SUPERVISED LEARNING

CLASSIFICATION CONT.

Keith Burghardt

USC Information Sciences Institute

DSCI 552 – Spring 2021

Topics this week

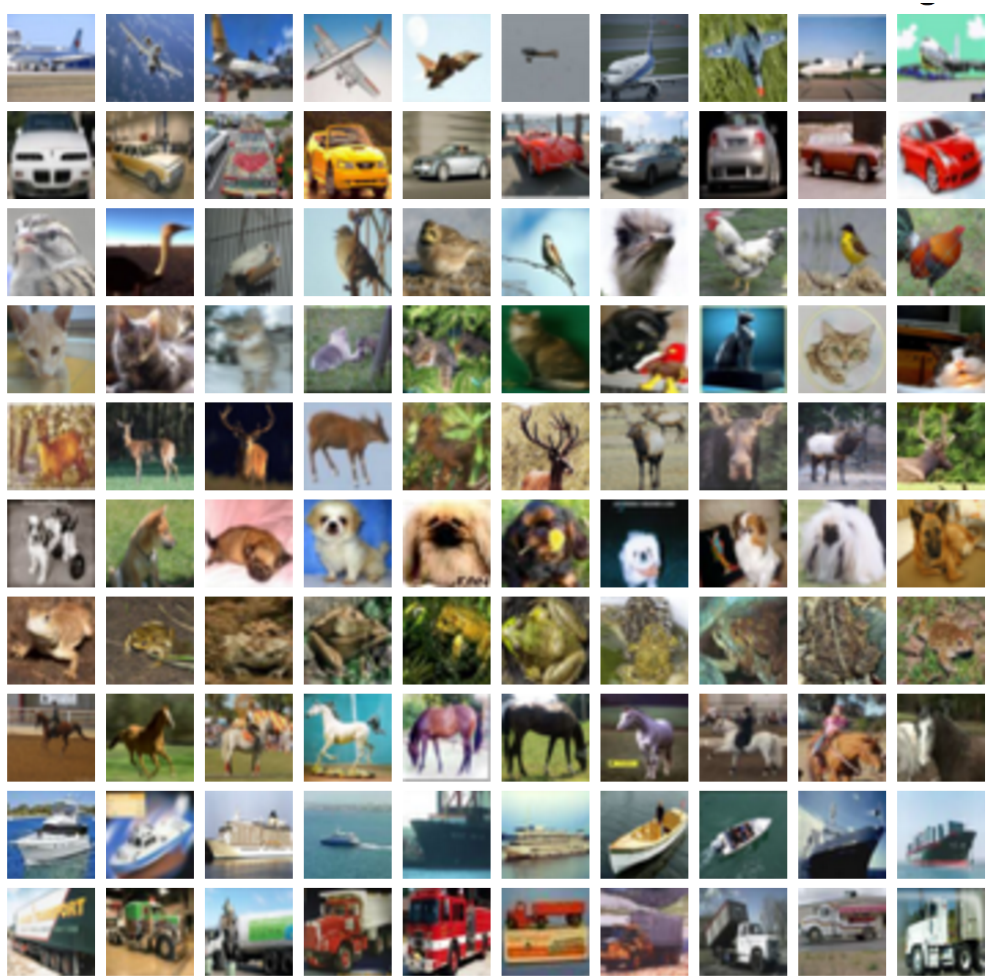
- **Reminder:**
  - Quiz 2 was due today
  - Literature review due Feb. 10
  - Quiz 3 will be posted today
- Taking a step back
- Supervised classification
  - Parametric models (ex: logistic regression)
  - K-Nearest Neighbo(u)rs
- Performance measures
  - Accuracy
  - F1
  - Receiver operating characteristic

Taking a step back

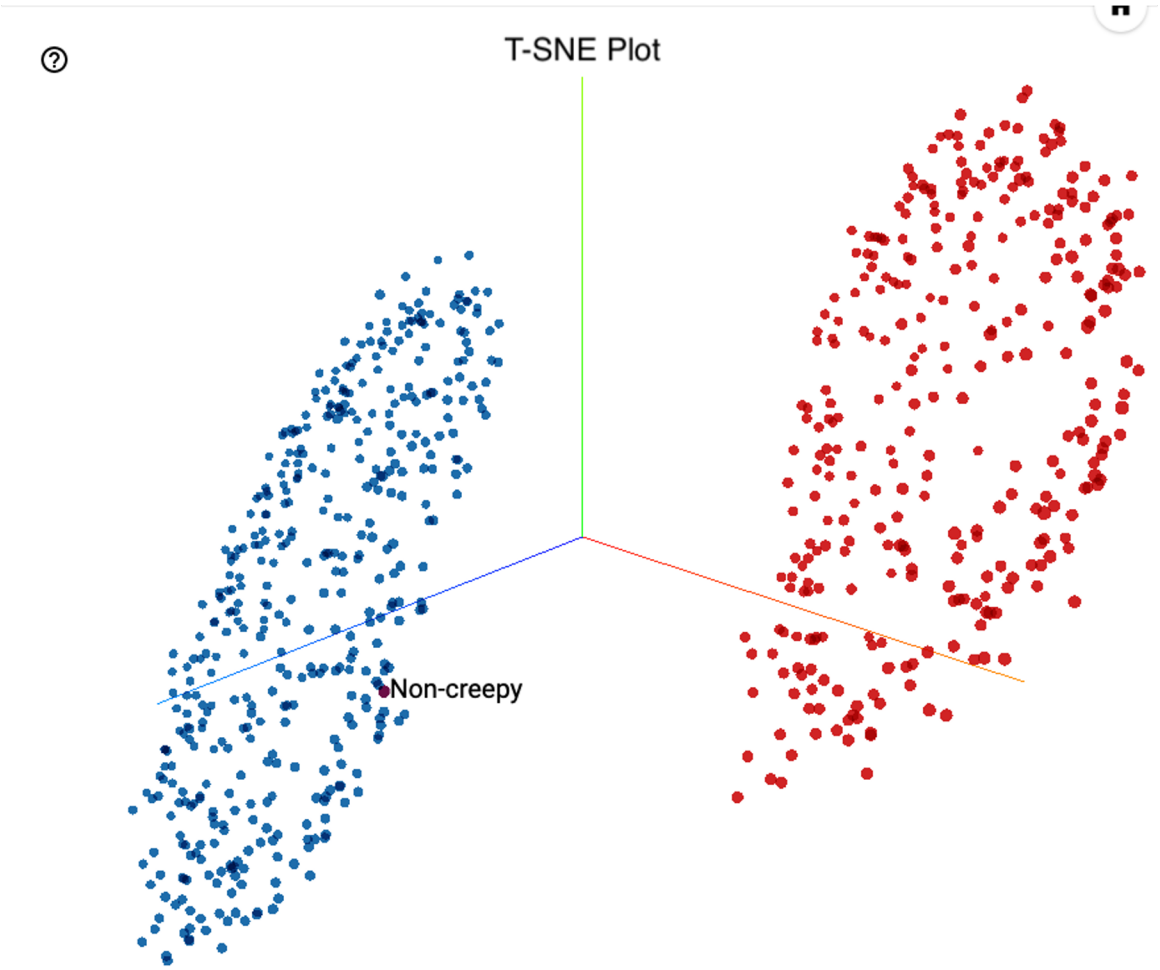
- I will avoid using any math
  - We want to give you intuition
  - Applications
- Understanding Bayesian Inference:
  - When to use and why
- Logistic regression
  - Interpretation
- K-Nearest Neighbors
- Intuition for performance metrics

Categories

- Computers like numbers
  - ‘airplane’ -> 0
  - ‘automobile’ -> 1
  - ‘bird’ -> 2
  - …
- Sometimes it is useful to turn multiple categories into binary categories
  - ‘airplane’ ->[1,0,0,0,…]
  - ‘automobile’ -> [0,1,0,0,0,…]
  - ‘bird’ ->[0,0,1,0,0,0,…]
  - “one hot encoding”



Category example: detecting creepy stories

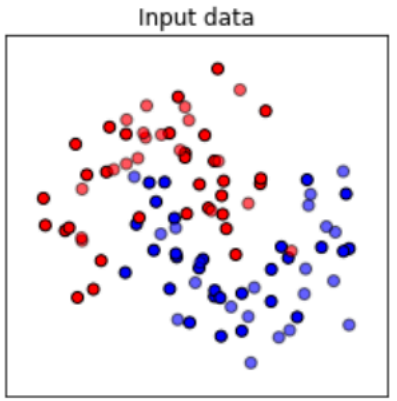


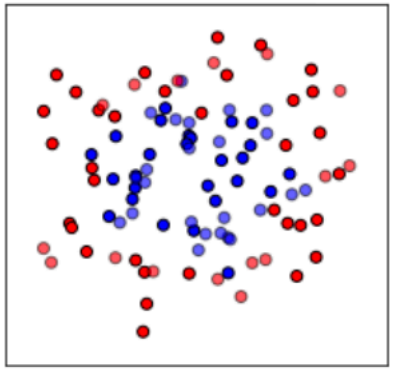
Category example: creepy or not creepy stories

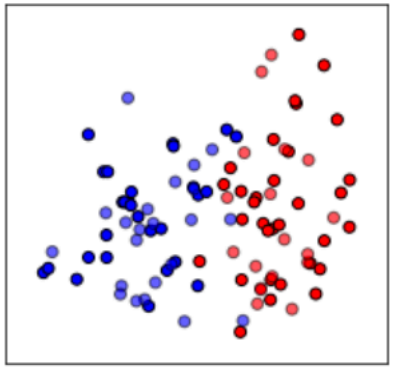
- Notice stories are turned into points in 3 dimensions
- This is called “embedding”
- We want to find embeddings that best separate categories
- Neural networks essentially turn a complex data into simple points that are well-separated
- These separate categories are clustered around “categories”
- Finding features to best separate categories is the biggest challenge of ML

How do we separate these categories?

- Imagine you have feature X1 and X2
- You have 2 classes: 0 and 1
- For each dataset, how would you separate these categories?
- I will show you a few methods needed to separate each
- Methods:
  - Linear vs. non-linear
  - Adding new features (“kernel trick”)
- Often data will not be easy to separate
- You want a pocket full of potential models to find the best

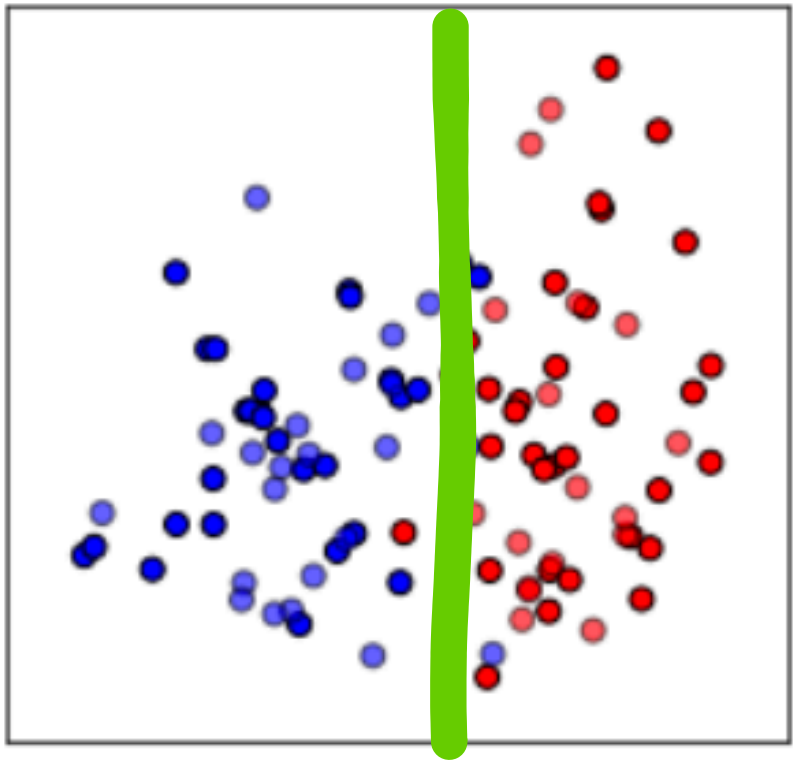






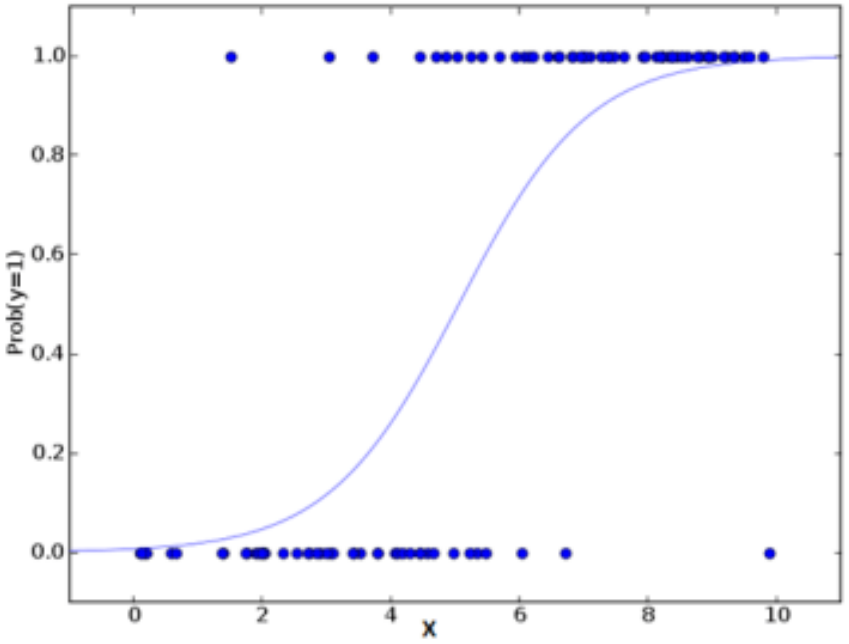
A few ways to separate categories

- Example dataset: linearly separable
- Simplest methods:
  - Support vector machine
  - Logistic regression
- **Support vector machine:**
  - Simply find the plane that best separates data
  - Left of plane: class 0, else class 1
- **Logistic regression:** gives us probabilities you're in each category



Logistic Regression

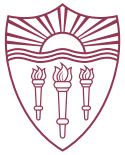
- What is it?
- Logistic(s) = $[1+\exp(-s)]^{-1}$
- When s is a large negative number, Logistic(s) approaches 0
  - You're "very sure" the data is in class "0"
- When s is a large positive number, Logistic(s) approaches 1
  - You're "very sure" the data is in class "1"
- What about in between?
  - It could be in either category



Fitting the Logistic Model

- When you fit data: Logistic(s) = Logistic ($b_0 + b_1*x_1+b_2*x_2+....$)
- Note that this output is the probability you are in class 1 (or in general, class k)
- b_0 is the "intercept" alike to linear regression
  - It says how likely are you in class 1 when all features are 0
  - Logistic(b_0) is near 0 when $b_0 < 0$
  - Logistic(b_0) is near 1 when $b_0 > 0$
- b_1 says how much changing a feature x_1 will change the chance of being in class 1 or 0
  - This is alike to the "slope"
- Similarly for b_2, b_3,… b_1

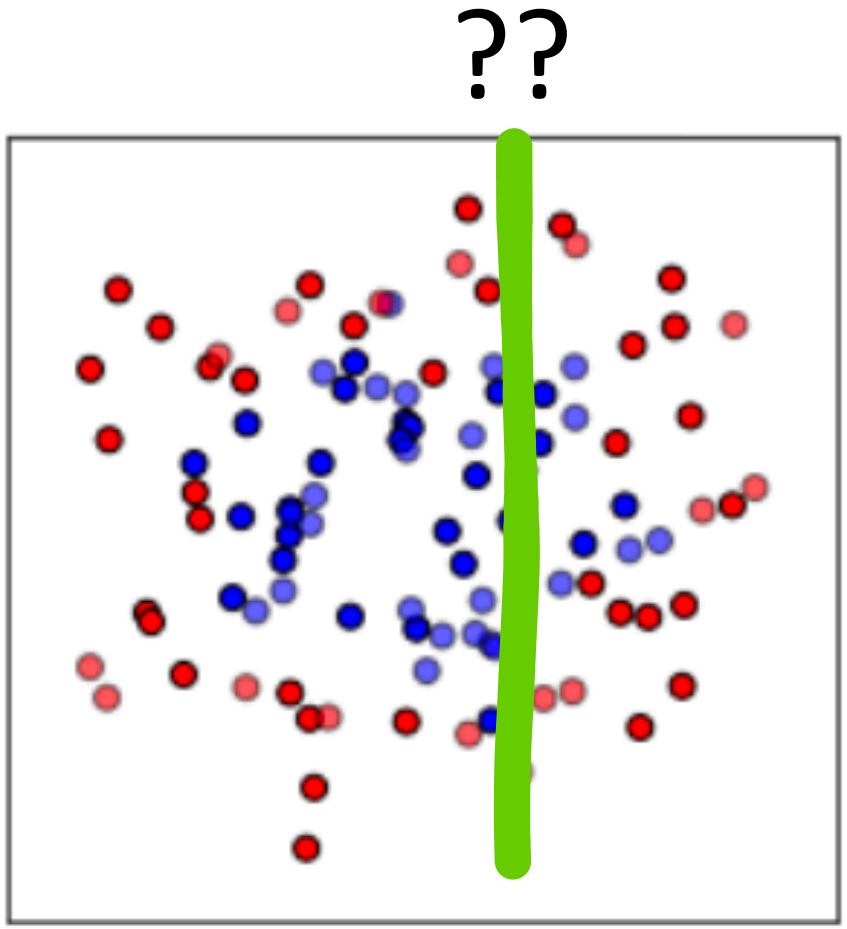
Making sense of Logistic Regression



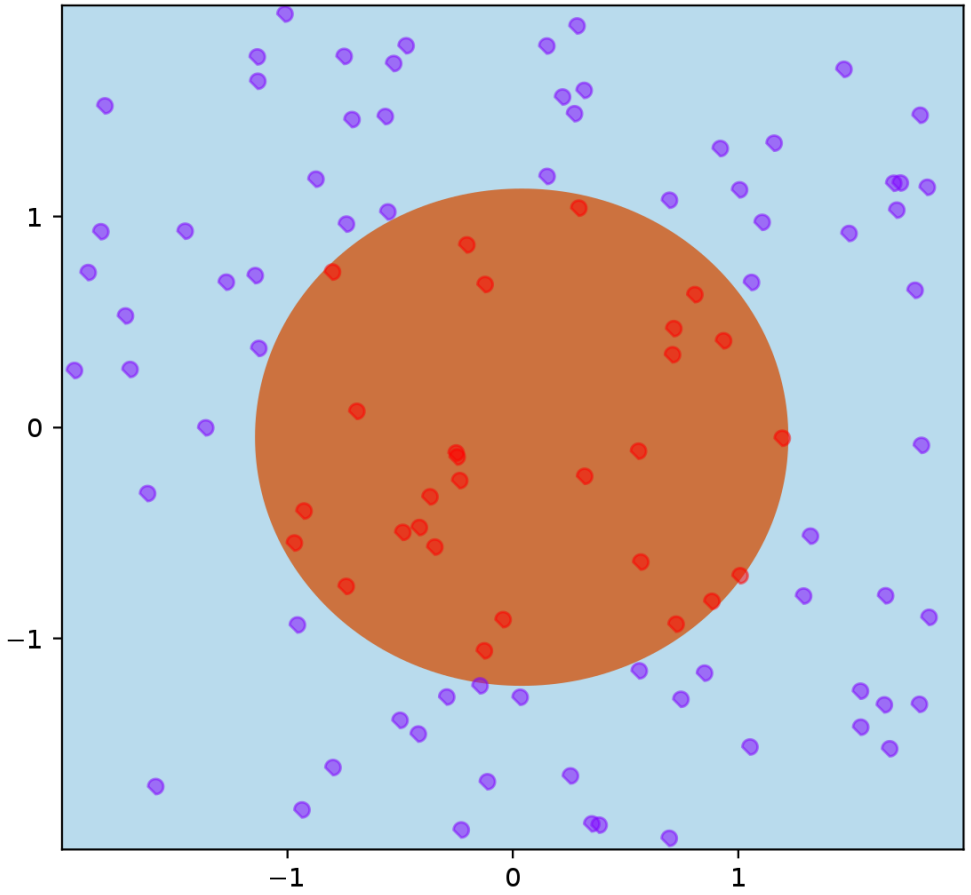
Let’s stop to explore the jupyter notebook

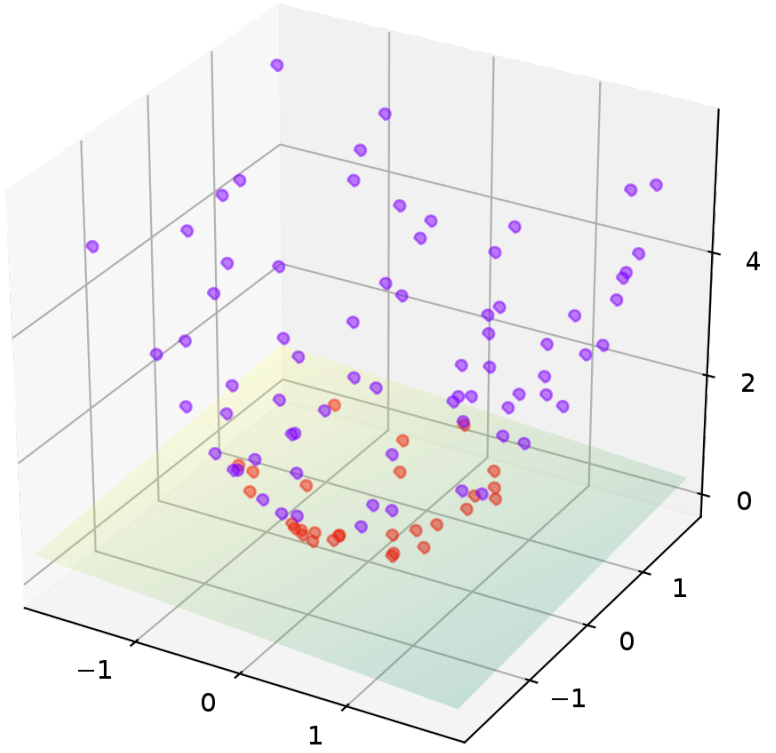
Support Vector Machine

- Separate data into a “hyperplane”
- What if your data looks like this?
- Don’t give up!
- Remember your goal is to transform your data such that classes are easy to separate
- How might you do this?



Kernel Trick



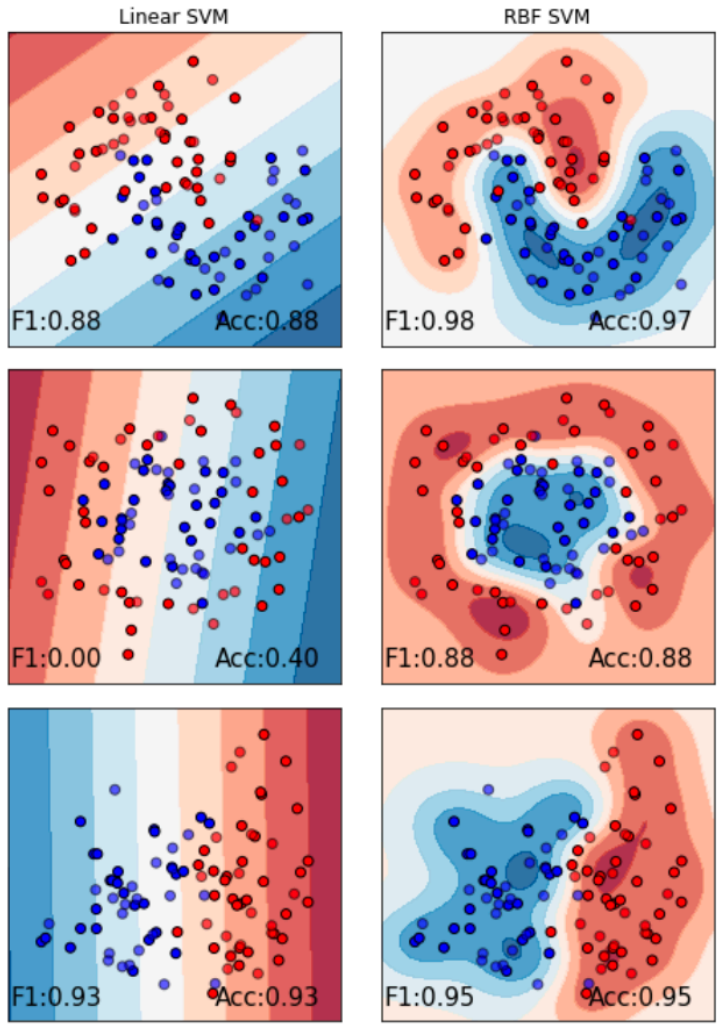


Kernel Trick

- "Kernel" = way to measure distances
- Linear kernel: |2 – 1| or |3 - 4|
  - More generally $|X^i - X^j|$
- RBF Kernel
  - "Radial Basis Function"
  - If you are curious $Exp(|X^i - X^j|^2/2\sigma^2)$
  - They are the default in sklearn
- Kernels act as distances in a transformed space
- You don't have to change the features for the SVM to work

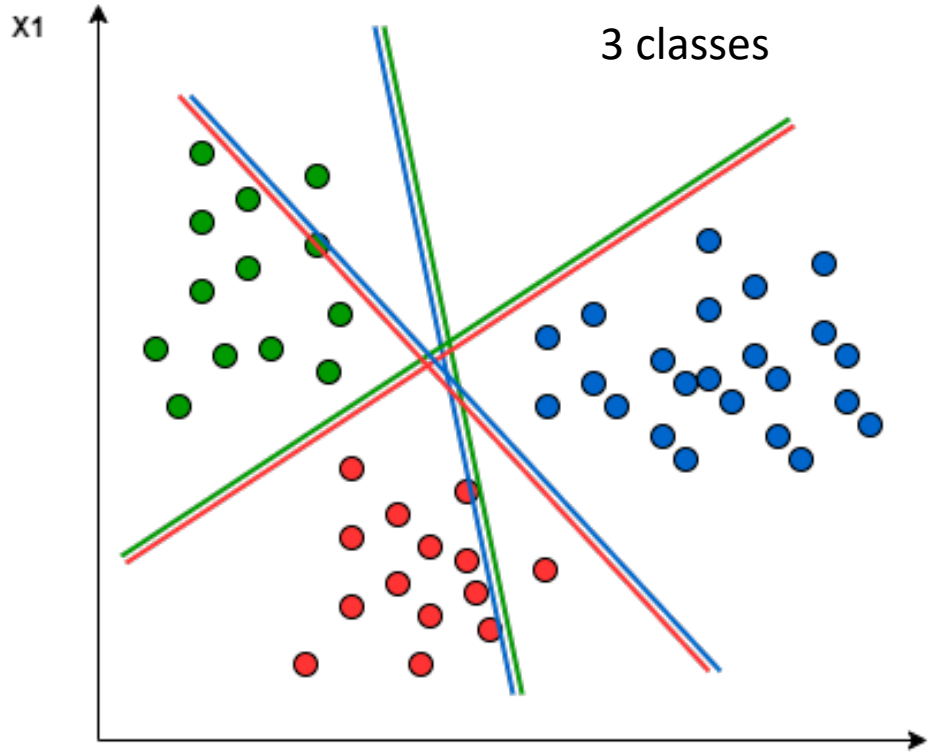
How well this works in practice

- Linear kernel (naïve method)
  - Consistently bad, especially for middle dataset
- RBF: works well in practice
  - Can be proven in certain models to be a universal classifier
  - Imagine class 0 datapoints = ant hill, class 1 datapoints = hole
  - Bunch of datapoints = large dirt pile surrounded by moat
  - SVM = elevation that separates “hill” from “mote”
  - Vary “width” of hill/hole to best separate test data



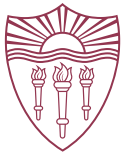
Multi-Class SVM

- Intuition:
  - Easy to separate data into planes
- Check if you are in class 1:
  - “not in class 0”
  - “not in class 2”
  - “in class 3”
- Use planes to filter out likely class
- Choose class that most models pick, or with highest certainty



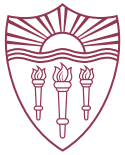
One-vs-one
One-vs-rest

Naïve Bayes



- Issue with SVM: kernels take a lot of time to train
- Why spend a day to make a good model, when other good ones can be found in a few minutes?
- In industry, you will find a trade-off between accuracy (e.g., predicting if someone will pay back a loan) and training time
- "How well does a model work?"
  - Absolute sense, e.g., "accuracy"
  - Relative sense: compare to super-simple methods like ***Naïve Bayes***
- This is not meant to be a method you should always use, but a model you can compare to

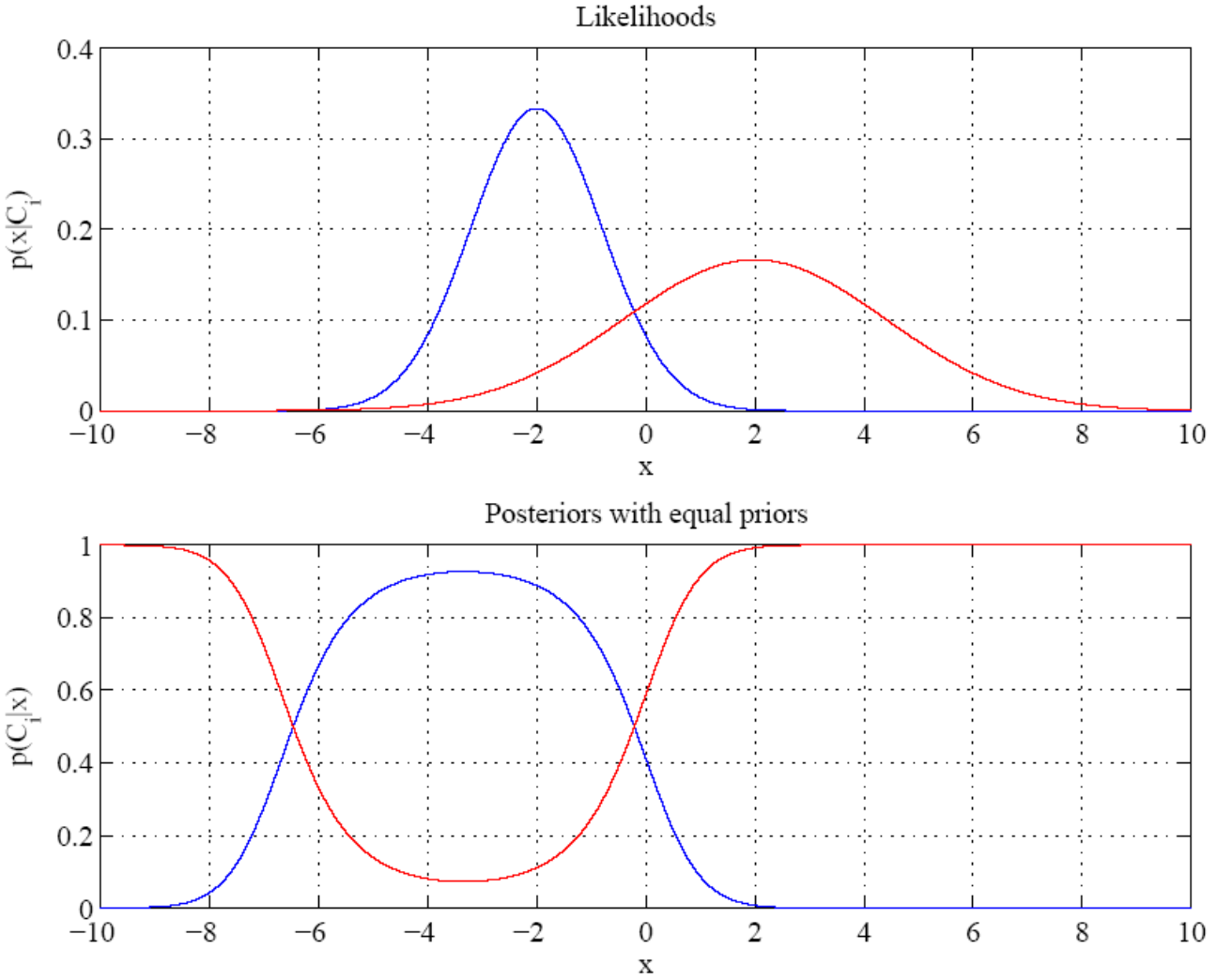
Naïve Bayes



- What is it?
- Sometimes features are correlated
  - Distinguish an apple from a basketball
  - “green” is correlated with “apple”
  - So is size (small)
  - So in this case size and color are correlated
- Naïve Bayes assumes no correlations (hence naïve)

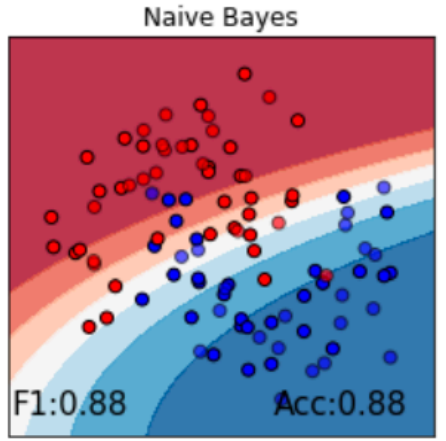
Naïve Bayes

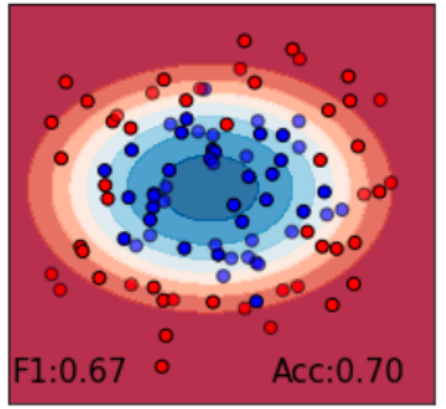
- For example: likelihood = Gaussian distribution
- Find the relative probability data is class 0 or 1
- Naïve Bayes in multiple dimensions
  - Pr(any datapoint in class 1) (prior)
  - Times likelihood class 1 has feature feature with value x_1
  - Times likelihood class 1 has value x_2

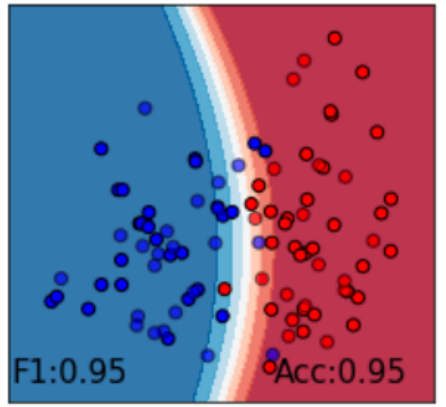


How well does this work?

- Pretty well!
  - Dataset 1: pretty clear separation
  - Dataset 2: better than nothing ¯_(ツ)_/¯
  - Dataset 3: very good!
- You will notice the numbers next to “F1” and “accuracy”
  - I will explain them shortly
  - Basically, think larger is good
  - If you added new data from the same distribution, these metrics say how well the new data is correctly classified







K-Nearest Neighbors

- This is another example of a “baseline”
- Imagine you have training data with known classes
- You find new data “nearby” trained data
- Assume that nearby data with known class is alike to new data

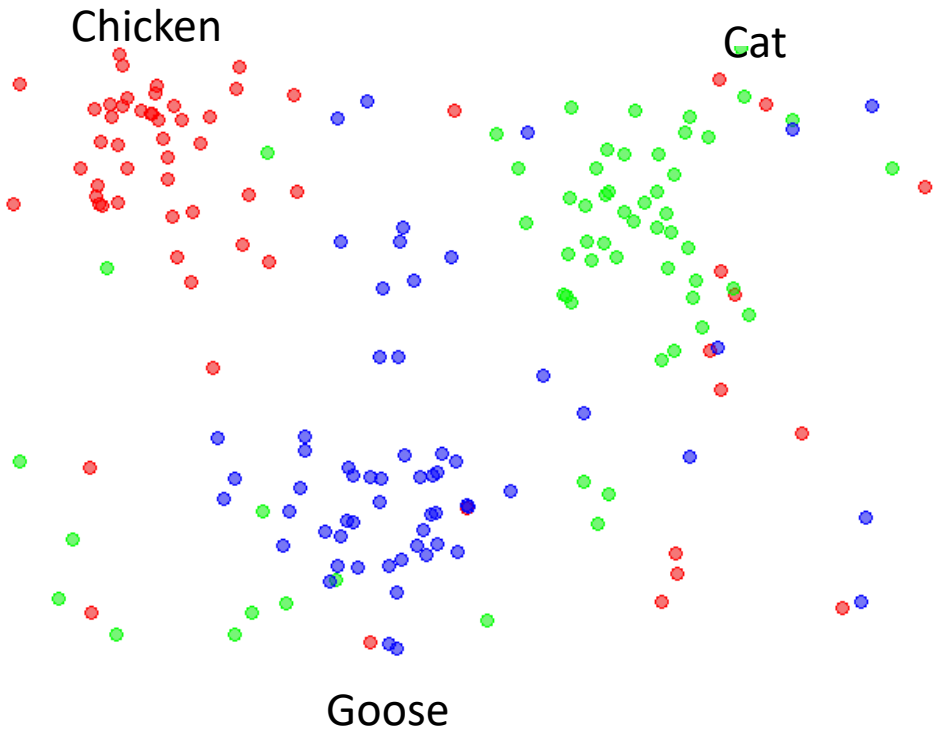
Human example

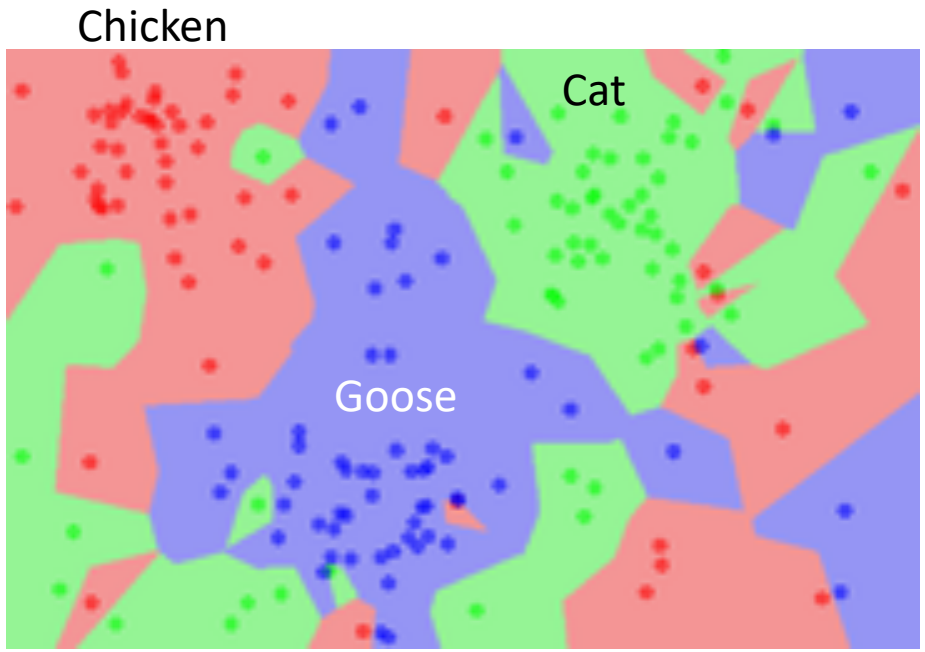
What is going on?

Kids use machine learning and don't even know it!

??

”Correctly Trained” animal recognition





What is going on?

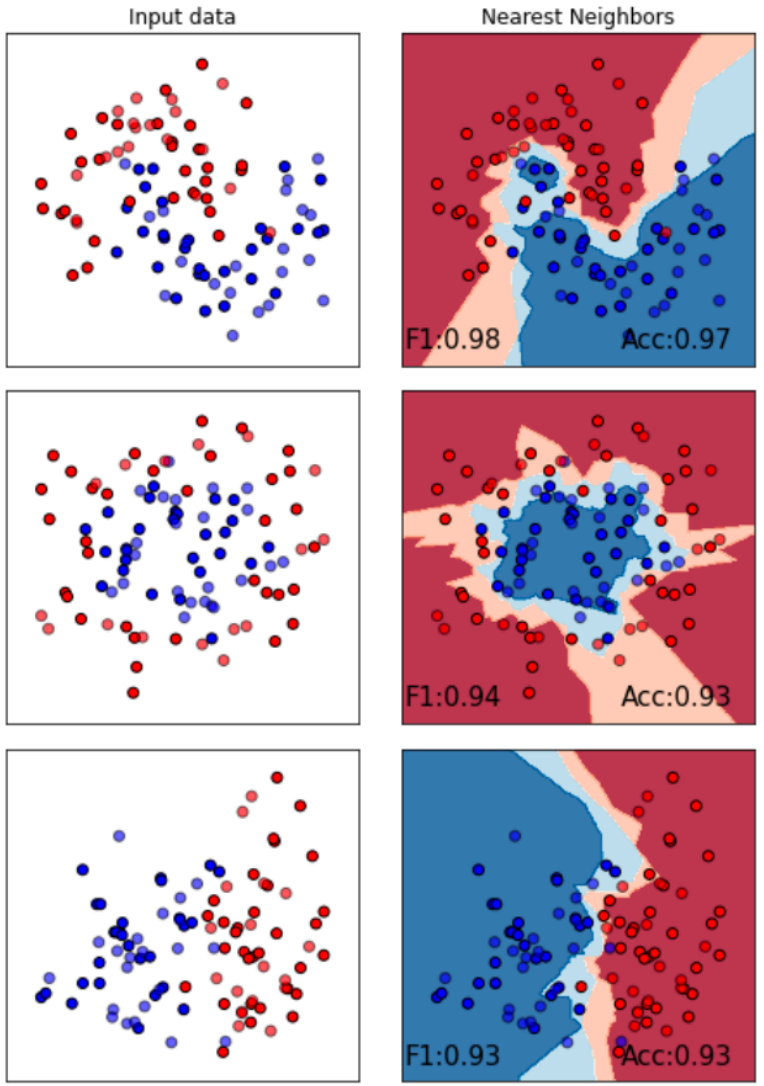
- Clearly, if we have some sense of “distance” we can correctly categorize animals
- Humans have a very natural way of determining “distances” of images
- Some neural networks do this for data
  - Take complex data
  - Simplify it
  - Embed simplified data in such a way that nearby classes are “similar”
- Hence child is wrong but for very good reasons!

K-Nearest Neighbors

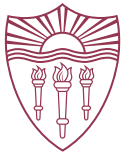
- K-Nearest Neighbors looks at data in a similar way
- Given trained data, test data class is based on trained data classes, weighted by distance to nearest trained data
- Problems:
  - "Distance" is a loose concept
  - How to weigh "nearby" data is not obvious
  - How many "nearby" data in training set to compare to is not obvious
  - We run into difficulty with too many features
    - With though features, all data is "close" to each other
- The values "K", "distance", and "weight" are called *hyperparameters*
- We change hyperparameters until test/validation data is best labeled

How well it works in practice

- Clearly, it works pretty well!
- It makes few assumptions of decision “boundary”
  - i.e., where we call data class 0 or 1
- Therefore it works well for all 3 datasets!
- Note limitation with too many features
- We will discuss later how to pick “relevant” features



Scenario



- What do you do when your boss asks you “what customers are likely to default on a loan?”
  - You try to collect as much data as you can!
  - Set aside some of your data to test your models
    - You can also use “validation,” where data is cut into slices, train on 80% of slices, test on other slices, repeat on each held out slice
  - Now find a suite of models to compare against
  - You may be tempted to use the latest and greatest methods
  - But don’t be fooled! Some methods are not much better than simple methods
  - New methods may be complicated, have buggy code, or take a long time to test/train

You collected your data and features. Now what?

- Latest and greatest models
  - E.g., neural networks
  - They can be complicated and hard to train
  - But they could have a significant payoff!
  - Trade-off between accuracy and time
- Baseline models
  - Logistic regression
  - Support vector machine
    - Note kernel trick!
  - Naïve Bayes
  - K-Nearest Neighbors

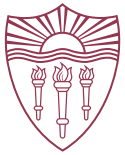
Why different models work?

- Logistic regression
  - Need to know uncertainty (class 0 or 1?)
  - Assumes classes are roughly on either side of a hyperplane
- SVM
  - Hard boundary between classes (classes should be well-separated)
  - Performance depends on features
  - “Kernel trick” = predict features are transformed, then model can work well
  - Unknown what kernel to use, hard to train on a lot of data
- Naïve Bayes
  - Simple assumption about data, works until it does not

Why different models work?

- KNN
  - VERY Simple method
  - Makes fewest assumptions
  - E.g., “decision boundary” (what is class 0 or 1) can be anything
  - Works well in all 3 example datasets
  - Problems:
    - Works poorly with too many features
    - Need to adjust hyperparameters (use for loops across possible values)
    - E.g., change distance, change K, change weight of trained data
  - Don’t expect this to always work, but again works well as a baseline

When Baselines May Fail



- High dimensional data
  - E.g., hard for SVM to understand what an image is
  - KNN works poorly
- Complex decision boundaries
  - Bad for logistic regression
  - Bad for Naïve Bayes
- A lot of data
  - Bad for SVM
- When all these properties exist, **you want to explore newest methods**
  - E.g., neural networks of various forms
  - Note neural networks can get very complex (many parameters/architectures)

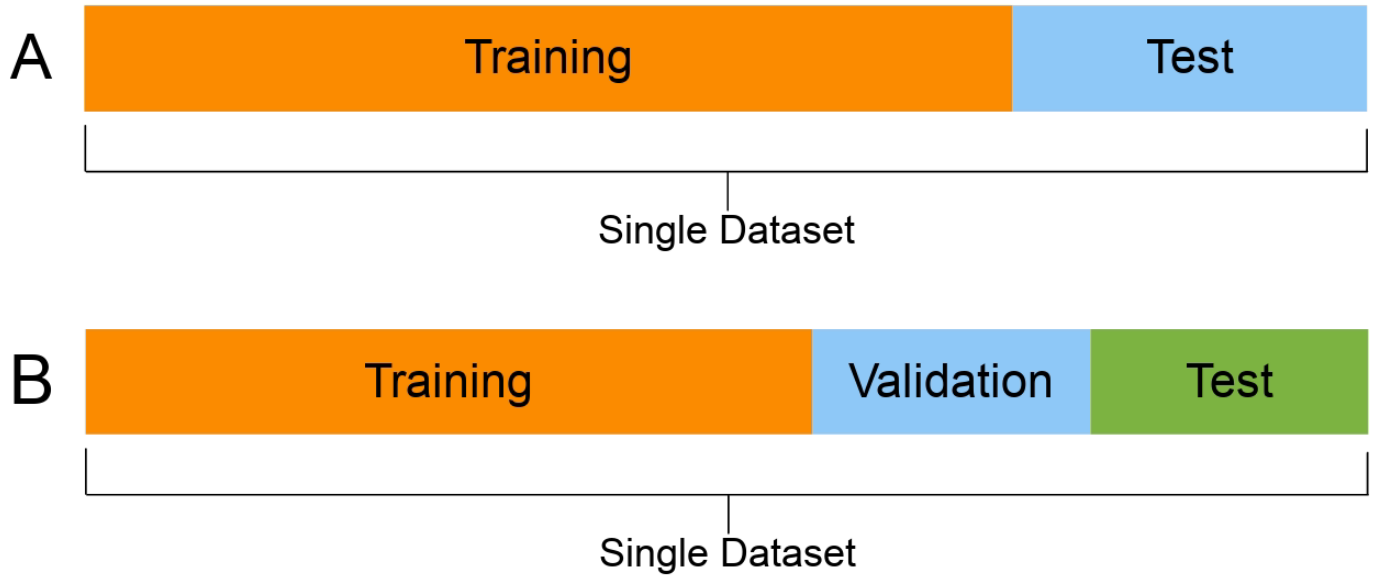
PERFORMANCE MEASURES

Previously

- We showed some reasonable baseline models
- We show they sometimes perform “well” and sometime not
- What is “good” performance?
- How do we test models?

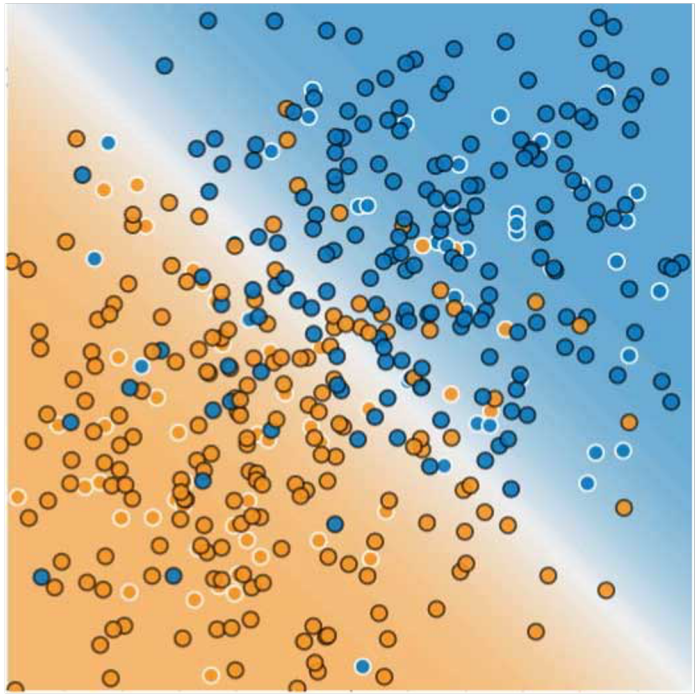
Training and testing

- **Training:** data you give to your model so it can separate classes
- **Validation:** use a subset of data to determine the best hyperparameters
- **Testing:** the last step. See if the model performs well

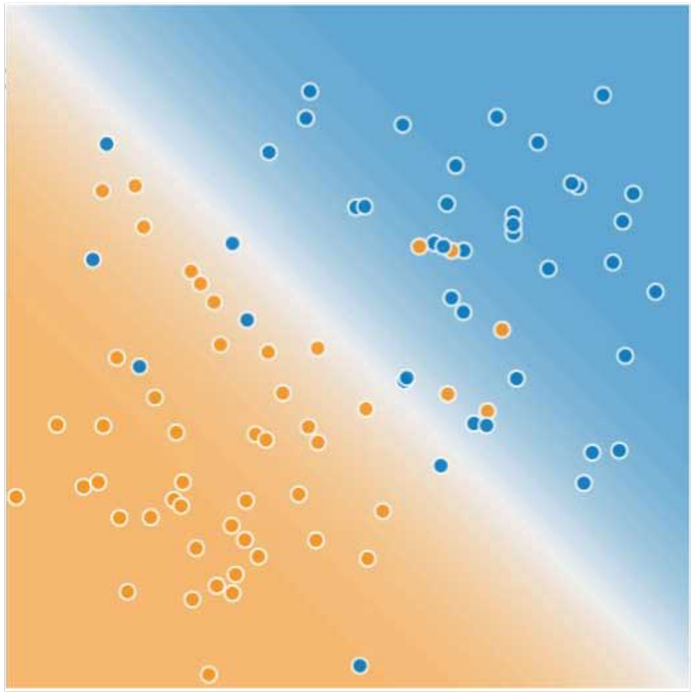


Example

- N.B., never use test data for any part of training
- Notice test data is randomly distributed
- Imaging test data came from same distribution



Training Data



Test Data

https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data

Caveats

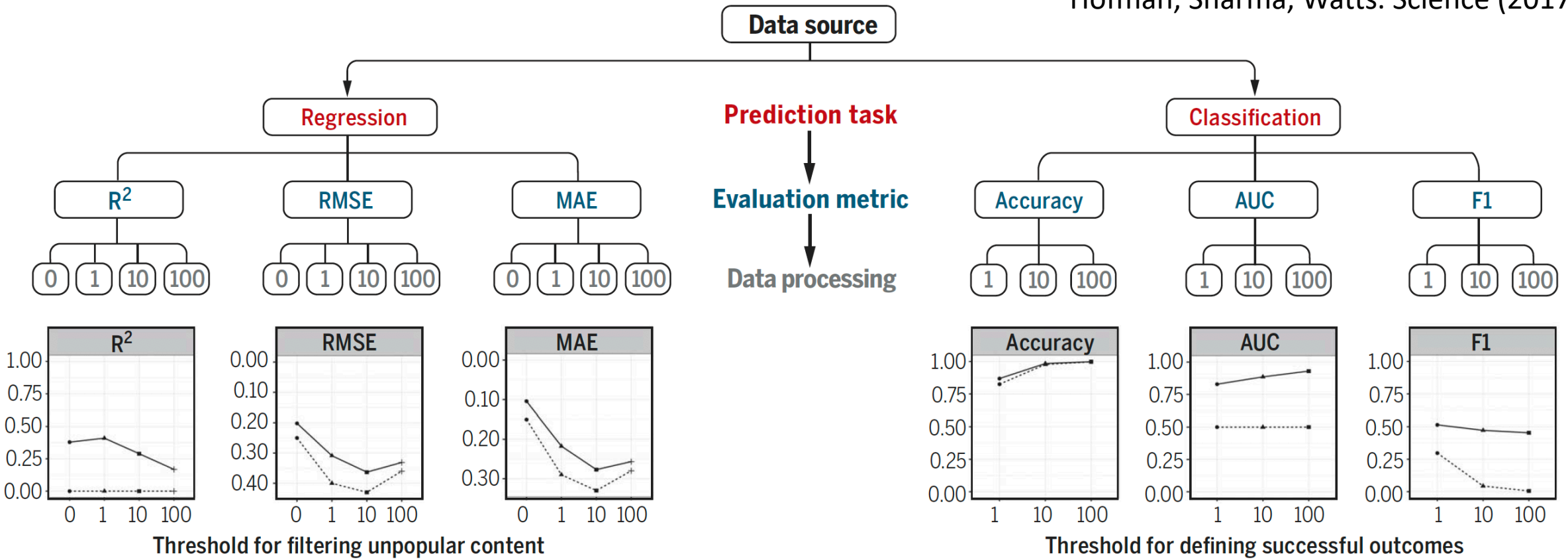
- Note that test data is meant to simulate realistic scenarios, e.g., new customers
- New data in your company will not always have the same distribution as training data
- “Data shift” problem (we will discuss this later)
- Effect: real data usually has poorer performance than test data
- Make sure model can generalize well and new data is not near decision boundaries (where performance is worst)

Now you have test data, so what?

- Given test data, use your trained model to make predictions
- Compare predictions to test data "ground truth" i.e., correct data
- Find out if model performs well using *performance metrics*
- Choose the best model if its performance metrics are highest
- Similarly, when choosing best hyperparameters, choose those that increase validation performance most
- Make sure to stick to consistent performance metric in training and testing

The Problem with Performance Metrics

Hofman, Sharma, Watts. Science (2017)

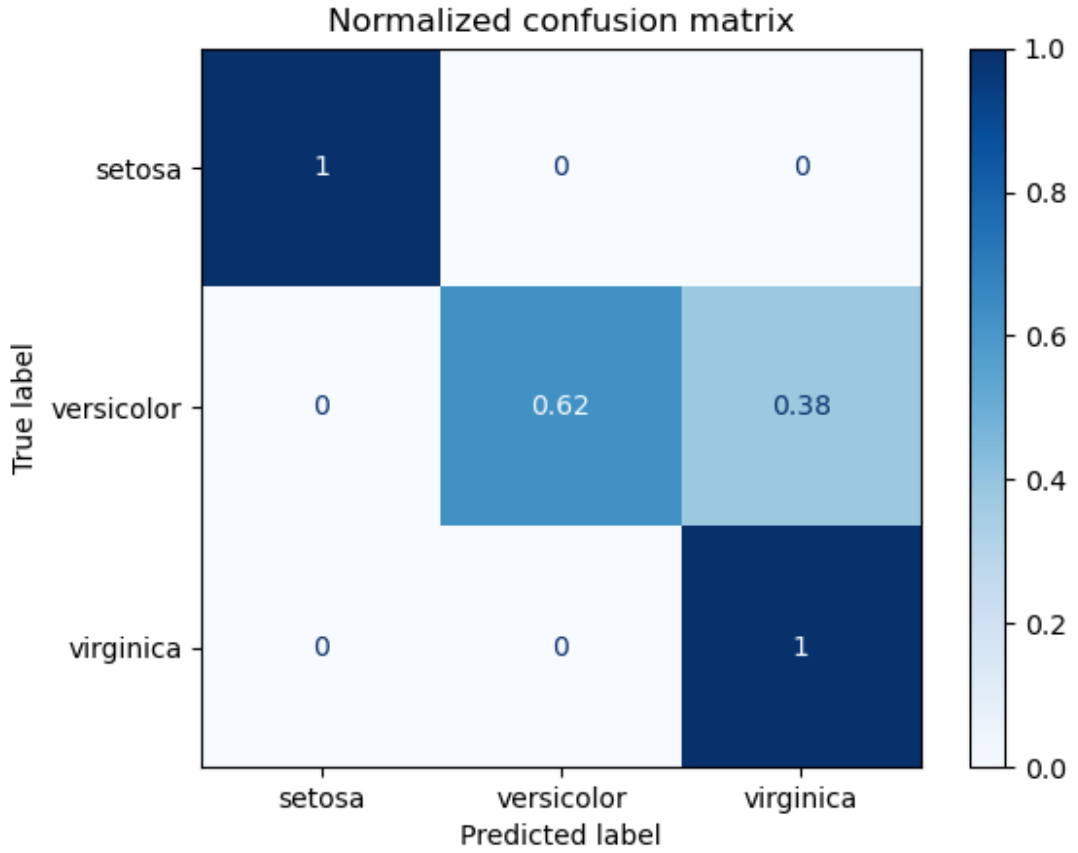


The Problem with Performance Metrics

- There is no single metric to determine the best model
- Therefore, don't put all your trust in 1 metric!
  - We will show you a few you can use
  - None of which give a true sense of model performance
- Also, many classification metrics are specialized to 2-class problems
- Multi-class metrics are more ad-hoc

Confusion Matrix

- Good for any number of classes
- See how well predictions = ground truth
- Often difficult to interpret: 3 labels = 9 values, 10 labels = 100 values.
- Can we say whether (overall) one model performs better?



Precision, Recall, and Accuracy

How many selected items are relevant?

Precision =

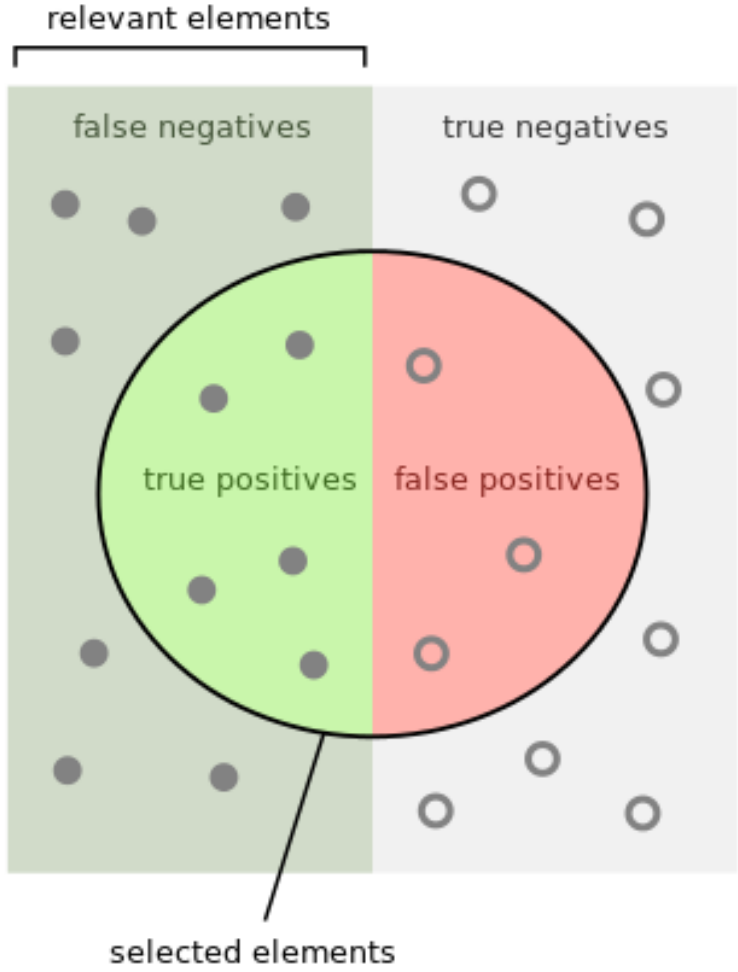
How many relevant items are selected?

Recall =

How many items were correctly classified?

Accuracy =

F1-score = 2 x x / +

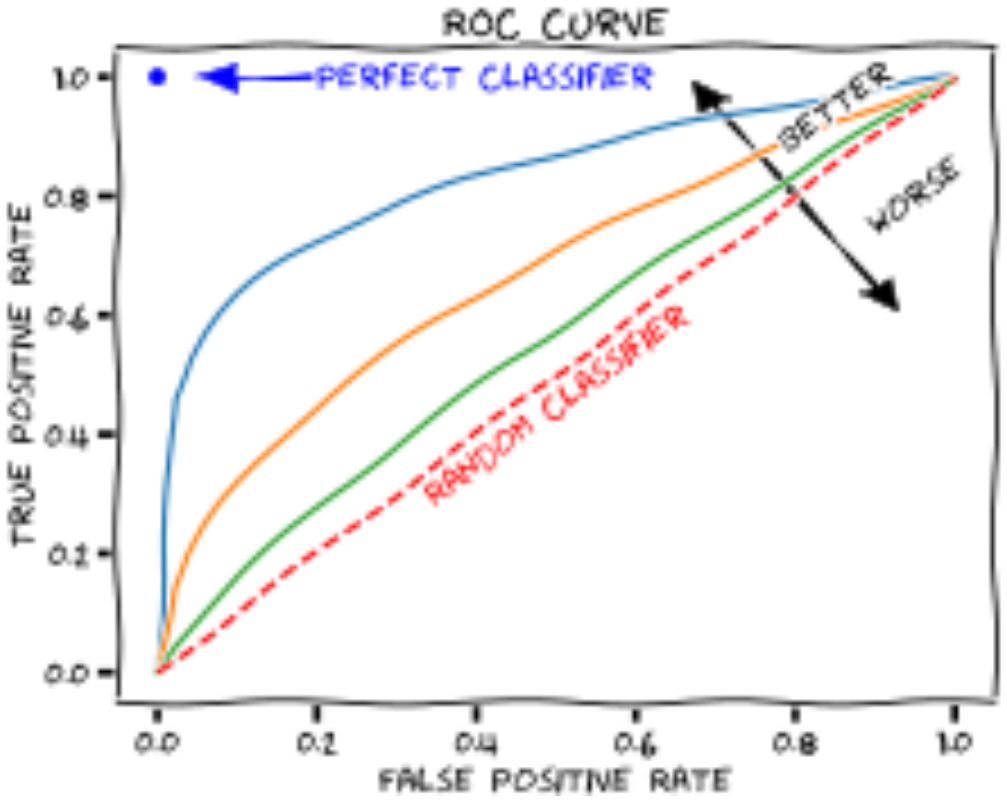


Problems

- Suppose there are 90 items with class 1, and 10 with class 0
- Naively pick class 1, what is the accuracy you're correct?
  - Answer: 90/100 = **0.9!**
- What will F1 be?
  - 2x1x0.9 / (1+0.9) = **0.95!**
- Need to compare against null model

Receiver Operating Characteristic (ROC)

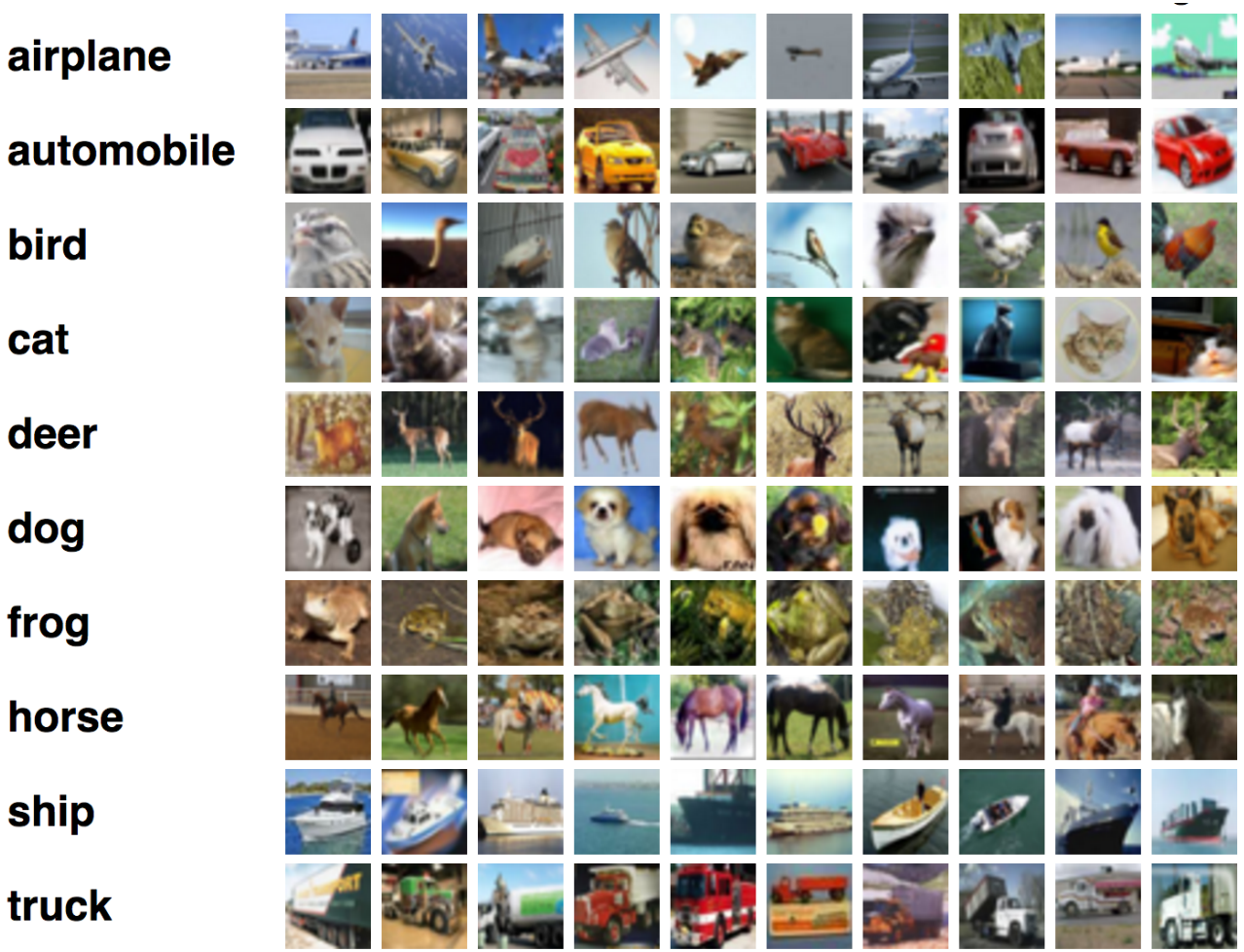
- Assume your model gives a “confidence” for a given class
  - E.g., Logistic regression probabilities: 0.9 = 90% chance state is 1
  - NOTE: the ROC is independent of and monotonic transform
  - Therefore as long as a model’s output value is monotonically related to its confidence, the ROC and AUC will be the same
- ROC: the probability class is correctly identified (true positive rate) vs incorrected identified (false positive rate)
- Area under this curve = ROC-AUC
- AUC = 0.5: no better than chance
- AUC = 1: perfect predictor



APPLICATIONS WITH JUPYTER NOTEBOOKS

Multi-Class Extensions

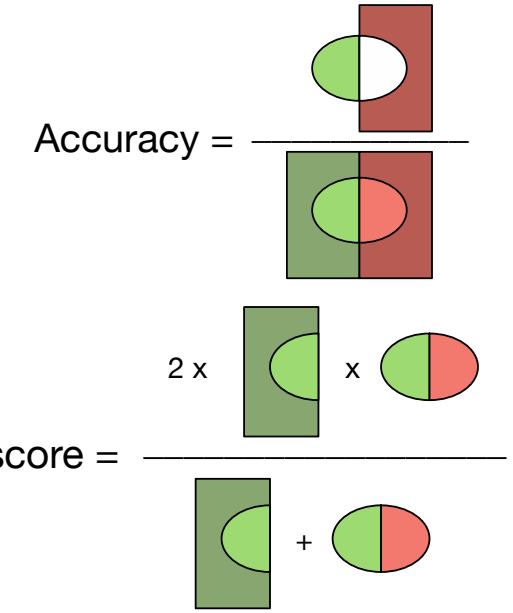
- What if we want to predict multiple classes?
- Usually boil down to these metrics
- Ex: top @ 1 or top @ 5
  - Is true class one of the top 1 or 5 predicted?
- Mean AUC: predict class C_i vs not.
  - Make AUC for each class
  - Take mean, weighed by frequency of each class

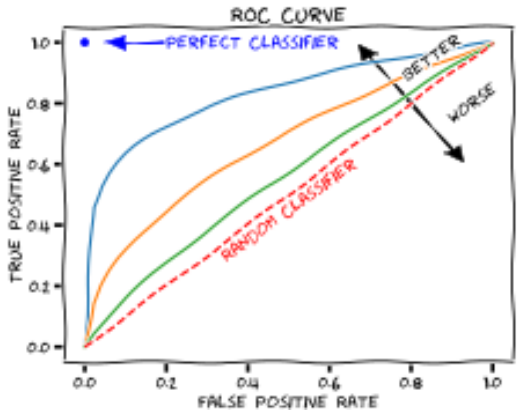


https://medium.com/@tifa2up/image-classification-using-deep-neural-networks-a-beginner-friendly-approach-using-tensorflow-94b0a090ccd4

Finding Good Models

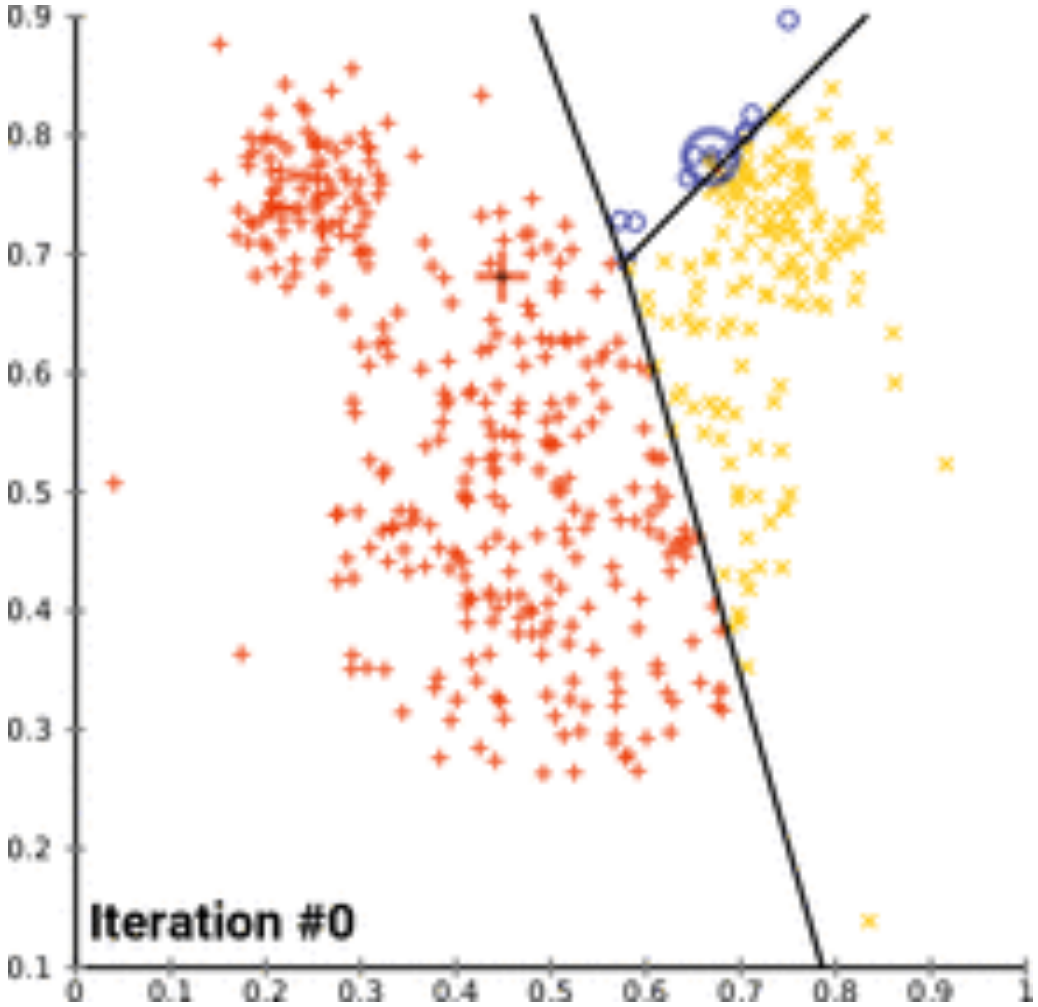
- Need to measure model performance
  - Confusion matrix
  - Accuracy, Precision, Recall
  - F1 score
  - AUC
- They are not equivalent!
- What metric should you use?
  - When model outputs class: F1/Accuracy
  - When model outputs value to represent class confidence: AUC
  - Multiple classes: top @ K, or weighted AUC





Next week: Unsupervised ML

- Example: Create K classifications with K-Means
  - Ex: 10 in the case of MNIST
- Lloyd's Algorithm (K-Means):
  - Start with K pre-assigned cluster means
  - Assign each datapoint to nearest cluster mean
  - Recalculate centroids
- Note: starting points can affect final results



Wikipedia illustration