Lecture 12: Classification

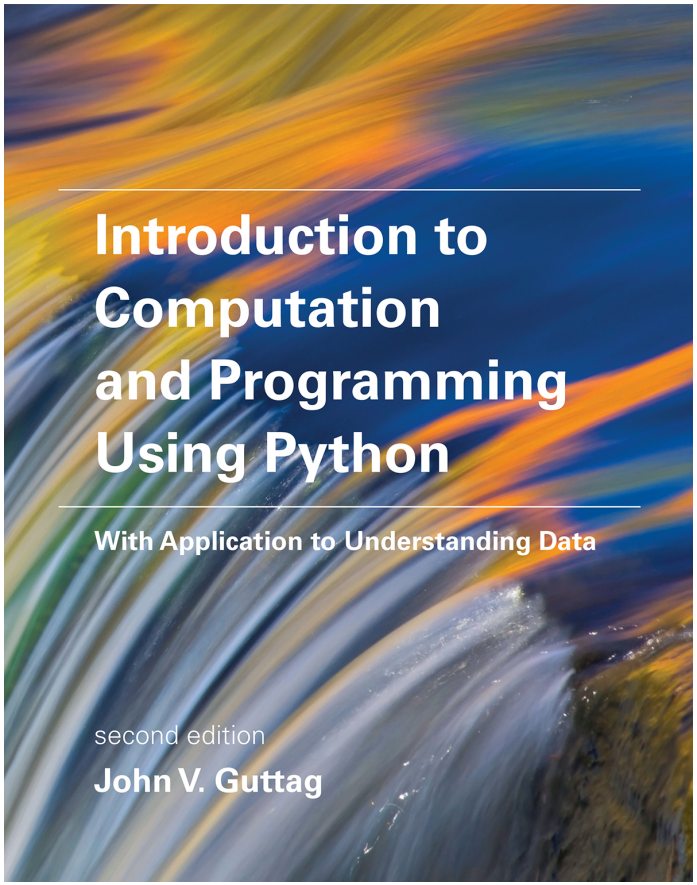
(download slides and .py files from Stellar to follow along)

John Guttag

MIT Department of Electrical Engineering and Computer Science

Relevant Reading

- Chapter 24



Quiz -> Problem Set

- See announcement posted this morning
- Questions?

Supervised Learning

- Regression
  - Predict a real number associated with a feature vector

Already spent time on this, right?

- *Classification*
  - Predict a discrete value (label) associated with a feature vector

Today's Lecture

- Metrics for evaluating a classification model
- Constructing training, validation, and test sets
- Two common classification methods
  - K-nearest neighbors
  - Logistic regression

Evaluation: Accuracy

- Accuracy is the fraction predicted correctly
- Certainly important, but is it enough?
- Consider a disease that occurs in 1% of population
  - Predicting disease-free has an accuracy of 0.99

A Fuller Picture of Performance

		predicted condition	
	total population	prediction positive	prediction negative
true condition	condition positive	**True Positive (TP)**	**False Negative (FN)** (type II error)
	condition negative	**False Positive (FP)** (Type I error)	**True Negative (TN)**

Confusion Matrix

Sensitivity and Specificity

$$sensitivity = \frac{true\ positive}{true\ positive + false\ negative}$$

Ability to know what is present

$$specificity = \frac{true\ negative}{true\ negative + false\ positive}$$

Ability to know what is not present

Rapid Strep Test

100 kids, 22 with strep throat
Diagnosed with strep: 14 with, 2 without
Diagnosed strep-free: 8 with, 76 without

$$sensitivity = \frac{14}{14+8} \approx .64$$

$$specificity = \frac{76}{76+2} \approx .97$$

sensitivity = recall
specificity = precision

Predictive Value

If the test is *positive*, how likely is it that he really ***has*** the disease? How *worried* should he be?

If the test is *negative*, how likely is it that he really ***does NOT*** have it? How *reassured* should he be?

http://sphweb.bumc.bu.edu/otlt/MPH-Modules/QuantCore/PH717_Screening/PH717_Screening5.html

$$Positive\ Predictive\ Value = \frac{true\ positives}{true\ positives + false\ positives}$$

$$Negative\ Predictive\ Value = \frac{true\ negatives}{true\ negatives + false\ negatives}$$

Rapid Strep Test

100 kids, 22 with strep throat

Diagnosed with strep: 14 with, 2 without

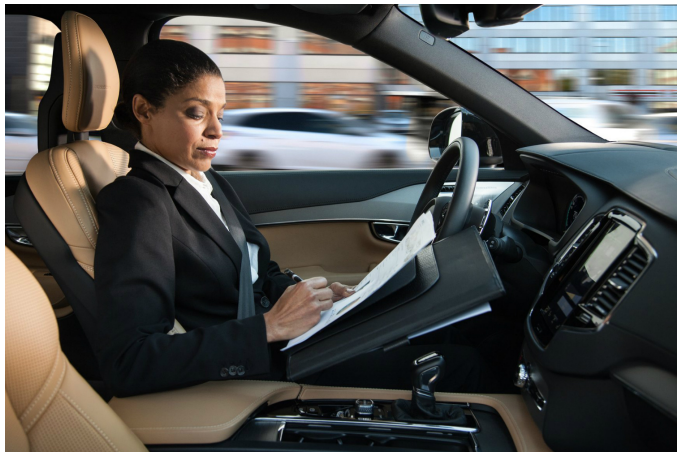
Diagnosed strep-free: 8 with, 76 without

$$PPV = \frac{14}{16} \approx .875$$

$$NPV = \frac{76}{84} \approx .905$$

Which Metric is Most Important?

- It depends



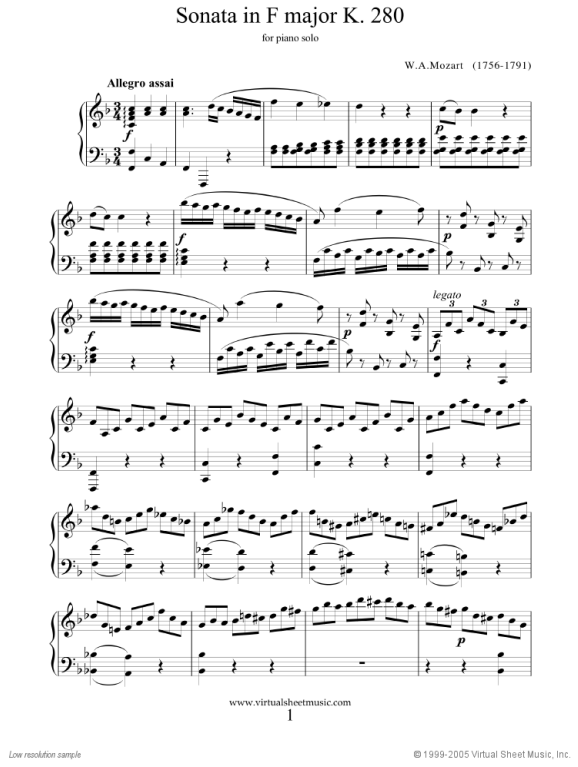
F-score

- In case of binary classification, F-score (aka F1-score) provides another way to summarize results:
  - Combines precision and recall (or specificity and sensitivity) into one value
  - Harmonic mean of precision and recall

$$F_1 = 2 \cdot \frac{1}{\frac{1}{recall} + \frac{1}{precision}}$$

$$= 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

- Ranges from 0 (worst) to 1 (best)

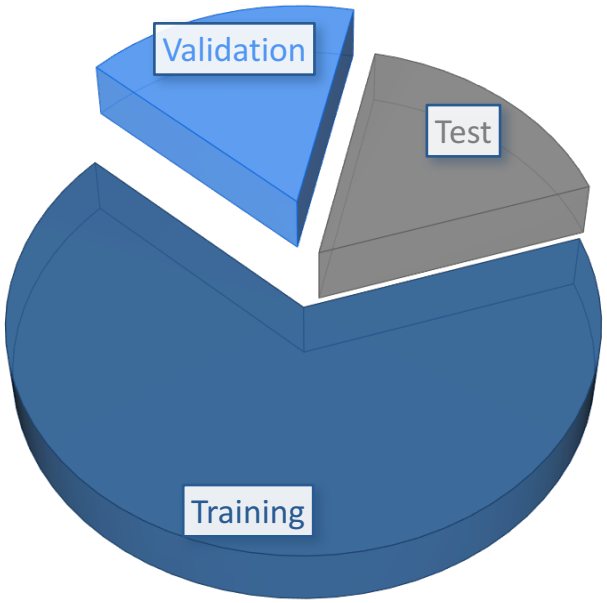


Using the Right Metrics Not Sufficient

- Have to apply them in the right way

Training/Testing Methodology

- Split dataset into groups, ideally
  - Training set: used to train the model
  - Validation set: used during training to select good parameter parameter values for a specific model
  - Test set: used to estimate the performance of the model

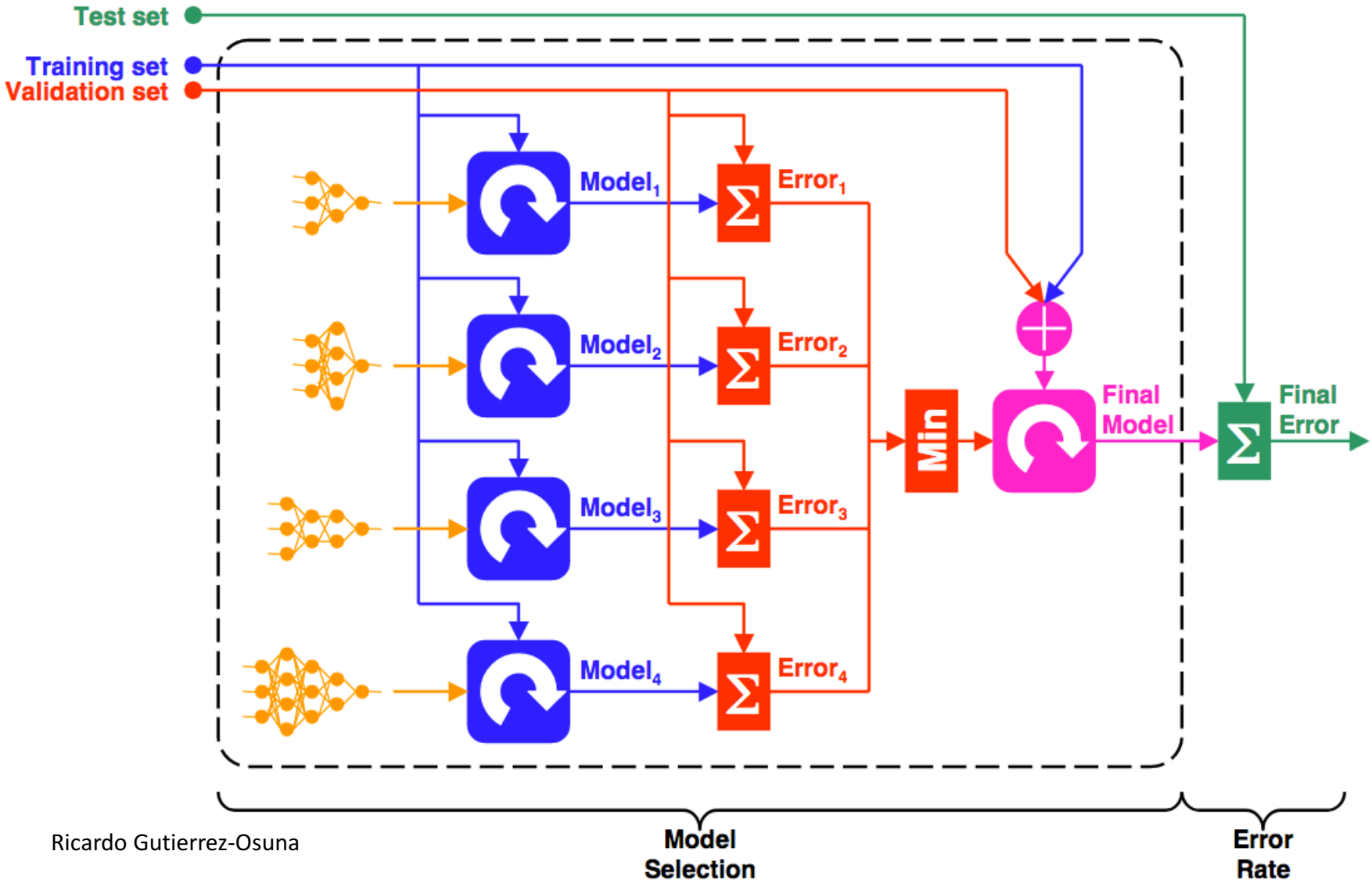


Why a Validation Set (or Sets)

- Almost all modeling techniques have free parameters, e.g.,
  - K for k-means
  - Degree for linear regression
  - Regularization for logistic regression
- Split training data into multiple training and validation sets
- Search for "optimal" parameter values

Warning: can lead to misleading estimates of error rates if there is not also a holdout set for testing

The Process



Ricardo Gutierrez-Osuna

How We Split Data Matters

- If we have a limited amount of data, may not have the luxury of holding out enough for meaningful evaluation
- If we do the split once, we could get lucky or unlucky about which examples end up where (unless dataset large enough for law of large numbers to kick in)
- Often resort to cross validation
  - Repeated random sampling
  - K-fold
  - Leave-one-out

Repeated Random Sampling

- Split the data into training and testing k times
  - Training set is a fixed number (say 0.8N) of examples that are randomly selected (without replacement)
  - Test set is the unselected examples
- For each split, i
  - Train a separate classifier (perhaps splitting the training data into training and validation sets)
  - Evaluate the model on the left-out examples to get a testing error, E_i
- Estimate error as $E = \frac{1}{k}\sum_1^k E_i$

Average error rate

K-fold Cross Validation

- Train k different models
  - Use N/k examples for testing
  - Train on the remaining N – N/k examples
- Again, estimate error as $E = \frac{1}{k}\sum_1^k E_i$
- Guarantees that all examples are used for both training and testing

Leave-on-out Cross Validation

- Degenerate case of k-fold where k = N
- Perform N experiments where we train on N-1 examples and test on the one held out example
- Estimate error the same way: $E = \frac{1}{k}\sum_1^k E_i$

Onto Classification Algorithms

- Nearest neighbor-based classification
- Training data:
  - Set of labeled feature vectors
- Learning algorithm:
  - Remember the training data
- Classification method:
  - Give new example a label based on proximity to examples in training data

An Example (similar to earlier lecture)

Features **Label**

Name	Egg-laying	Scales	Poisonous	Cold-blooded	Has legs	Reptile
Cobra	1	1	1	1	0	1
Rattlesnake	1	1	1	1	0	1
Boa constrictor	0	1	0	1	0	1
Chicken	1	1	0	1	1	0
Dart frog	1	0	1	0	1	0

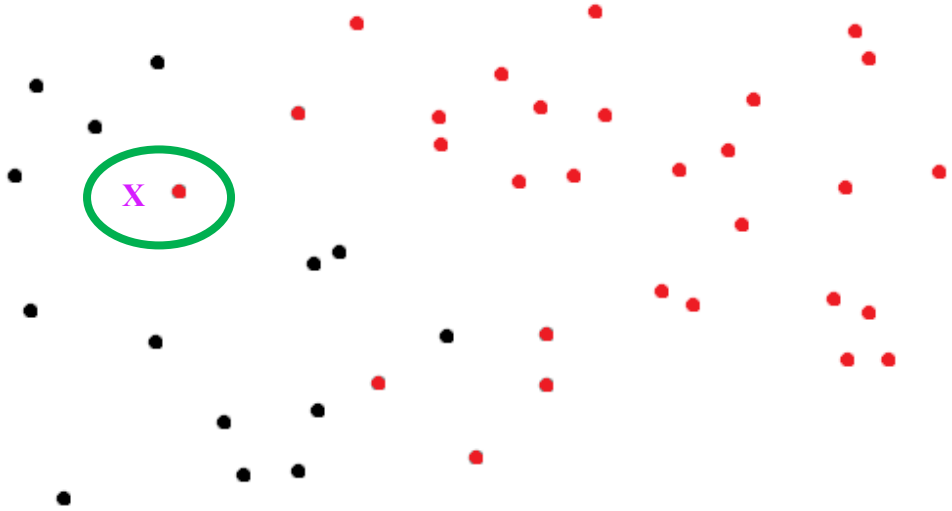
An Example (similar to earlier lecture)

Features **Label**

Name	Egg-laying	Scales	Poisonous	Cold-blooded	Has legs	Reptile
Cobra	1	1	1	1	0	1
Rattlesnake	1	1	1	1	0	1
Boa constrictor	0	1	0	1	0	1
Chicken	1	1	0	1	1	0
Dart frog	1	0	1	0	1	0
Zebra	0	0	0	0	1	?
Alligator	1	1	0	1	1	?

Nearest-neighbor Classification

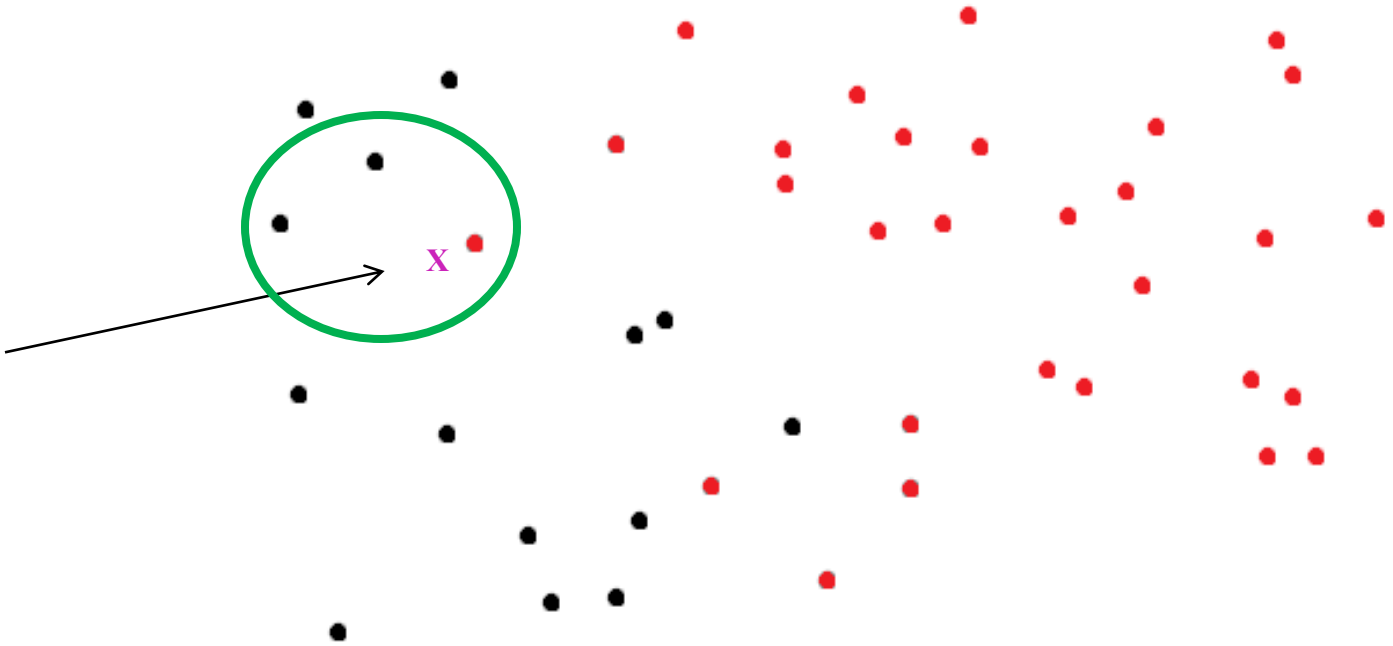
- When predicting the label of a new example
  - Find the nearest example in the training data
  - Predict the label associated with that example



Classify Test Animals

	Cobra R	Rattler R	Boa R	Chicken ~R	Dart Frog ~R
Zebra	2.236	2.236	1.732	2.0	1.414
Alligator	1.414	1.414	1.414	1.0	1,732

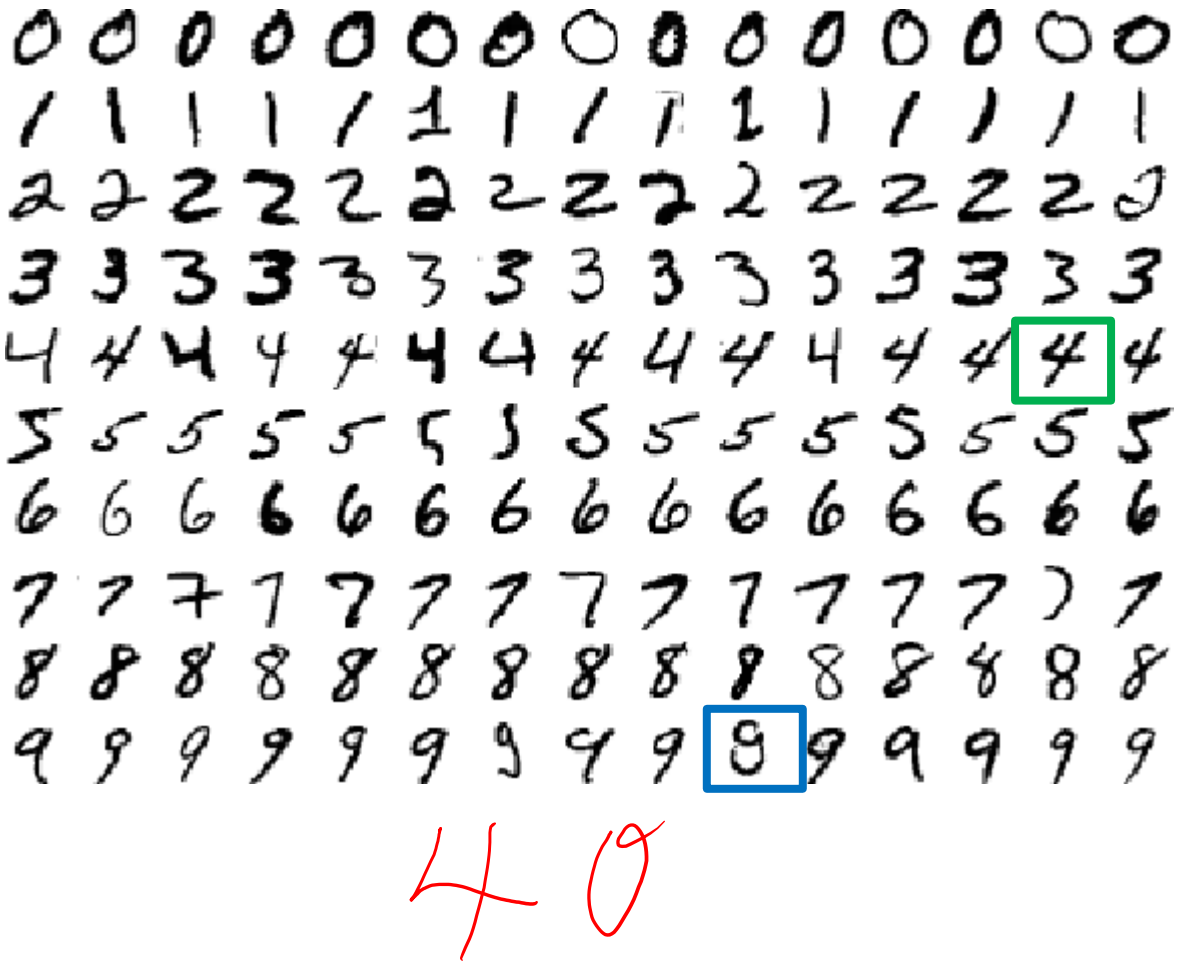
K-nearest Neighbors



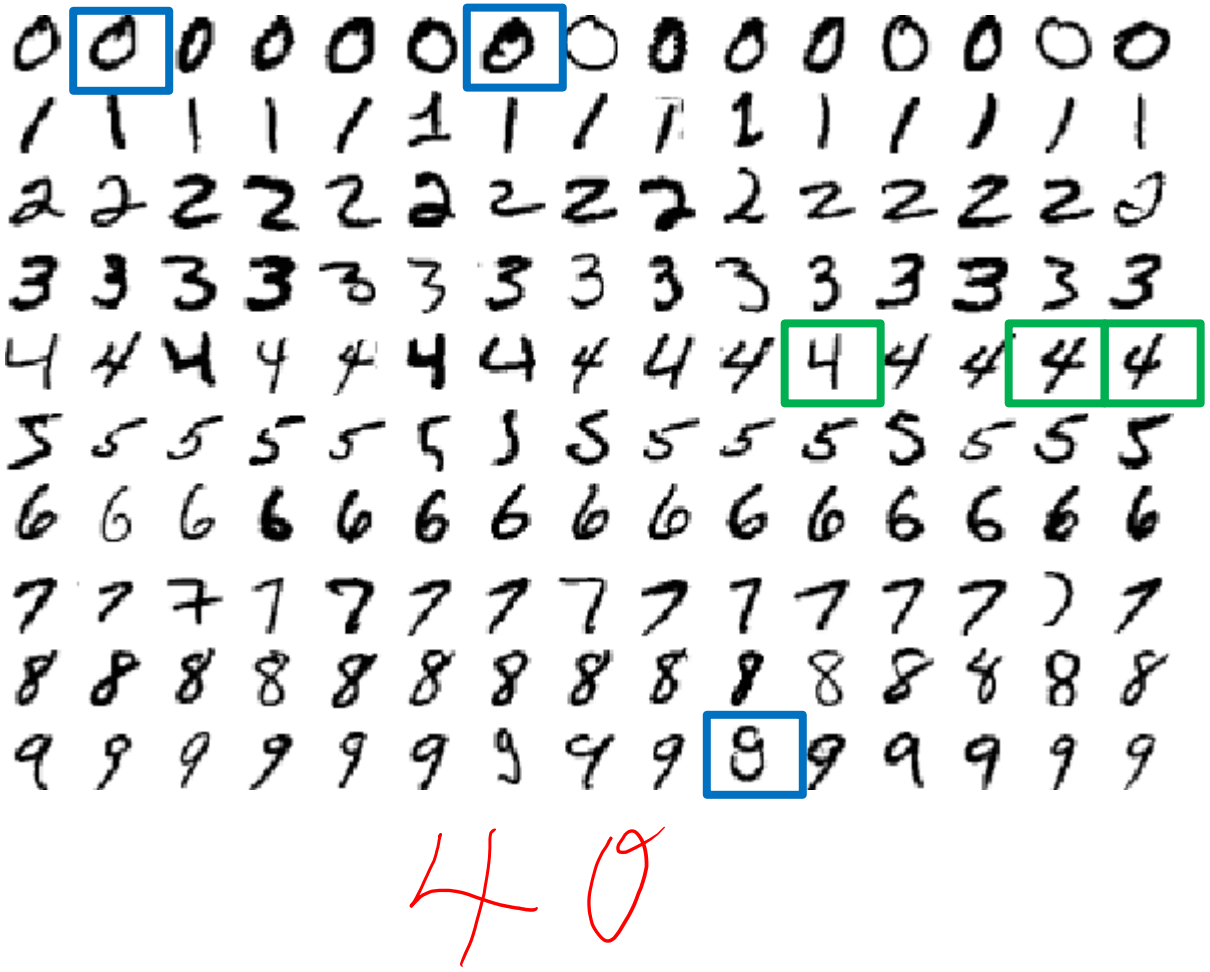
Classify Test Animals

	Cobra R	Rattler R	Boa R	Chicken ~R	Dart Frog ~R
Zebra	2.236	2.236	1.732	2.0	1.414
Alligator	1.414	1.414	1.414	1.0	1,732

Another Example



An Example

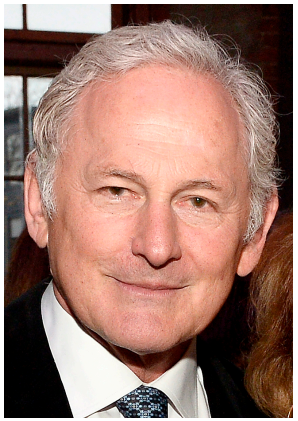


The Titanic Disaster

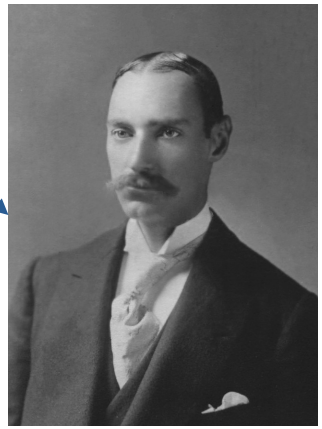
- RMS Titanic sank in the North Atlantic the morning of 15 April 1912, after colliding with an iceberg. Of the 1,300 passengers aboard, 812 died. (703 of 918 crew members died.)
- Database of 1046 passengers
  - Cabin class
    - 1st, 2nd, 3rd
  - Age
  - Gender
  - Outcome
    - (59% of passengers died)
  - Name

Sample of Data

1,0.92,M,1,Allison, Master. Hudson Trevor
1,2.0,F,0,Allison, Miss. Helen Loraine
1,30.0,M,0,Allison, Mr. Hudson Joshua Creighton
1,25.0,F,0,Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
1,39.0,M,0,Andrews, Mr. Thomas Jr
1,47.0,M,0,Astor, Col. John Jacob
1,18.0,F,1,Astor, Mrs. John Jacob (Madeleine Talmadge Force)
1,24.0,F,1,Aubart, Mme. Leontine Pauline
1,26.0,F,1,Barber, Miss. Ellen ""Nellie"”
1,80.0,M,1,Barkworth, Mr. Algernon Henry Wilson
1,24.0,M,0,Baxter, Mr. Quigg Edmond
2,33.0,F,1,West, Mrs. Edwy Arthur (Ada Mary Worth)
2,66.0,M,0,Wheadon, Mr. Edward H
2,31.0,M,1,Wilhelms, Mr. Charles
2,26.0,F,1,Wright, Miss. Marion
2,24.0,F,0,Yrois, Miss. Henriette (""Mrs Harbeck"")
3,42.0,M,0,Abbing, Mr. Anthony
3,13.0,M,0,Abbott, Master. Eugene Joseph
3,16.0,M,0,Abbott, Mr. Rossmore



Victor Garber as Thomas Andrews



John Jacob Astor IV

Class Passenger

```
class Passenger(object):
    featureNames = ('C1', 'C2', 'C3', 'age', 'male gender')
    def __init__(self, pClass, age, gender, survived, name):
        self.name = name
        self.featureVec = [0, 0, 0, age, gender]
        self.featureVec[pClass - 1] = 1
        self.label = survived
        self.cabinClass = pClass
    def distance(self, other):
        return minkowskiDist(self.featureVec, other.featureVec, 2)
```

Why treat cabin class as three binary variables?

Let's Try KNN

```
def KNearestClassify(training, testSet, label, k):
    """Assumes training & testSet lists of examples, k an int
       Predicts whether each example in testSet has label
       Returns number of true positives, false positives,
          true negatives, and false negatives"""


for k in (1,2,3):
    knn = lambda training, testSet:\
                KNearestClassify(training, testSet,
                                 'Survived', k)
    numSplits = 10
    print('Average of', numSplits,
          '80/20 splits using KNN (k=' + str(k) + ')')
    truePos, falsePos, trueNeg, falseNeg =\
          randomSplits(examples, knn, numSplits)

    print('Average of LOO testing using KNN (k=' + str(k) + ')')
    truePos, falsePos, trueNeg, falseNeg =\
          leaveOneOut(examples, knn)
```

Observations About Results

```
Finished processing 1046 passengers
with toScale = False

Average of LOO testing using KNN (k=1)
 Accuracy = 0.73
 Sensitivity = 0.681
 Specificity = 0.764
 Pos. Pred. Val. = 0.666
Average of LOO testing using KNN (k=2)
 Accuracy = 0.765
 Sensitivity = 0.534
 Specificity = 0.924
 Pos. Pred. Val. = 0.829
Average of LOO testing using KNN (k=3)
 Accuracy = 0.769
 Sensitivity = 0.663
 Specificity = 0.842
 Pos. Pred. Val. = 0.743
```

It took a while to run

Why that baseline?

Accuracy considerably better than 59%

k matters

Let's Try It With Scaling

```
Finished processing 1046 passengers
with toScale = False

Average of LOO testing using KNN (k=1)
 Accuracy = 0.73
 Sensitivity = 0.681
 Specificity = 0.764
 Pos. Pred. Val. = 0.666
Average of LOO testing using KNN (k=2)
 Accuracy = 0.765
 Sensitivity = 0.534
 Specificity = 0.924
 Pos. Pred. Val. = 0.829
Average of LOO testing using KNN (k=3)
 Accuracy = 0.769
 Sensitivity = 0.663
 Specificity = 0.842
 Pos. Pred. Val. = 0.743
```

```
Finished processing 1046 passengers
with toScale = True

Average of LOO testing using KNN (k=1)
 Accuracy = 0.731
 Sensitivity = 0.691
 Specificity = 0.759
 Pos. Pred. Val. = 0.664
Average of LOO testing using KNN (k=2)
 Accuracy = 0.772
 Sensitivity = 0.555
 Specificity = 0.922
 Pos. Pred. Val. = 0.832
Average of LOO testing using KNN (k=3)
 Accuracy = 0.794
 Sensitivity = 0.707
 Specificity = 0.855
 Pos. Pred. Val. = 0.77
```

Scaling matters

Advantages and Disadvantages of KNN

- Advantages
  - Learning fast, no explicit training
  - No theory required
  - Easy to explain method and results
- Disadvantages
  - Memory intensive and predictions can take a long time
    - What is complexity of brute force algorithm?
  - No model to shed light on process that generated data

Logistic Regression

- Analogous to linear regression
- Designed explicitly for predicting probability of an event
  - Dependent variable (label) can only take on a finite set of values. (Often 0 or 1)
- Learns a function that approximates P(Y|X)
- Assumes that P(Y|X) can be approximated by summing a series of terms of the form $w_i x_i$

More Formally

- Model fit (given feature values x_i) is:

$$P(Y|X) = \sum_{i=0}^{n-1} w_i * x_i$$

- Goal is to find weights w_i so that model best fits labels of examples:
  - Maximize the likelihood of observing the labels for the examples, given the weights on the features
  - Positive implies variable positively correlated with outcome
  - Negative implies variable negatively correlated with outcome
  - Absolute magnitude related to strength of the correlation

Linear versus Logistic Regression

- **Desired result is different**
  - In linear regression, outcome (dependent variable) is continuous. More appropriate when output can realistically assume any one of an infinite number of possible values.
  - In logistic regression, outcome (dependent variable) has only a limited number of possible values. More appropriate when trying to assign the output to a category, e.g., true/false; A/B/C; one of a small number of labels. We want *probability* of that outcome.
- **Error minimization technique**
  - Linear regression uses ordinary *least squares* to minimize the errors and find the best possible fit; logistic regression uses *maximum likelihood* to arrive at solution.

Maximum Likelihood Estimation (MLE)

- Estimates values for the parameters of a model
- Parameters chosen to maximize the likelihood that the process described by the model produced the data that were actually observed
- Typically approximated using gradient descent
- Good news is that there exists libraries for computing logistic regression

Logistic regression package

- Built-in class in `sklearn` package
- `LogisticRegression` is a class
- Calling `LogisticRegression()` creates an instance
- Calling `LogisticRegression().fit(SetOfFeatures, setOfLabels)` finds weights such that logistic function of linear combination of weights and features minimizes goodness of fit
- Result is an object containing coefficients (or weights) and function that will compute probability of label for any other feature vector

Class LogisticRegression

```
import sklearn.linear_model as lm
```

`fit`(sequence of feature vectors, sequence of labels)

Returns object of type LogisticRegression

`coef_`

Returns weights of features

`predict_proba`(feature vector)

Returns probabilities of labels

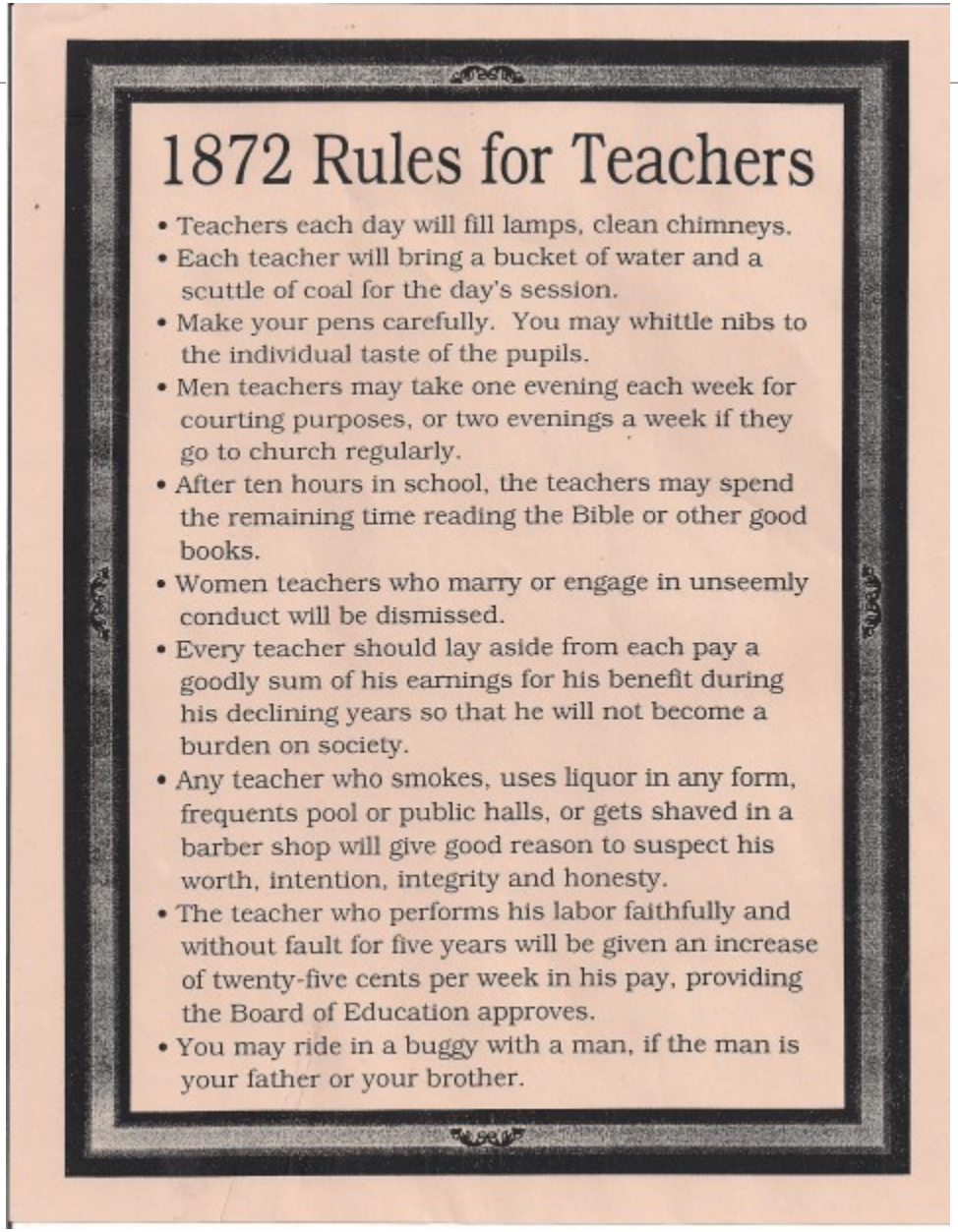
Building a Model

```
def buildModel(examples, toPrint = True):
    featureVecs, labels = [],[]
    for e in examples:
        featureVecs.append(e.getFeatures())
        labels.append(e.getLabel())
    LogisticRegression = lm.LogisticRegression
    model = LogisticRegression().fit(featureVecs, labels)
    if toPrint:
        print('model.classes_ =', model.classes_)
        for i in range(len(model.coef_)):
            print('For label', model.classes_[1])
            for j in range(len(model.coef_[0])):
                print('   ', Passenger.featureNames[j], '=',
                      model.coef_[0][j])
    return model
```

Building a Model

- `LogisticRegression` is a class
- Calling `LogisticRegression()` creates an instance
- Calling `LogisticRegression().fit(…,…)` finds weights such that logistic function of linear combination of weights and features minimizes goodness of fit
- `model` is object containing coefficients (or weights) and function that will label any other feature vector

Break



1872 Rules for Teachers

- Teachers each day will fill lamps, clean chimneys.
- Each teacher will bring a bucket of water and a scuttle of coal for the day's session.
- Make your pens carefully. You may whittle nibs to the individual taste of the pupils.
- Men teachers may take one evening each week for courting purposes, or two evenings a week if they go to church regularly.
- After ten hours in school, the teachers may spend the remaining time reading the Bible or other good books.
- Women teachers who marry or engage in unseemly conduct will be dismissed.
- Every teacher should lay aside from each pay a goodly sum of his earnings for his benefit during his declining years so that he will not become a burden on society.
- Any teacher who smokes, uses liquor in any form, frequents pool or public halls, or gets shaved in a barber shop will give good reason to suspect his worth, intention, integrity and honesty.
- The teacher who performs his labor faithfully and without fault for five years will be given an increase of twenty-five cents per week in his pay, providing the Board of Education approves.
- You may ride in a buggy with a man, if the man is your father or your brother.

Building a Model (recap)

- `LogisticRegression` is a class
- Calling `LogisticRegression()` creates an instance
- Calling `LogisticRegression().fit(…,…)` finds weights such that logistic function of linear combination of weights and features minimizes goodness of fit
- `model` is object containing coefficients (or weights) and function that will label any other feature vector

Applying Model

```
def applyModel(model, testSet, label, prob = 0.5):
    testFeatureVecs = [e.getFeatures() for e in testSet]
    probs = model.predict_proba(testFeatureVecs)
    truePos, falsePos, trueNeg, falseNeg = 0, 0, 0, 0
    for i in range(len(probs)):
        if probs[i][1] > prob:
            if testSet[i].getLabel() == label:
                truePos += 1
            else:
                falsePos += 1
        else:
            if testSet[i].getLabel() != label:
                trueNeg += 1
            else:
                falseNeg += 1
    return truePos, falsePos, trueNeg, falseNeg
```

Putting It Together

```
def lr(trainingData, testData, prob = 0.5):
    model = buildModel(trainingData, False)
    results = applyModel(model, testData, 'Survived', prob)
    return results

examples = buildTitanicExamples('TitanicPassengers.txt', True)

random.seed(0)
numSplits = 10
print('Average of', numSplits, '80/20 splits LR')
truePos, falsePos, trueNeg, falseNeg =\
      randomSplits(examples, lr, numSplits)

print('Average of LOO testing using LR')
truePos, falsePos, trueNeg, falseNeg =\
      leaveOneOut(examples, lr)
```

Results

```
Finished processing 1046 passengers
with toScale = True

Average of 10 80/20 splits LR
 Accuracy = 0.772
 Sensitivity = 0.697
 Specificity = 0.827
 Pos. Pred. Val. = 0.745
Average of LOO testing using LR
 Accuracy = 0.784
 Sensitivity = 0.703
 Specificity = 0.84
 Pos. Pred. Val. = 0.752
```

Why is LOO better?

Looking at Feature Weights

```
def buildModel(examples, toPrint = True):
    ...
    if toPrint:
        print('model.classes_ =', model.classes_)
        for i in range(len(model.coef_)):
            print('For label', model.classes_[1])
            for j in range(len(model.coef_[0])):
                print('    ', Passenger.featureNames[j], '=',
                      model.coef_[0][j])
```

Be wary of reading too much into the weights

Features are often correlated with each other

```
model.classes_ = ['Died', 'Survived']
For label Survived
   C1 =  1.63216724529
  C2 =  0.450445901237
  C3 =  -0.524767915253
    age  =  -0.0321807370825
    male  =  -2.29930576947
```

rich

young?

female

Changing the Cutoff

```
trainingSet, testSet = split80_20(examples)
model = buildModel(trainingSet, False)
for p in (0.1, 0.5, 0.9):
    print('Try p =', p)
    truePos, falsePos, trueNeg, falseNeg =\
                        applyModel(model, testSet,
                        'Survived', p)
    getStats(truePos, falsePos, trueNeg, falseNeg)
```

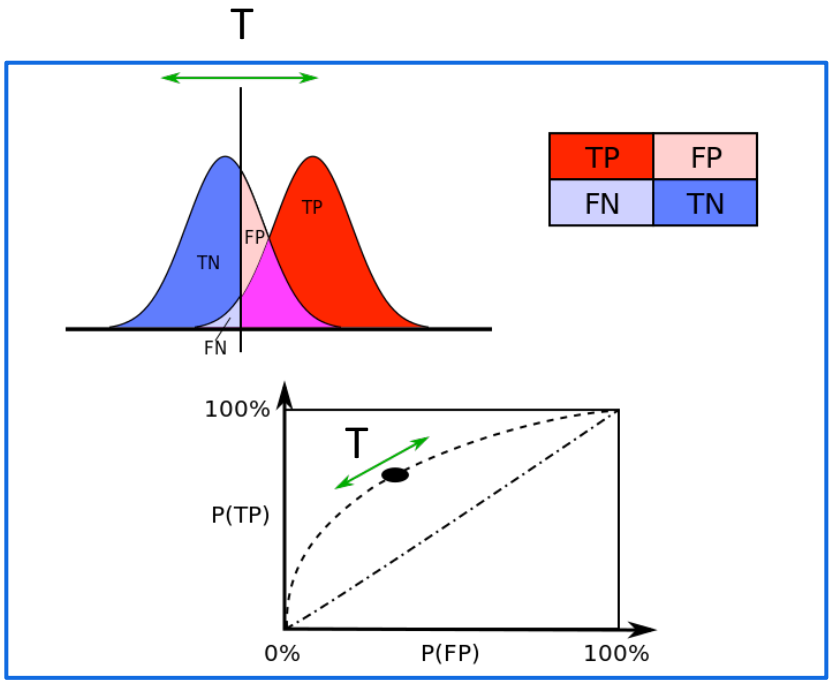
Try p = 0.1
Accuracy = 0.493
Sensitivity = 0.976
Specificity = 0.161
Pos. Pred. Val. = 0.444

Try p = 0.5
Accuracy = 0.818
Sensitivity = 0.8
Specificity = 0.831
Pos. Pred. Val. = 0.764

Try p = 0.9
Accuracy = 0.656
Sensitivity = 0.176
Specificity = 0.984
Pos. Pred. Val. = 0.882

ROC (Receiver Operating Characteristic)

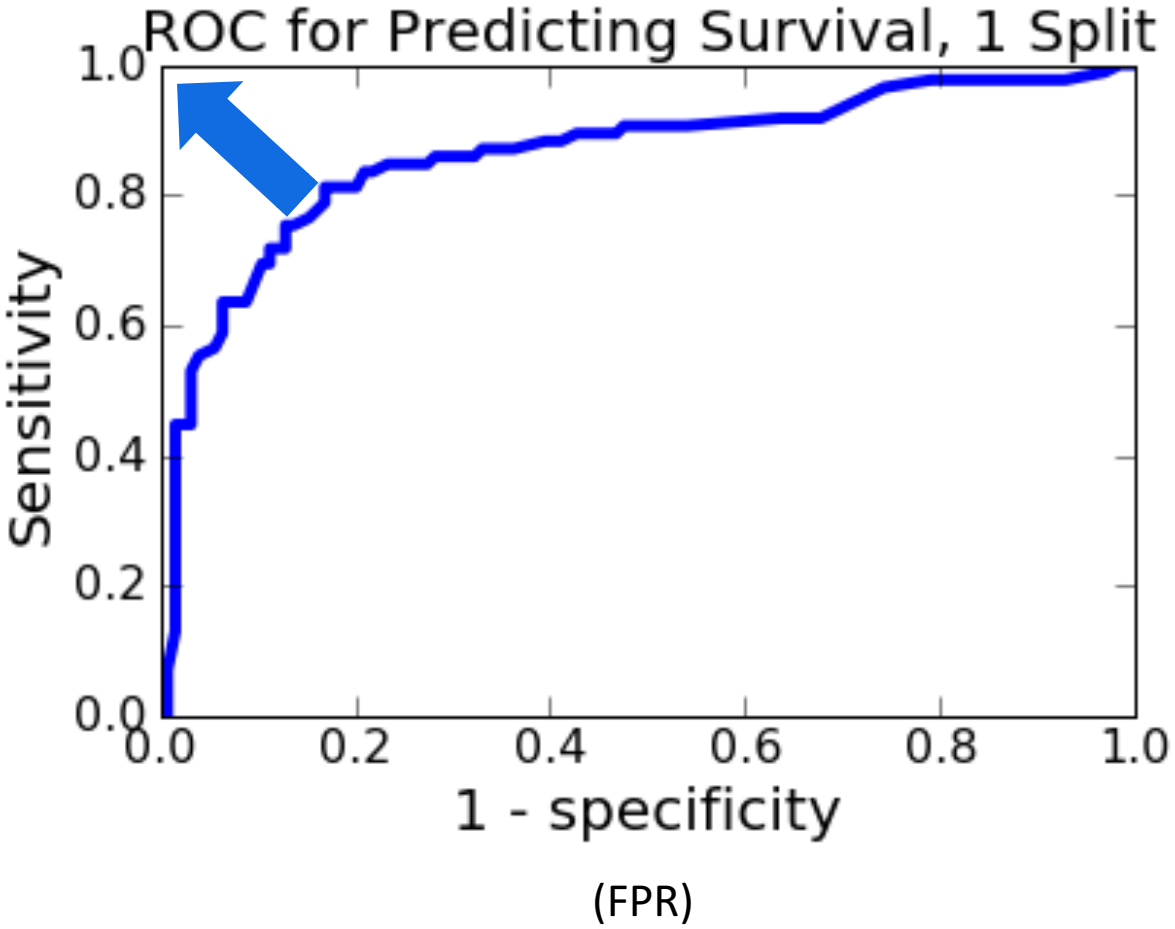
- Suppose we are assigning a binary label to an instance (e.g., using logistic regression to fit model to new examples)
- Imagine that probability of each label is described by a probability distribution
- Set a threshold T such that if probability is greater than T, assign associated label
- Plot probability of true positive versus probability of false positive
- As we vary T, change position along ROC curve
  - Increasing threshold results in fewer false positives (and more false negatives), corresponding to a leftward movement on the curve



ROC (Receiver Operating Characteristic)

```
def buildROC(trainingSet, testSet, title, plot = True):
    model = buildModel(trainingSet, True)
    xVals, yVals = [], []
    p = 0.0
    while p <= 1.0:
        truePos, falsePos, trueNeg, falseNeg =\
                                applyModel(model, testSet,
                                'Survived', p)
        xVals.append(1.0 - specificity(trueNeg, falsePos))
        yVals.append(sensitivity(truePos, falseNeg))
        p += 0.01
    ⋮
```

Output



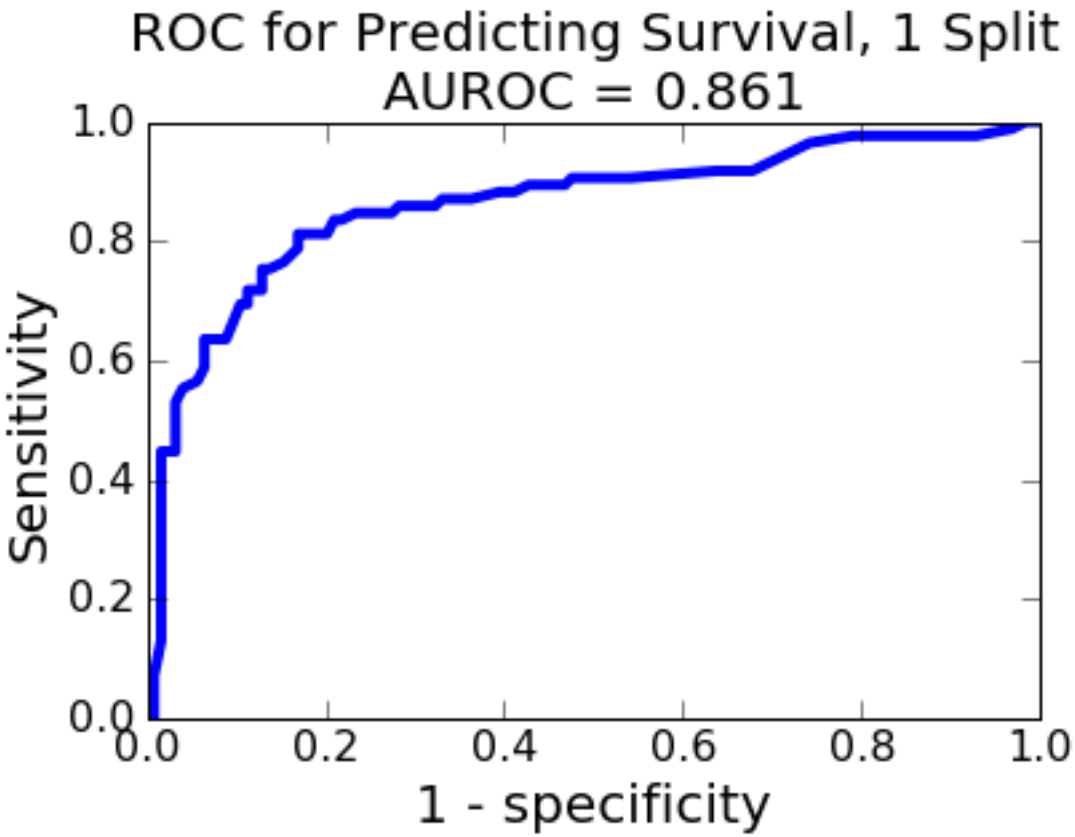
Ideal case

How might we summarize ROC?

Area under the curve: **AUROC**

AUROC

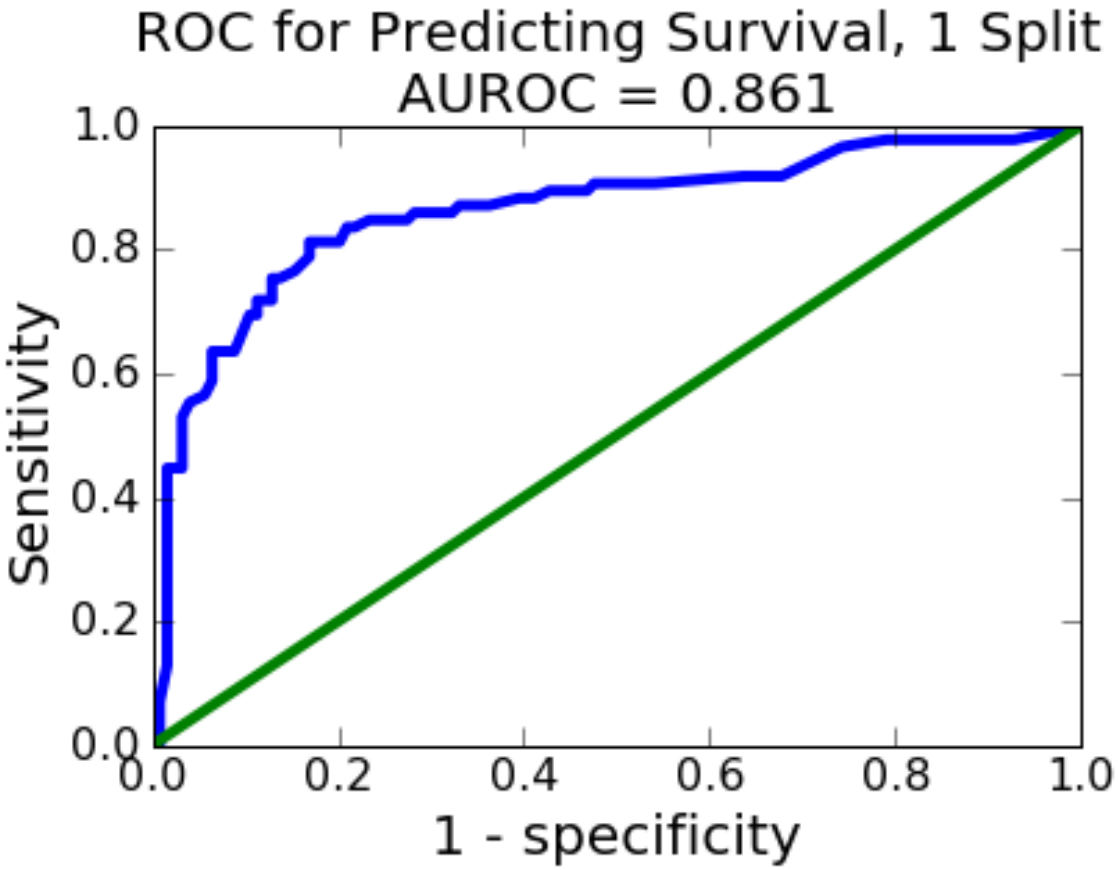
```
auroc = sklearn.metrics.auc(xVals, yVals, True)
```



How to interpret AUROC?

- Optimum would be AUROC = 1.0
- Here AUROC is 0.861

AUROC vs. Random Classifier

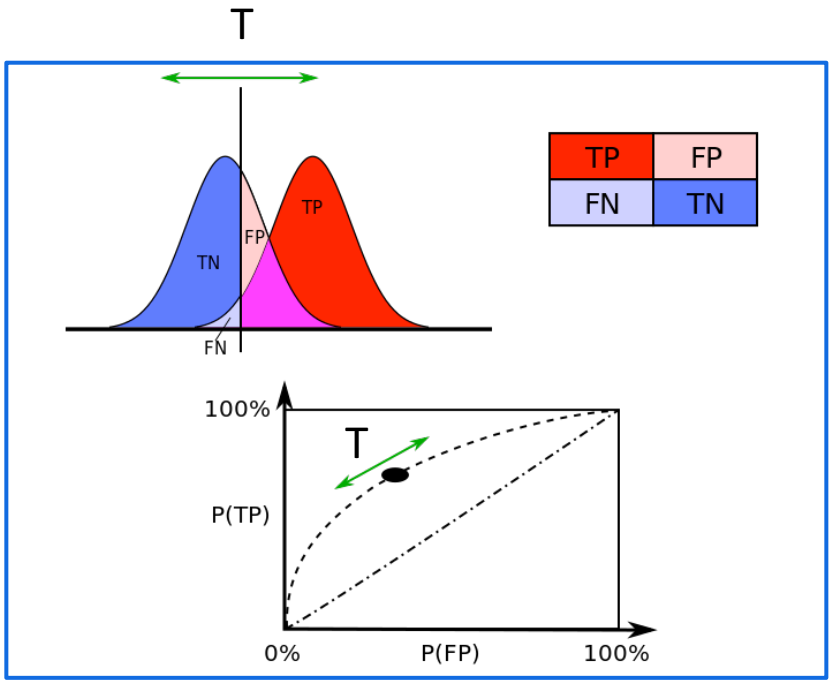


To what should we compare AUROC?

When TPR = FPR, have no discrimination – choosing at random

Adjusting threshold to meet objectives

- Adjusting threshold for assigning label moves us along the ROC curve
- Can trade off sensitivity for specificity



Wrapping Up ML

- It will play an increasingly important role in science, engineering, and society
- You should be comfortable with basic principles
  - Supervised vs. unsupervised
  - Regression vs. classification
  - Evaluation methods and metrics
- You should feel empowered to try it

Monday

- How to lie with statistics
- Course wrap up