Other approaches: look ahead, simulated annealing, …
* **Challenge:** A problem is that we usually have no idea what function we’re trying to minimize
* We can can only compute the function at a finite number of points and evaluation is expensive



* **What if we don’t know and the gradient?**
  + You can estimate the partial derivative with respect to each dimension using sampling
    - Take current estimate:



* + - Create *k* new estimates, such that



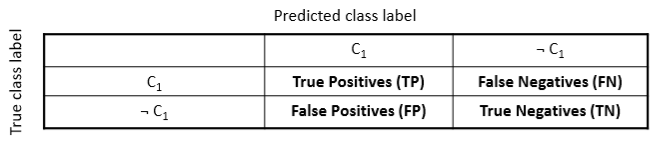
* + - Evaluate each estimate using objective function *f*
    - Estimate gradient of objective function *f* at *x*
  + This gets impractical if the number of parameters being estimated *N* is large
* **For Lab 1:** Grid search will likely be less expensive than gradient search because we don’t have a good way to estimate the gradient except for sampling. But getting the samples needed for this estimate is probably more expensive than the full brute force search since our parameter space is relatively small.

## Classification Evaluation Metrics

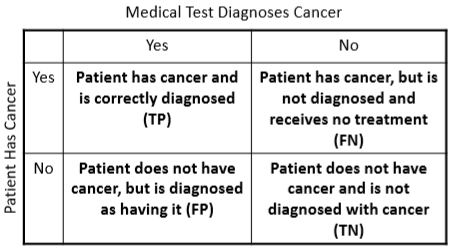
* Most common metrics:
  + Confusion Matrix
  + Accuracy
  + Precision and Recall
  + F measure
* We’re going to use binary classification examples but they generalize to multi-class classifications

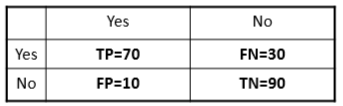
**Confusion Matrix**

* The basis for analyzing what’s going wrong and for calculating other performance measures

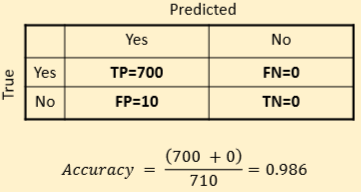


* + For example:

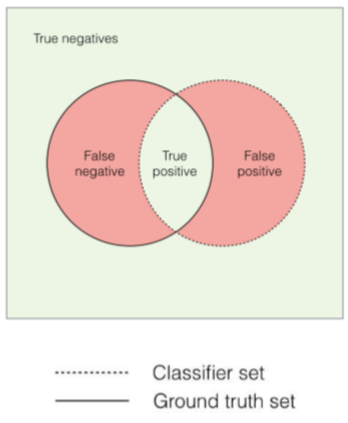




* **Accuracy:** The percentage of correctly classified instances
  + *N* is the total number of instances in the dataset
  + With the example points above:
  + This can be misleading if there is a large class imbalance (i.e. there are many more examples of one class label than another)
  + For instance, say the classifier always predicts the class that is more common (in the case below, Yes is much more common, and among the Yesses, most are true):

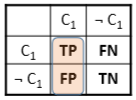
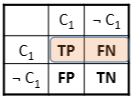


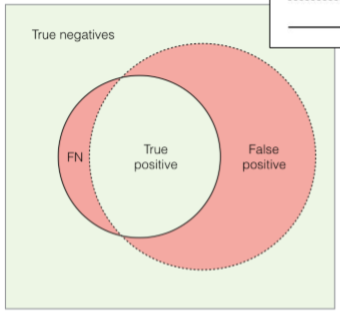
* Lets visualize the classification performance differently
  + Square: All the available data
  + Solid circle: True set of samples corresponding to some label *X*
  + Dotted circle: Predicted set of samples corresponding to some label *X*
  + The overlapping green region is true posivive samples (truly *X* and predicted as *X*)



* + Best performance is when the circles fully overlap

**Precision and Recall**

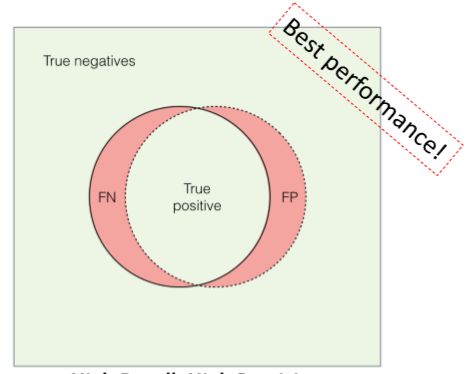
* **Precision:** The fraction of samples classified as belonging to class *X* that truly belong to that class
  + “When the classifier predicts *yes*, how often is it correct?”
* **Recall:** Of the total number of samples that truly belong to class *X*, how many does the model correctly classify?
  + “When the sample is actually *yes*, how often does the classifier predict *yes*?”
* **High Recall, Low Precision:** The classifier correctly classifies a lot of instances as class *X*, but incorrectly classifies a lot of other things as belonging to *X* as well



* **Low Recall, High Precision:** The classifier correctly classifies only a fraction of instances truly belonging to *X*, but misclassifies very few extraneous instances

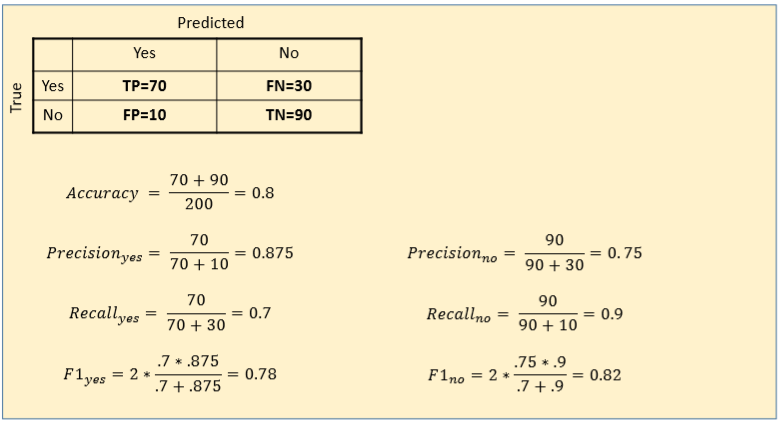


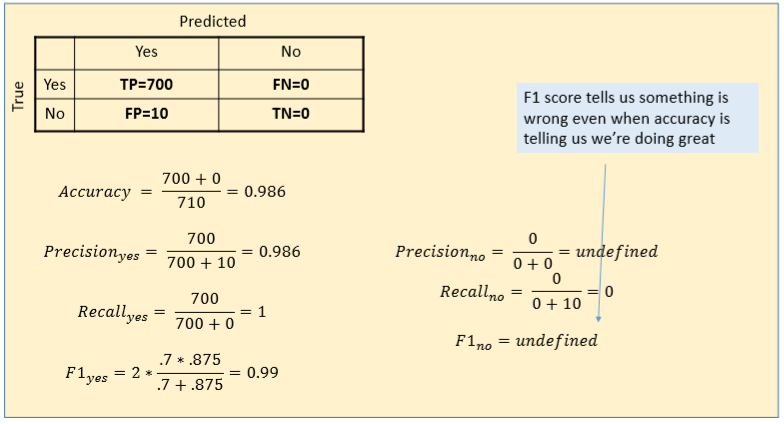
* **High Recall, High Precision:** The classifier correctly classifies most of the instances truly belonging to class *X*, and misclassifies very few extraneous instances as belonging to *X*



**F1 Measure / F1 Score**

* Combines Precision and Recall (equally) and allows for more accurate evaluation that considers both FP and FN into account
  + There’s a more general formulation, called , that can give more weight to either precision or recall. This is important for some applications
  + You need to **run this on each variable**
* Example:





* **In the multi-class case, F1 score is calculated as the weighted average of the F1 score of each class**
* **Receiver Operating Characteristic (ROC) curve:** Deals with tuning classification thresholds and not directly with measuring performance, it’s related to what we just covered but we don’t cover it.

**Takeaway**

* **If you have a balanced dataset, accuracy and F1 score are going to give you roughly the same result**
* When the dataset is unbalanced, accuracy becomes unreliable and F1 score should be used
* Look at the confusion matrix to see what’s going wrong

**Resources**

* <http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html>
* <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html>