

Logistic Regression

(download slides and .py files from class web site)

Ana Bell

MIT Department of Electrical Engineering and
Computer Science

The Titanic Disaster: An example

- 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 **Represent by:**
 - 1st, 2nd, 3rd • **binary (x3)**
 - Age • **float**
 - Gender • **binary**
 - Outcome
 - (59% died)

Do these features predict likelihood of survival?



Sample of Data

1,92,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

Let's Try It With Scaling

Finished processing 1046 passengers
with toScale = False

Average of L00 testing using KNN (k=1)

Accuracy = 0.73

Sensitivity = 0.681

Specificity = 0.764

Pos. Pred. Val. = 0.666

Average of L00 testing using KNN (k=2)

Accuracy = 0.765

Sensitivity = 0.534

Specificity = 0.924

Pos. Pred. Val. = 0.829

Average of L00 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 L00 testing using KNN (k=1)

Accuracy = 0.731

Sensitivity = 0.691

Specificity = 0.759

Pos. Pred. Val. = 0.664

Average of L00 testing using KNN (k=2)

Accuracy = 0.772

Sensitivity = 0.555

Specificity = 0.922

Pos. Pred. Val. = 0.832

Average of L00 testing using KNN (k=3)

Accuracy = 0.794

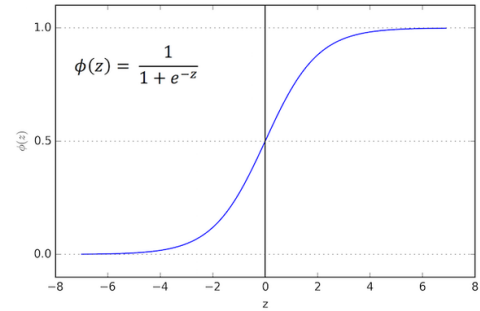
Sensitivity = 0.707

Specificity = 0.855

Pos. Pred. Val. = 0.77

Scaling matters

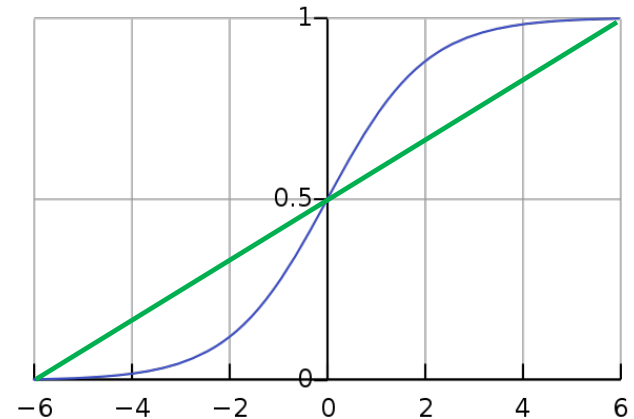
Logistic Regression



- Analogous to linear regression
- Designed explicitly for predicting **probability** of an event
 - Dependent variable (label) can only take on a finite set of values. Usually 0 or 1 (generalizations for more than 2).
- Finds **weights** for each feature
 - Positive implies feature positively correlated with outcome
 - Negative implies feature negatively correlated with outcome
 - Absolute magnitude related to strength of the correlation
- Optimization problem a bit complex, key is use of a log function—will simply give you intuition behind method

Logistic Regression

- Given a set of features associated with each example
- Examples belong to one of two classes (there are generalizations) – label is 0 or 1
- Logistic function is $\sigma(t) = \frac{1}{1+e^{-t}}$



- If t is a weighted sum of feature values

$$t = \sum_{i=0}^{n-1} w_i * f_i$$

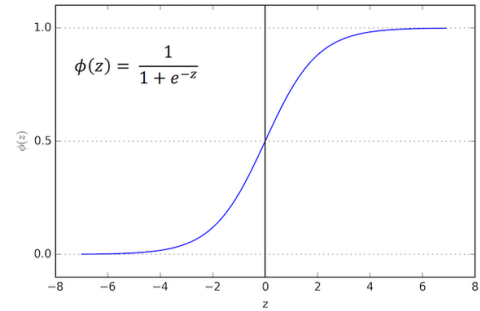
then probability of label $y = 0$ or 1 is

$$\sigma(t)^y (1 - \sigma(t))^{(1-y)}$$

Note that form of logistic function pushes probability smoothly towards 0 or 1

Compare to a pure linear relationship

Logistic Regression



- Model fit (given feature values f_i) is:

$$t = \sum_{i=0}^{n-1} w_i * f_i$$

- Goal is to find weights w_i so that model best fits labels of examples:
 - Minimize the R^2 goodness of fit, or equivalently, maximize the likelihood of observing the labels for the examples, given the weights on the features
- Algorithm is slightly more complicated than linear regression
 - But same kinds of methods – Newton's method or generalization of gradient descent – still apply
- 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
```

`fit(sequence of feature vectors, sequence of labels)`
Returns object of type LogisticRegression

`coef_`
Returns weights of features learned from fit

`predict_proba(sequence of feature vectors)`
Returns probabilities of labels for each feature vector

Building a Model

```
def buildModel(examples, toPrint = True):  
    featureVecs, labels = [], []  
    for e in examples:  
        featureVecs.append(e.getFeatures())  
        labels.append(e.getLabel())  
    LogisticRegression = sklearn.linear_model.LogisticRegression  
    model = LogisticRegression().fit(featureVecs, labels)  
    if toPrint:  
        ...|  
    return 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

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  
  
numSplits = 10  
print('Average of', numSplits, '80/20 splits LR')  
truePos, falsePos, trueNeg, falseNeg =\  
    divide80_20(examples, lr, numSplits)  
  
print('Average of L00 testing using LR')  
truePos, falsePos, trueNeg, falseNeg =\  
    leaveOneOut(examples, lr)
```

Results using Logistic Regression

Average of 10 80/20 splits LR

Accuracy = 0.771

Sensitivity = 0.697

Specificity = 0.824

Pos. Pred. Val. = 0.742

Average of L00 testing using LR

Accuracy = 0.786

Sensitivity = 0.705

Specificity = 0.842

Pos. Pred. Val. = 0.754

In this instance, looks like LOO is slightly better?

Compare to KNN Results

LOO results

using KNN (k=3)

Accuracy = 0.769

Sensitivity = 0.663

Specificity = 0.842

Pos. Pred. Val. = 0.743

LOO using LR

Accuracy = 0.786

Sensitivity = 0.705

Specificity = 0.842

Pos. Pred. Val. = 0.754

LR slightly better

Compare to KNN Results

Average of 10 80/20 splits
using KNN (k=3)

Accuracy = 0.766

Sensitivity = 0.67

Specificity = 0.836

Pos. Pred. Val. = 0.747

Average of 10 80/20 splits LR

Accuracy = 0.771

Sensitivity = 0.697

Specificity = 0.824

Pos. Pred. Val. = 0.742

LR still slightly better?

LR also provides insight about variables

And facilitates tradeoff between precision and recall

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; yes/no; one of a small number of labels.

■ 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.
- Linear regression is usually solved by minimizing the least squares error of the model to the data, thus large errors are penalized with a quadratic effect.
- Logistic regression reduces the impact of large errors, as large errors are penalized by an amount that is asymptotically a constant.

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_[i])  
            for j in range(len(model.coef_[0])):  
                print('    ', Passenger.featureNames[j], '=',  
                      model.coef_[0][j])  
    return model
```

```
buildModel(examples, True)
```

Be wary of reading too
much into the weights
Features are often
correlated with each other

model.classes_ = ['Died', 'Survived']

For label Survived

C1 = 1.4167156941 rich

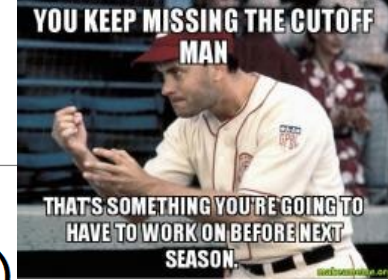
C2 = 0.2102635871

C3 = -0.777912594

age = -0.491363033 young?

male gender = -2.31666 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

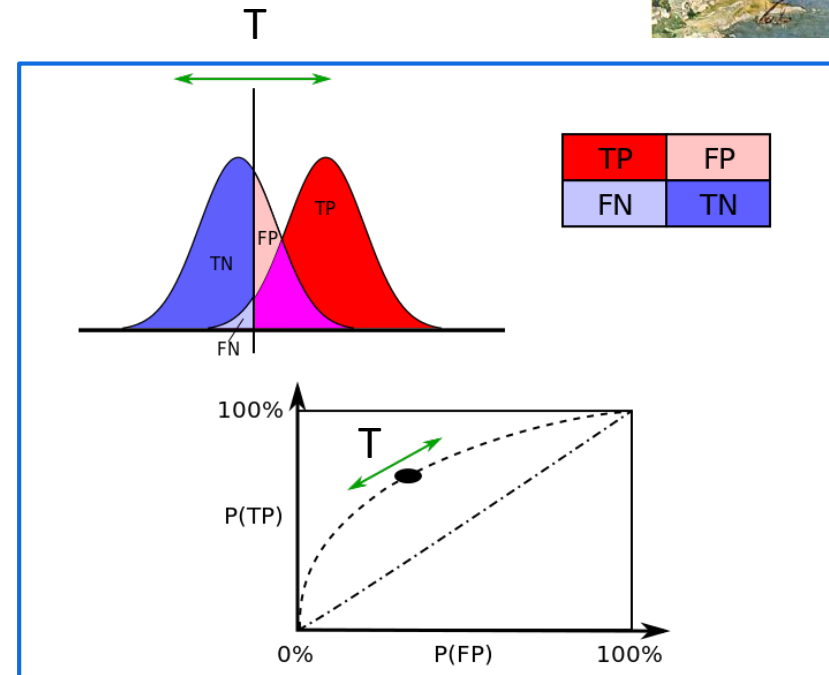
Specificity = 0.984

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
- Can set a threshold T such that if probability is greater than T , assign associated label
- Can 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



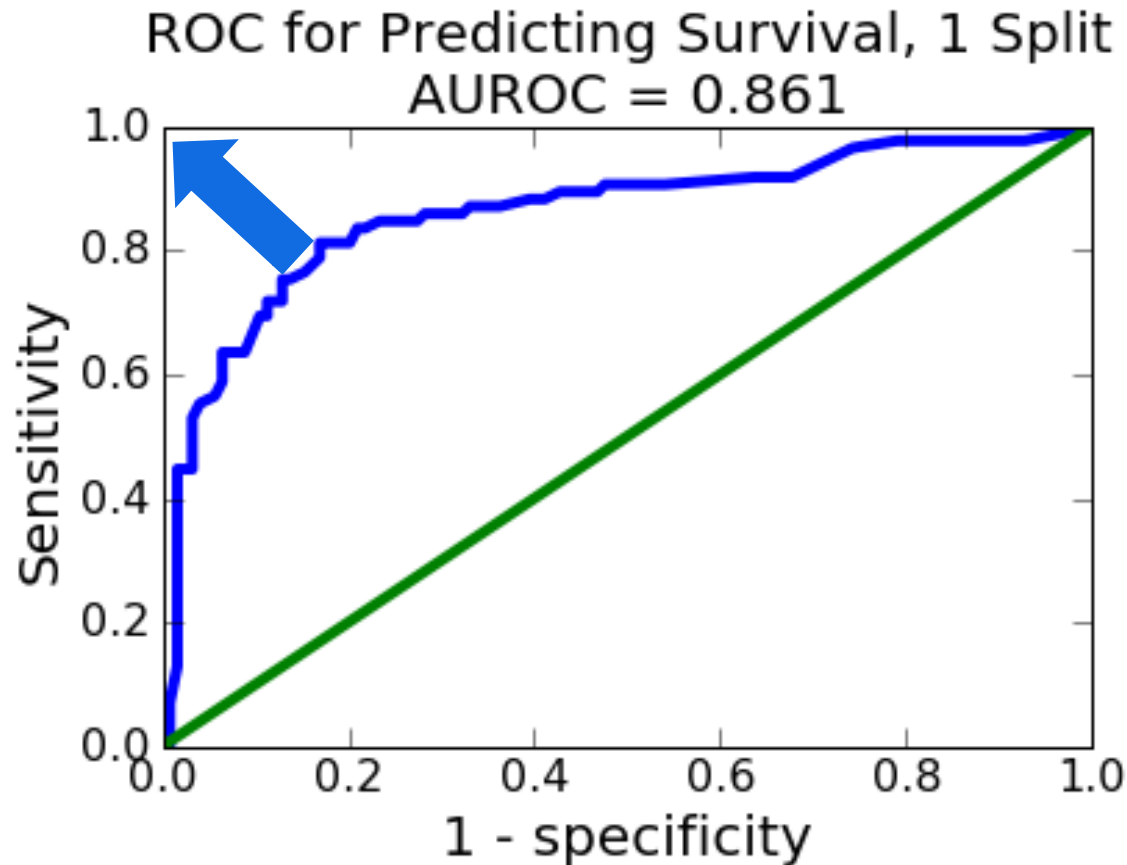
ROC (Receiver Operating Characteristic)



```
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  
  
    :
```

AUROC

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

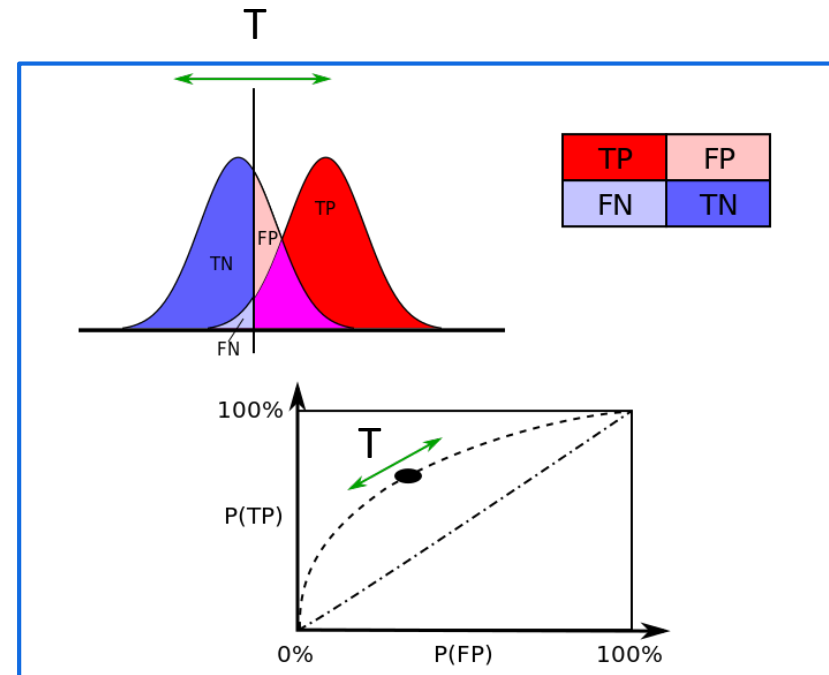


How to interpret AUROC?

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

Adjusting threshold to meet objectives

- Adjusting threshold for assigning label moves us along the ROC curve
- Can trade off sensitivity for specificity
- Balance point often provides convenient compromise, though in some circumstances may want higher sensitivity



Summary

- Introduced machine learning as way to fit a model to data
 - Account for observed effects
 - Explain underlying process
 - Predict results for new instances
- Clustering methods
 - Unsupervised way of grouping similar examples
 - k means
- Classification methods
 - Supervised way of learning model
 - k nearest neighbors
 - Logistic regression
- Measuring performance of methods
 - Specificity, selectivity, ROC, others

