LECTURE 6: LINEAR CLASSIFICATION AND LOGISTIC REGRESSION

Ehsan Aryafar
earyafar@pdx.edu
http://web.cecs.pdx.edu/~aryafare/ML.html

Recall: Model Selection via Sparsity

	MSZoning_FV	MSZoning_RH	MSZoning_RL	MSZoning_RM	Street_Pave	LotShape_IR2	LotShape_IR3	LotShape_Reg	LandContour_HLS
0	0	0	1	0	1	0	0	1	0
1	0	0	1	0	1	0	0	1	0
2	0	0	1	0	1	0	0	0	0
4	0	0	1	0	1	0	0	0	0
5	0	0	1	0	1	0	0	0	0

174 variables after one-hot coding

- Model selection problem: Need to identify the parameters that *really* matter
  - Help interpret results
  - Improves generalization error (less parameters)
- Idea: Fit model under sparsity constraint:
  - Linear model: $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$
  - Feature x_j is ignored if $\beta_j = 0$
  - Try to force most $\beta_j = 0 \Rightarrow$ Model only uses a few of the variables

Recall: Regularized LS Estimation

- Regularization: General method for finding constrained solutions
  - E.g. solutions that are sparse
- Standard least squares estimation (from Lecture 3):

$$\hat{\beta} = \arg\min_{\beta} MSE(\beta), \qquad MSE(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Regularized estimator:

$$\hat{\beta} = \arg\min_{\beta} J(\beta), \qquad J(\beta) = MSE(\beta) + \phi(\beta)$$

  - $MSE(\beta) =$ mean-squared prediction error from before
  - $\phi(\beta)$ = regularizing function.
- Concept: Regularizer penalizes β that are "unlikely"
  - Constrains estimate to smaller set of parameters

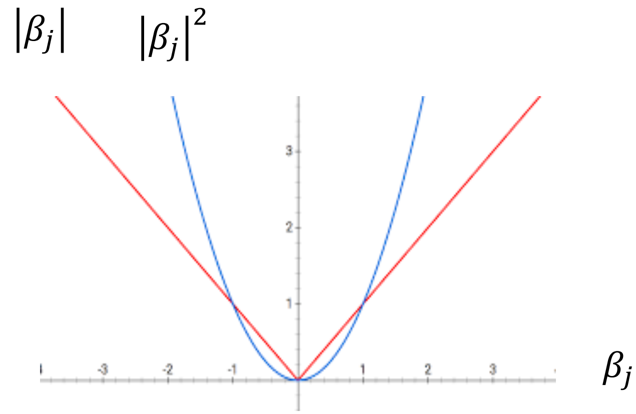
Recall: Two Common Regularizers

- Ridge regression (called L2)

$$\phi(\beta) = \frac{\alpha}{n}\sum_{j=1}^{d} |\beta_j|^2$$

- LASSO regression (called L1)

$$\phi(\beta) = 2\alpha \sum_{j=1}^{d} |\beta_j|$$



- Coefficient $\alpha > 0$ determines regularization level
  - Higher $\alpha \Rightarrow$ Higher level of regularization, more constrained
  - Will show how to select α later via cross-validation
  - Scaling factors adjust to match sklearn convention
- Both penalize large β_j: Tries to make β_j small
  - Will see that L1 also promotes sparsity
- Convention: Do not include intercept term β_0
  - In general, no reason to make this term small

Recall: Solving Ridge Regression

- Ridge regression problem: Find β to minimize

$$J(\boldsymbol{\beta}) = \|\boldsymbol{y} - A\boldsymbol{\beta}\|^2 + \alpha\|\boldsymbol{\beta}\|^2$$

- Solution for given regularization level

$$\boldsymbol{\beta}_{ridge} = (\boldsymbol{A}^T\boldsymbol{A} + \alpha\boldsymbol{I})^{-1}\boldsymbol{A}^T\boldsymbol{y}$$

  - Set gradient = 0
- Sklearn function for ridge regression:
  - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

Recall: Solving LASSO Regression

- LASSO cost function:

$$J(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha\sum_{j=1}^{d}\left|\beta_j\right| = \frac{1}{2n}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}\|^2 + \alpha\|\boldsymbol{\beta}\|_1$$

- Because derivative of $\left|\beta_j\right|$ is not continuous, there is no closed-form solution.
- Many methods to solve iteratively
  - Least angle regression (LAR), coordinate descent, ADMM
  - However, the cost function is convex ⇒ no local minima
  - Beyond the scope of this class
  - See textbook [Hastie2008] for LAR method

Recall: Summary

Method	Regularizer	Effect on parameters	Solution for Fitting
None	$\phi(\boldsymbol{\beta}) = 0$	Leaves parameters unconstrained	$\widehat{\boldsymbol{\beta}} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\ \boldsymbol{A}^T\boldsymbol{y}$
Ridge	$\phi(\boldsymbol{\beta}) = \frac{\alpha}{n}\lVert\boldsymbol{\beta}\rVert_2^2$	Makes parameters small Close to zero	$\widehat{\boldsymbol{\beta}} = (\boldsymbol{A}^T\boldsymbol{A} + \alpha\,\boldsymbol{I})^{-1}\ \boldsymbol{A}^T\boldsymbol{y}$
LASSO	$\phi(\boldsymbol{\beta}) = 2\alpha\lVert\boldsymbol{\beta}\rVert_1$	Makes parameters sparse. Many coefficients exactly zero	No analytic solution. Need to run an optimizer

- Regularized least squares

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\beta} J(\boldsymbol{\beta})\,, \qquad J(\boldsymbol{\beta}) = \frac{1}{n}\lVert\boldsymbol{y} - \boldsymbol{A}\boldsymbol{\beta}\rVert^{\,2} + \phi(\boldsymbol{\beta})$$

- Whatever you choose for the regularizer:
  - **Scale data before training**
  - **Select regularization level with cross-validation**

Learning Objectives

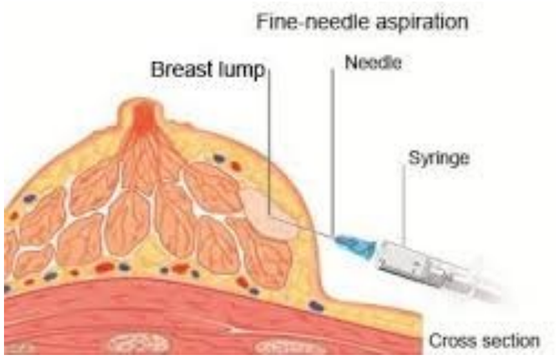
- Formulate a machine learning problem as a classification problem
  - Identify features, class variable, training data
- Visualize classification data using a scatter plot.
- Mathematically describe a linear classifier as an equation and on plot.
  - Determine visually if data is perfect linearly separable.
- Formulate a classification problem using logistic regression
  - Binary and multi-class
  - Describe the logistic and soft-max function
  - Logistic function to approximate the probability
- Derive the loss function for ML estimation of the weights in logistic regression
- Use sklearn packages to fit logistic regression models
- Measure the accuracy of classification
- Adjust threshold of classifiers for trading off types of classification errors. Draw a ROC curve.
- Perform LASSO regularization for feature selection

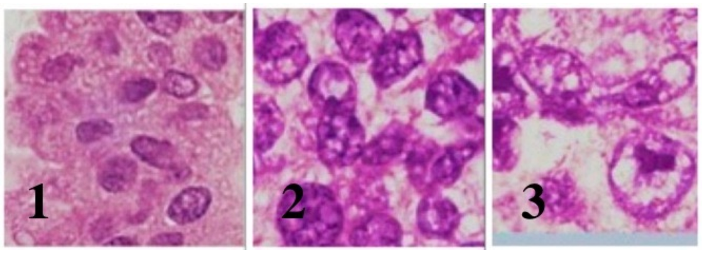
Outline

- Motivating Example: Classifying a breast cancer test
- Linear classifiers
- Logistic regression
- Fitting logistic regression models
- Returning to the breast cancer dataset
- Measuring accuracy in classification

Diagnosing Breast Cancer

- Fine needle aspiration of suspicious lumps
- Cytopathologist visually inspects cells
  - Sample is stained and viewed under microscope
- Determines if cells are benign (no cancer) or malignant (possibly cancer)
  - Also provides grading if malignant
- Uses many features (years of training):
  - Size and shape of cells, degree of mitosis, differentiation, …
- Diagnosis is not exact
- If uncertain, use a more comprehensive biopsy
  - Additional cost and time
  - Stress to patient as wait for results
- Can machine learning provide better rules?





Grades of carcinoma cells
http://breast-cancer.ca/5a-types/

Data

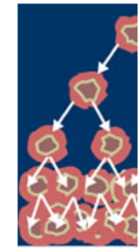
- Univ. Wisconsin study, 1994
- 569 samples
- 10 visual features for each sample
  - Not actual images, but doctor measurements
- Ground truth determined by more comprehensive biopsy

- First publication: O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.

Breast Cancer Wisconsin (Diagnostic) Data Set

Download: Data Folder, Data Set Description

Abstract: Diagnostic Wisconsin Breast Cancer Database



Data Set Characteristics:	Multivariate	**Number of Instances:**	569	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	32	**Date Donated**	1995-11-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	442524

Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

Demo

Breast Cancer Diagnosis via Logistic Regression

In this demo, we will see how to visualize training data for classification, plot the logistic function and perform logistic regression. As an example, we will use the widely-used breast cancer data set. This data set is described here:

https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin

Each sample is a collection of features that were manually recorded by a physician upon inspecting a sample of cells from fine needle aspiration. The goal is to detect if the cells are benign or malignant.

Loading and Visualizing the Data

We first load the packages as usual.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets, linear_model, preprocessing
%matplotlib inline
```

Next, we load the data. It is important to remove the missing values.

```
names = ['id','thick','size_unif','shape_unif','marg','cell_size','bare',
         'chrom','normal','mit','class']
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/' +
                 'breast-cancer-wisconsin/breast-cancer-wisconsin.data',
                names=names,na_values='?',header=None)
df = df.dropna()
df.head(6)
```

	id	thick	size_unif	shape_unif	marg	cell_size	bare	chrom	normal	mit	class

Loading The Data

- Follow standard pandas routine

```
names = ['id','thick','size_unif','shape_unif','marg','cell_size','bare',
        'chrom','normal','mit','class']
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/' +
                 'breast-cancer-wisconsin/breast-cancer-wisconsin.data',
                names=names,na_values='?',header=None)
df = df.dropna()
df.head(6)
```

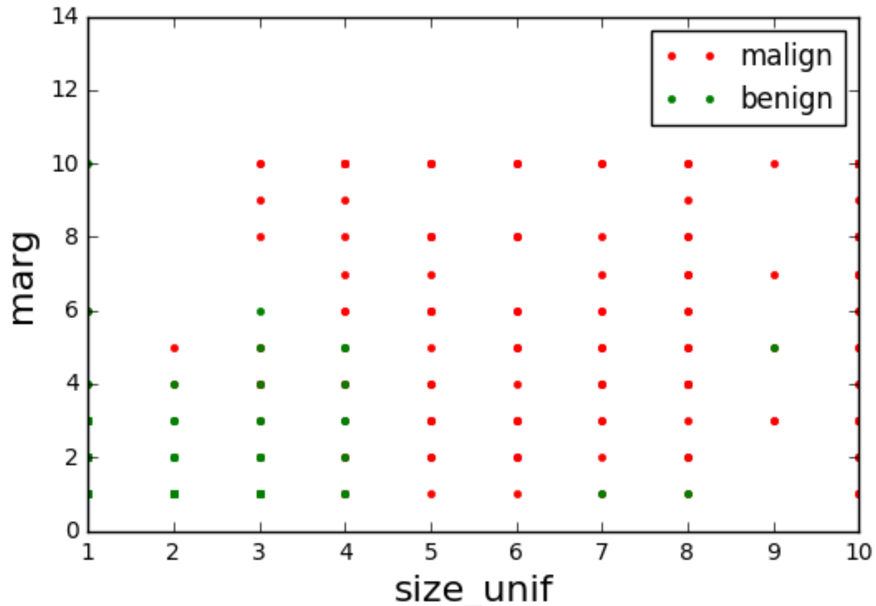
	id	thick	size_unif	shape_unif	marg	cell_size	bare	chrom	normal	mit	class
0	1000025	5	1	1	1	2	1.0	3	1	1	2
1	1002945	5	4	4	5	7	10.0	3	2	1	2
2	1015425	3	1	1	1	2	2.0	3	1	1	2
3	1016277	6	8	8	1	3	4.0	3	7	1	2
4	1017023	4	1	1	3	2	1.0	3	1	1	2
5	1017122	8	10	10	8	7	10.0	9	7	1	4

Drops missing samples

Class = 2 => benign
Class = 4 => malignant
The convention used by authors

See following for explanation of attributes
https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.names

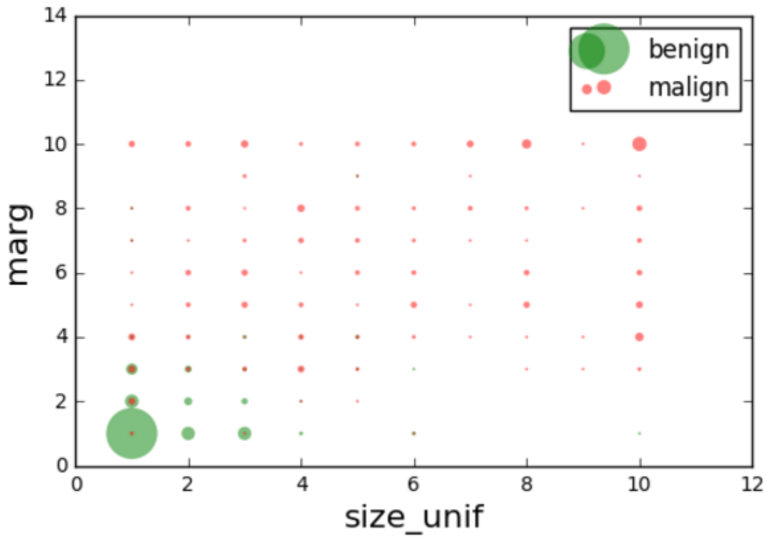
Visualizing the Data



- Scatter plot of points from each class
- Axis are two features
- Plot not informative
  - Many points overlap
  - Relative frequency at each point not visible

```
y = np.array(df['class'])
xnames =['size_unif','marg']
X = np.array(df[xnames])

Iben = np.where(y==2)[0]
Imal = np.where(y==4)[0]

plt.plot(X[Imal,0],X[Imal,1],'r.')
plt.plot(X[Iben,0],X[Iben,1],'g.')
plt.xlabel(xnames[0], fontsize=16)
plt.ylabel(xnames[1], fontsize=16)
plt.ylim(0,14)
plt.legend(['malign','benign'],loc='upper right')
```

Improving the Plot

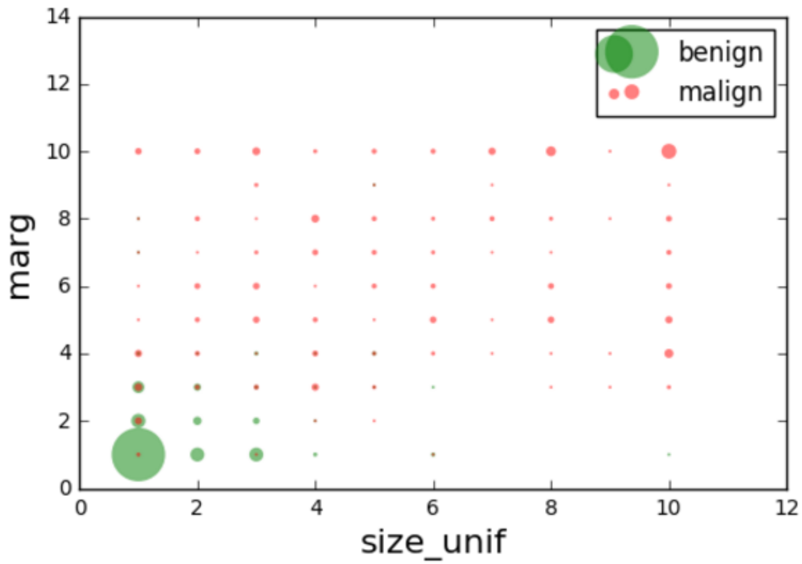


- Make circle size proportional to count
- Many gymnastics to make this plot in python

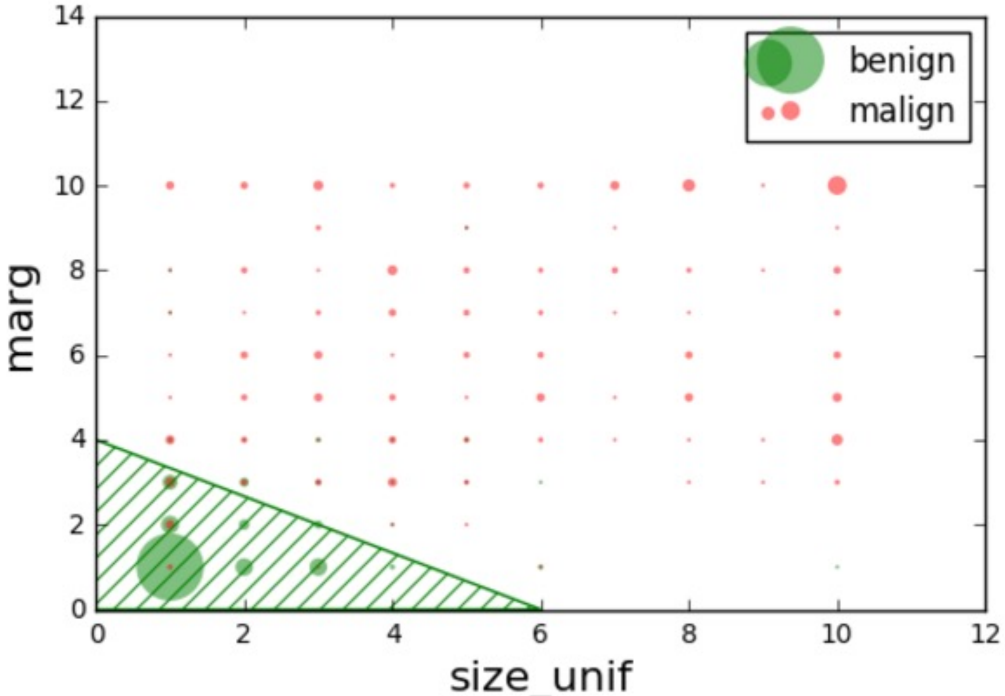
```
# Compute the bin edges for the 2d histogram
x0val = np.array(list(set(X[:,0]))).astype(float)
x1val = np.array(list(set(X[:,1]))).astype(float)
x0, x1 = np.meshgrid(x0val,x1val)
x0e= np.hstack((x0val,np.max(x0val)+1))
x1e= np.hstack((x1val,np.max(x1val)+1))

# Make a plot for each class
yval = [2,4]
color = ['g','r']
for i in range(len(yval)):
    I = np.where(y==yval[i])[0]
    cnt, x0e, x1e = np.histogram2d(X[I,0],X[I,1],[x0e,x1e])
    x0, x1 = np.meshgrid(x0val,x1val)
    plt.scatter(x0.ravel(), x1.ravel(), s=2*cnt.ravel(),alpha=0.5,
                c=color[i],edgecolors='none')
plt.ylim([0,14])
plt.legend(['benign','malign'], loc='upper right')
plt.xlabel(xnames[0], fontsize=16)
plt.ylabel(xnames[1], fontsize=16)
```

In-Class Exercise

- Determine a classification rule
  - Predict class label from the two features



A Possible Classification Rule



- From inspection, benign if:

$$\text{marg} + \frac{2}{3}(\text{size\_unif}) < 4$$

- Classification rule from linear constraint
- What are other possible classification rules?
- Every rule misclassifies some points
- What is optimal?

Mangasarian's Original Paper

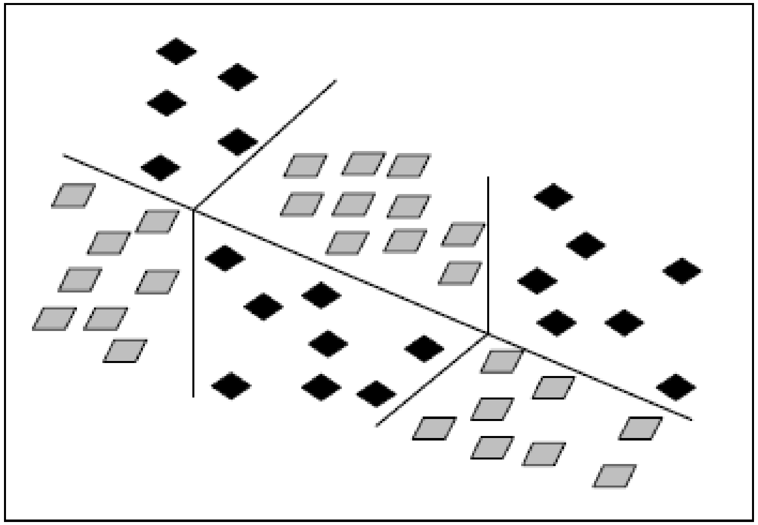


Figure 2.2 - Decision boundaries generated by MSM-T. Dark objects represent benign tumors while light object represent malignant ones.

- Proposes Multisurface method – Tree (MSM-T)
  - Decision tree based on linear rules in each step
- Fig to left from
  - Pantel, "Breast Cancer Diagnosis and Prognosis," 1995
- Best methods today use neural networks
- This lecture will look at linear classifiers
  - These are much simpler
  - Do not provide same level of accuracy
- But, building block to more complex classifiers

Outline

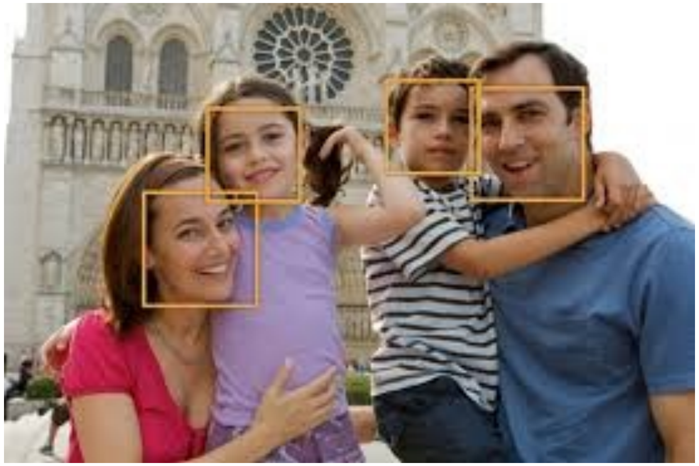
- Motivating Example: Classifying a breast cancer test
- Linear classifiers
- Logistic regression
- Fitting logistic regression models
- Returning to the breast cancer dataset
- Measuring accuracy in classification

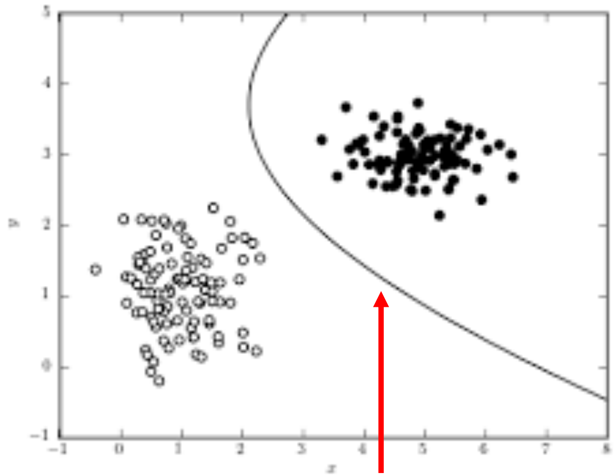
Classification

- Given features $\boldsymbol{x}$, determine its class label, $y = 1, \dots, K$
- Binary classification: $y = 0$ or 1
- Many applications:
  - Face detection: Is a face present or not?
  - Reading a digit: Is the digit 0,1,…,9?
  - Are the cells cancerous or not?
  - Is the email spam?
- Equivalently, determine classification function (learn function f(**x**)):

$$\hat{y} = f(\boldsymbol{x}) \in \{1, \dots, K\}$$

  - Like regression, but with a discrete response
  - May index $\{1, \dots, K\}$ or $\{0, \dots, K-1\}$





Not the function but boundary. Function would get x_1 and x_2 and map to 0 or 1.

Linear Classifier

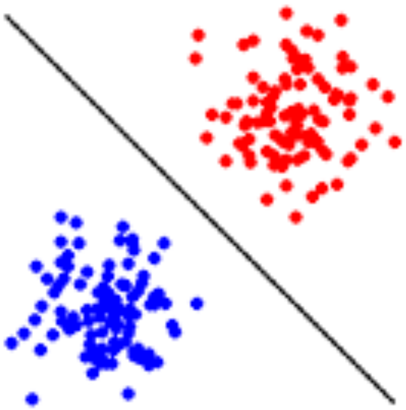
- General binary classification rule:

$$\hat{y} = f(x) = 0 \text{ or } 1$$

- Linear classification rule:
  - Take linear combination $z = w_0 + \sum_{j=1}^{d} w_d x_d$
  - Predict class from z (real valued number)

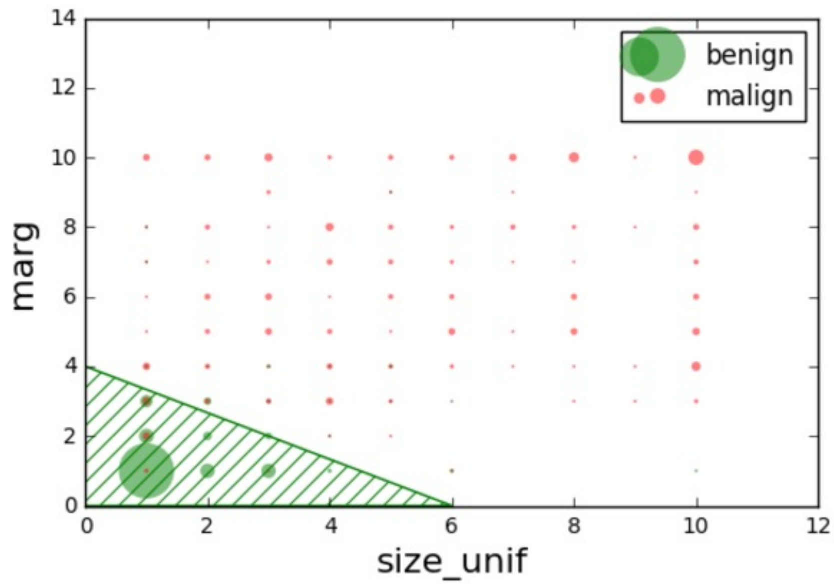
$$\hat{y} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

- Decision regions described by a half-space.
- $\boldsymbol{w} = (w_0, \ldots, w_d)$ is called the weight vector



Line or plane where z is zero. Half-spaces on the left and right.

Breast Cancer Example




```
# A simple function with a linear decision rule
def predict(X):
    marg = X[:,1]
    size_unif = X[:,0]
    z = marg + 2/3*size_unif - 4
    yhat = (z > 0).astype(int)
    return yhat

# Test on the data
yhat = predict(X)
acc = np.mean(y == yhat)
print('Accuracy = %7.4f' % acc)
```

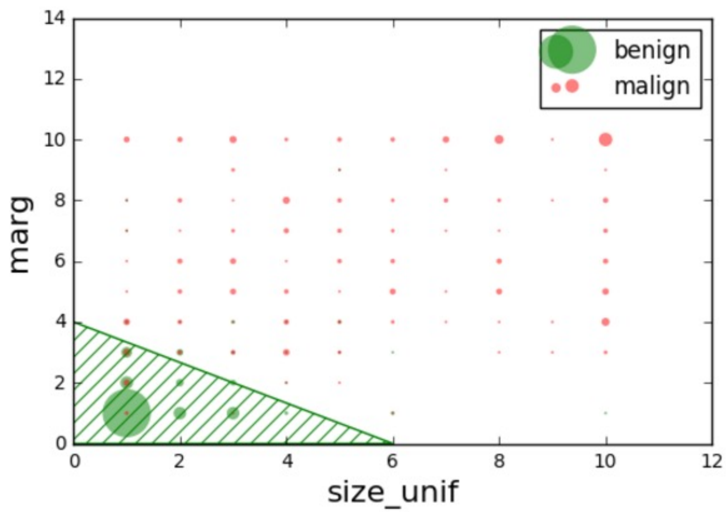
```
Accuracy =  0.9268
```

- From inspection, benign if:

$$\text{marg} + \frac{2}{3}(\text{size\_unif}) < 4$$

- Mathematically:
  - $z = w_0 + w_1(\text{marg}) + w_2(\text{size\_unif})$
  - $\boldsymbol{w} = [-4, 1, \frac{2}{3}]$
  - $\hat{y} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$
- Classification rule from linear constraint
  - Gets 93% accuracy with just 2 features!

Breast Cancer Example



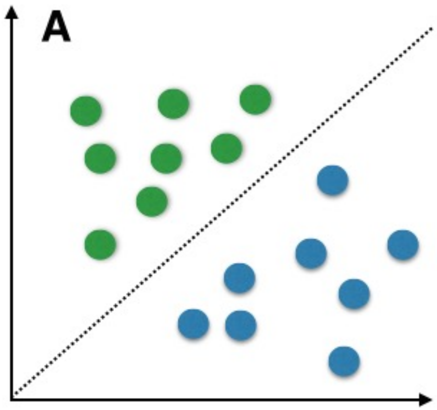

```python
# A simple function with a linear decision rule
def predict(X):
    marg = X[:,1]
    size_unif = X[:,0]
    z = marg + 2/3*size_unif - 4
    yhat = (z > 0).astype(int)
    return yhat

# Test on the data
yhat = predict(X)
acc = np.mean(y == yhat)
print('Accuracy = %7.4f' % acc)
```

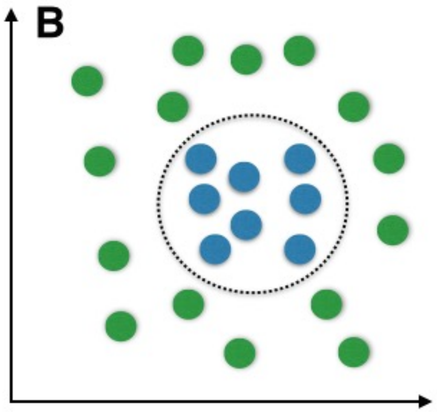
```
Accuracy =  0.9268
```

- Questions for today:
  - How do we use all 10 features?
  - How do we fit an optimal linear classifier?
  - What is optimal?

Linear vs. Non-Linear

- Linear boundaries are limited
- Can only describe very simple regions
- But, serves as building block
  - Many classifiers use linear rules as first step
  - Neural networks, decision trees, …

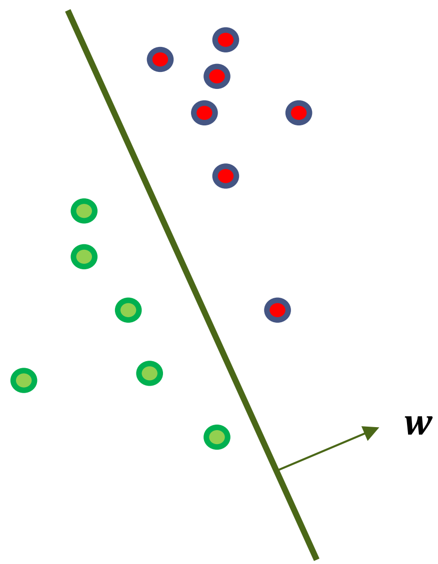




- Breast cancer example:
  - Is the region linear or non-linear?

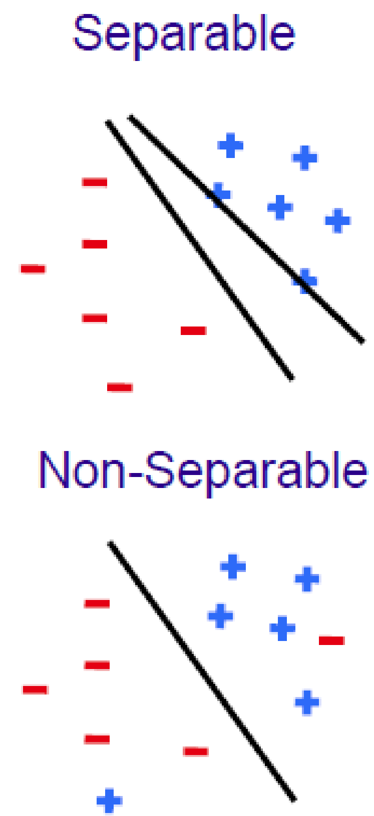
Perfect Linear Separability

- Given training data $(\boldsymbol{x}_i, y_i), i = 1, \dots, N$
  - Binary class label: $y_i = \{0,1\}$
- Perfectly linearly separable if:
  there is a linear classifier that makes no errors on the training data
- Visually: You can draw a line between the points
- Mathematically: There exists a $\boldsymbol{w} = (w_0, w_1, \dots, w_d)$ such that:
  - $w_0 + w_1 x_{i1} + \cdots w_d x_{id} > 0$ when $y_i = 1$
  - $w_0 + w_1 x_{i1} + \cdots w_d x_{id} < 0$ when $y_i = 0$



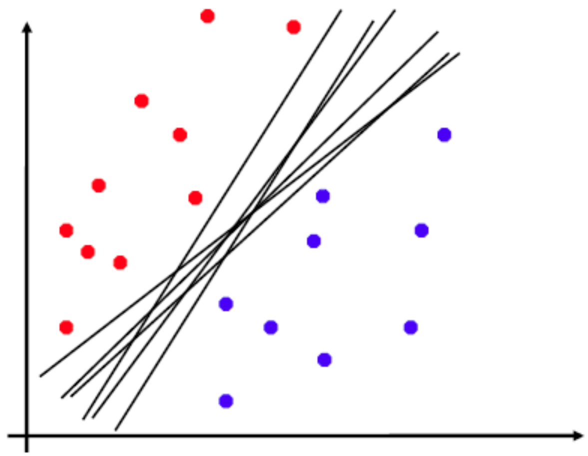
Most Data not Perfectly Separable

- Generally cannot find a separating hyperplane
- Always, some points that will be mis-classified, irrespective of classifier model
- Algorithms attempt to find “good” hyper-planes
  - Reduce the number of mis-classified points
  - Or, some similar metric



Non-Uniqueness

- When one exists, separating hyper-plane is not unique
- Fig. on right: Many separating planes
- Example:
  - If $\boldsymbol{w}$ is separating, then so is $\alpha\boldsymbol{w}$ for all $\alpha > 0$ (since at the end we only care about the sign)
- Which one is optimal? (Next Section!)



Outline

- Motivating Example: Classifying a breast cancer test
- Linear classifiers
- Logistic regression
- Fitting logistic regression models
- Returning to the breast cancer dataset
- Measuring accuracy in classification

Hard vs. Soft Decision Classifiers

- Binary classification problem:
  - Given feature vector x, estimate class label 0 or 1
  - Ex: cat vs. dog
- Hard decision classifier (previous section):
  - Output a class label: $\hat{y} = 0$ or 1
  - Ex: $\hat{y} = 1 \Rightarrow$ *Image is a dog!*
- Soft decision classifier:
  - Does not give a hard outcome
  - Output a conditional probability $P(y = 1|x)$
  - $P(y = 1|x)$ is between 0 and 1
  - Ex: $P(y = 1|x) = 0.9 \Rightarrow$ *Given this image, there is a 90% chance it is a dog*

Cat $y = 0$ Dog $y = 1$

Why Use Soft Decision Classifiers?

- In most problems, classifiers make errors
- Example: Is the digit a 5 or 6?
  - Hard decision makes decision with certainty
  - But the decision can be wrong
- Soft decision classifiers recognize this uncertainty

- **Easier to train soft decision classifier**
  - Allows for error in training data
  - See next section
- Provides a confidence measure

Example from MNIST dataset (Lecture 1)

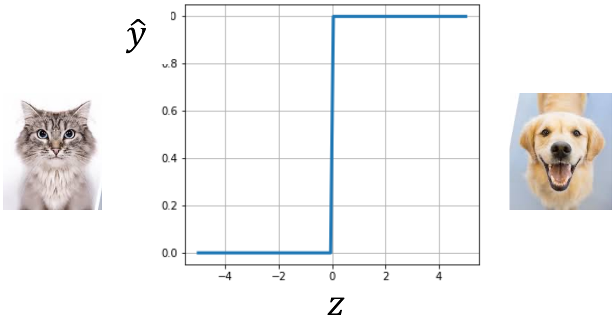
True digit = 5

Hard decision
Digit = 6!

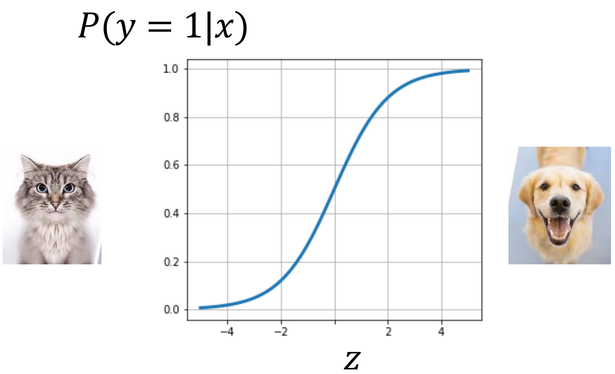
Soft decision
Digit = 5 with probability 30%
Digit = 6 with probability 70%

Logistic Model for Binary Classification

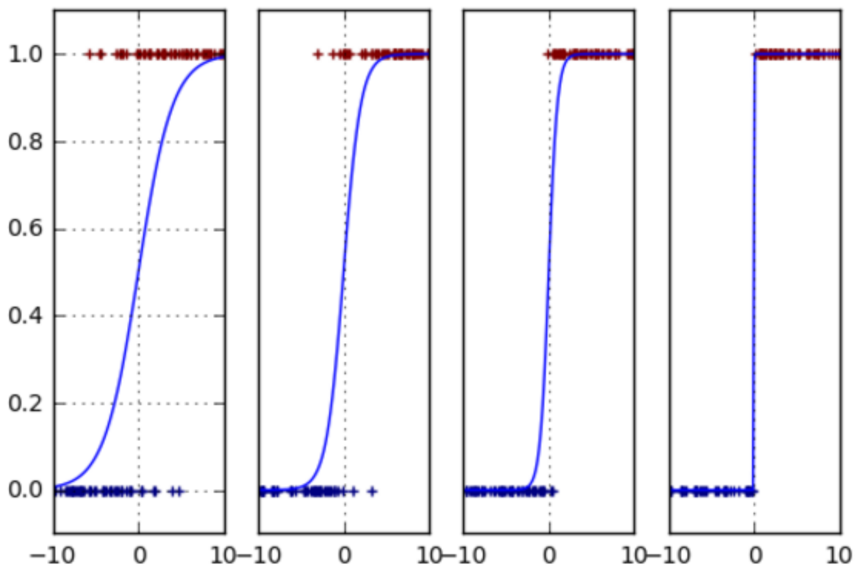
- Binary classification problem: $y = 0, 1$
- Hard decision linear classifier
  - Predict a class label $\hat{y} = 0$ or 1
  - $z = w_0 + \sum_j w_j x_j$
  - $\hat{y} = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$



- Logistic soft decision classifier
  - Predict a probability $P(y = 1|x)$
  - $z = w_0 + \sum_j w_j x_j$
  - Pass z through: $P(y = 1|x) = \frac{1}{1+e^{-z}}$
    - Sometime called Sigmoid function



Logistic Model as a “Soft” Classifier

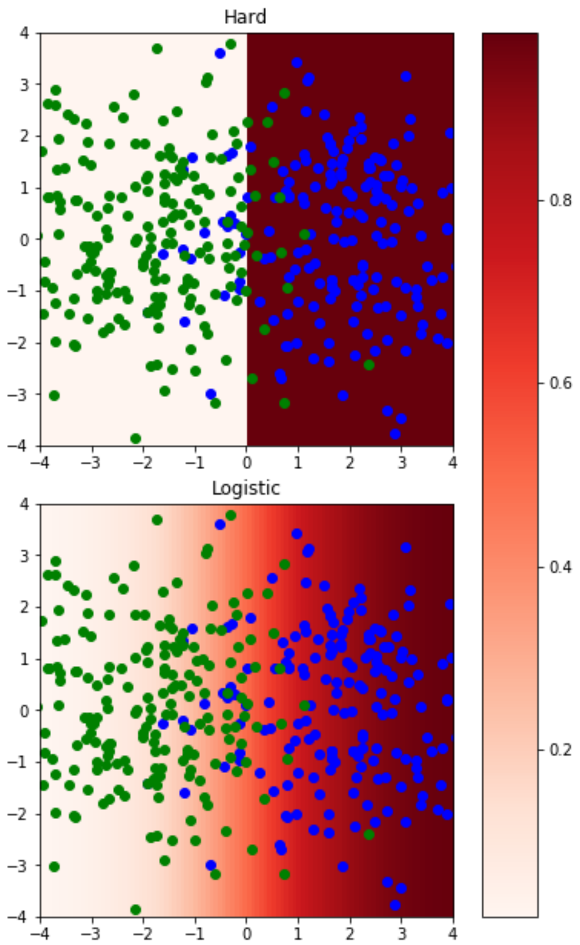


$z = 0.5x$ $\quad z = x$ $\quad z = 2x$ $\quad z = 100x$

- Plot of
  $$P(y = 1|x) = \frac{1}{1 + e^{-z}}, \qquad z = w_1 x$$
  - Markers are random samples
- Higher w_1: prob transition becomes sharper
  - Fewer samples occur across boundary
- As $w_1 \to \infty$ logistic becomes “hard” rule
  $$P(y = 1|x) \approx \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

Hard vs. Soft Classification in 2D

- Hard decision:
  - Divides space into two half-space
  - One for each class label
  - Some wrong decisions
- Soft decision
  - Gradual transition for probability 0 to 1



Multi-Class Logistic Regression

- Logistic regression easily extended to multiple classes
- Suppose $y \in 1, \ldots, K$
  - K possible classes (e.g. digits, letters, spoken words, …)
- Two parameters: $\boldsymbol{W} \in R^{K \times d}, \boldsymbol{w}_0 \in R^K$ Slope matrix and bias (intercept)
- Step 1: Create K linear functions (K outputs for each class).

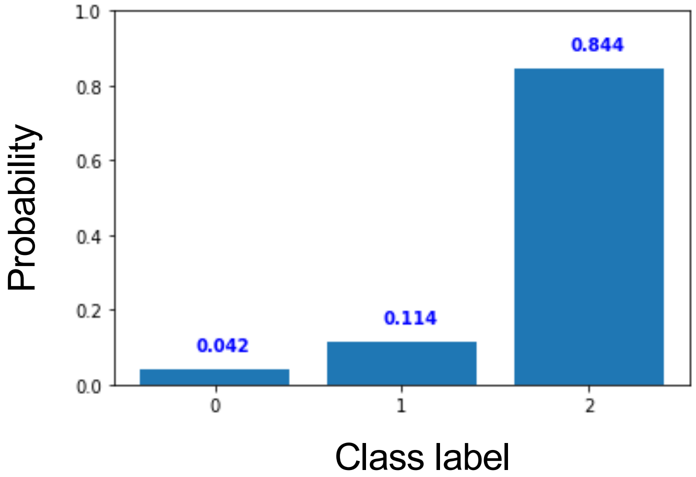
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{w}_0$$

  - Called scores or logits
- Step 2: Predict probabilities via softmax function

$$P(y = k|\boldsymbol{x}) = \frac{\mathrm{e}^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$$

Softmax Example

- Suppose:
  - Three class $y \in \{0,1,2\}$
  - For some x, the logits are $z = [-1, 0, 2]$
- Then $\exp(z) = [0.36, 1, 7.36]$
- Softmax probabilities:
  - $P(y = 0|x) = \frac{0.36}{0.36+1+7.36} \approx 0.042$
  - $P(y = 1|x) = \frac{1}{0.36+1+7.36} \approx 0.114$
  - $P(y = 2|x) = \frac{7.36}{0.36+1+7.36} \approx 0.844$



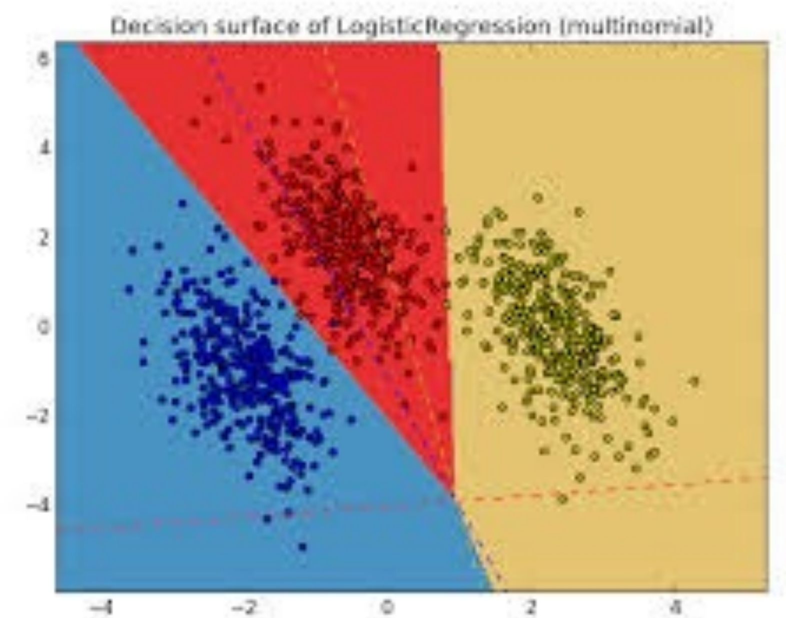
Softmax Properties

- Consider soft-max function:

$$g_k(\mathbf{z}) = \frac{\mathrm{e}^{z_k}}{\sum_{\ell=1}^{K} e^{z_\ell}}$$

  - K inputs $\mathbf{z} = (\mathrm{z}_1, \dots, z_K)$
  - K outputs $g(\mathbf{z}) = (g(\mathbf{z})_1, \dots, g(\mathbf{z})_K)$
- $g(\mathbf{z})$ is like a PMF on the labels $[0,1, \dots, K-1]$
  - $g_k(\mathbf{z}) \in [0,1]$ for each component k
  - $\sum_{k=1}^{K} g_k(\mathbf{z}) = 1$
- Softmax term comes from Softmax property: When $z_k \gg z_\ell$ for all $\ell \neq k$:
  - $g_k(\mathbf{z}) \approx 1$
  - $g_\ell(\mathbf{z}) \approx 0 \text{ for all } \ell \neq k$
- Assigns highest probability to class k when z_k is largest (not necessarily huge)

Multi-Class Logistic Regression Decision Regions



- Each decision region defined by set of hyperplanes
- Intersection of linear constraints
- Sometimes called a polytope

Transform Linear Models

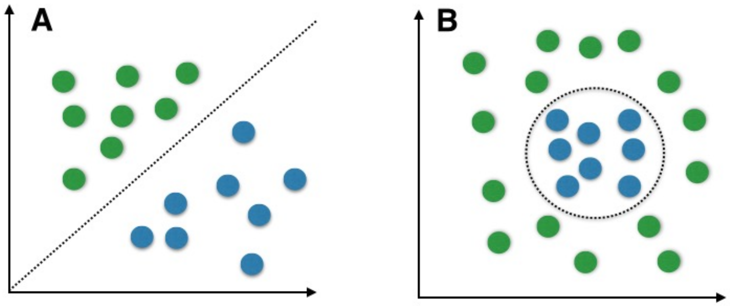
- As in regression, logistic models can be applied to transform features
- Step 1: Map $\boldsymbol{x}$ to some transform features, $\phi(\boldsymbol{x}) = \left[\phi_1(\boldsymbol{x}), \ldots, \phi_p(\boldsymbol{x})\right]^T$
- Step 2: Linear weights: $z_k = \sum_{j=1}^{p} W_{kj}\phi_j(\boldsymbol{x})$
- Step 3: Soft-max $P(y = k|\boldsymbol{z}) = g_k(\boldsymbol{z})$, $g_k(\boldsymbol{z}) = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$

Additional transform step

- Example transforms:
  - Standard regression $\phi(\boldsymbol{x}) = \left[1, x_1, \ldots, x_j\right]^T$ (j original features, j+1 transformed features)
  - Polynomial regression: $\phi(\boldsymbol{x}) = [1, x, \ldots, x^d]^T$ (1 original feature, $d + 1$ transformed features)

Using Transformed Features

- Enables richer class boundaries
- Example: Fig B is not linearly separable



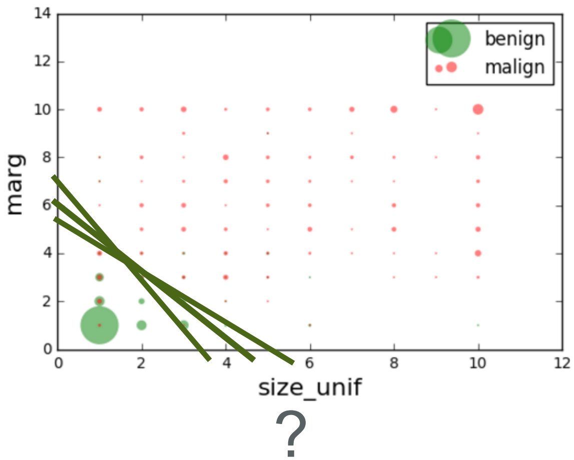
- But, consider nonlinear features
  - $\phi(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_2^2]^T$
- Then can discriminate classes with linear function
  - $w = [-r^2, 0,0,1,1]$
  - $z = w^T \phi(x) = x_1^2 + x_2^2 - r^2$
  - Blue when $z \leq 0$ and Green when $z > 0$ (if origin was at center)

Outline

- Motivating Example: Classifying a breast cancer test
- Linear classifiers
- Logistic regression
- Fitting logistic regression models
- Returning to the breast cancer dataset
- Measuring accuracy in classification

How To Fit Logistic Models?

- Given training data, $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$
  - Binary labels $y_i \in \{0,1\}$



- Binary logistic model: Given *new* $\boldsymbol{x}$ predict class probability via:
  - Linear weights: $\boldsymbol{z} = \boldsymbol{w}_{1:p}^T \boldsymbol{x} + w_0$
  - Sigmoid: $P(y = 1|\boldsymbol{x}) = \frac{1}{1+e^{-z}}$

- Weight vector $\boldsymbol{w}$ represents unknown model parameters

- Learning problem: Learn weight vector $\boldsymbol{w}$ from the data

Maximum Likelihood Principle

- Likelihood function: From the logistic model, we can derive:

$P(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})$ = Probability of class labels given inputs $\boldsymbol{X}$ and weights $\boldsymbol{w}$

  - $(\boldsymbol{X}, \boldsymbol{y})$ are the data matrices for all n training samples
  - $\boldsymbol{w}$ is the vector of parameters

- Key idea: $P(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})$ is higher $\Rightarrow$ data is a better match with the parameters

- Maximum Likelihood Principle: Given data $(\boldsymbol{X}, \boldsymbol{y})$:

Find parameters $\boldsymbol{W}$ to maximize $P(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{W})$

Binary Cross Entropy

- Given data $(x_i, y_i), i = 1, \dots, N$ with binary labels $y_i \in \{0,1\}$
- Theorem: MLE for logistic model is equivalent to minimizing the binary cross entropy:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\ln[1 + e^{z_i}] - y_i z_i \right), \qquad z_i = w_0 + \sum_{j=1}^{d} w_j x_{ij}$$

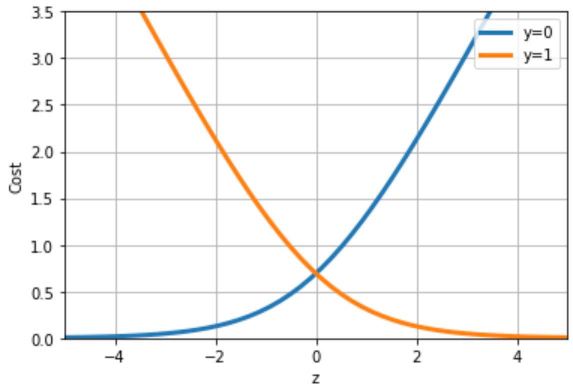
  - Find the weight vector $\boldsymbol{w}$ to minimize $J(\boldsymbol{w})$
  - Will prove below this is equivalent to maximizing $P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})$
- Provides a simple cost function to minimize for fitting
- Note that z_i are implicitly function of weights $\boldsymbol{w}$

Visualizing BCE

- Binary cross entropy

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\ln[1 + e^{z_i}] - y_i z_i\right), \qquad z_i = w_0 + \sum_{j=1}^{d} w_j x_{ij}$$

- Each term has cost $\ln[1 + e^{z_i}] - y_i z_i$
- $y_i = 0 \Rightarrow$ Tries to make z_i negative
- $y_i = 1 \Rightarrow$ Tries to make z_i positive

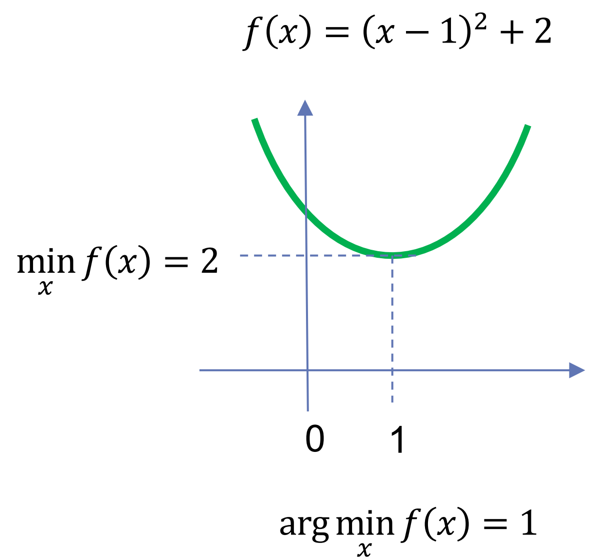


Higher cost for $y = 1$

Higher cost for $y = 0$

Min and Argmin

- Given a function $f(x)$
- $\min_x f(x)$
  - Minimum value of the $f(x)$
  - Point on the y-axis
- $\arg\min_x f(x)$
  - Value of x where $f(x)$ is a minimum
  - Point on the x-axis



- Similarly, define $\max_x f(x)$ and $\arg\max_x f(x)$

MLE Using Argmax

- We can write the MLE using argmax
- Suppose we have
  - Data $(\boldsymbol{X}, \boldsymbol{y})$
  - Likelihood function $P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})$
- Then, MLE is equivalent to:

$$\boxed{\widehat{\boldsymbol{w}} = \arg\max_{w} P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{w})}$$

  - Equivalent to saying find $\mathbf{w}$ to maximize $\mathrm{P}(\mathbf{y}|\mathbf{X}, \mathbf{w})$

Log Likelihood

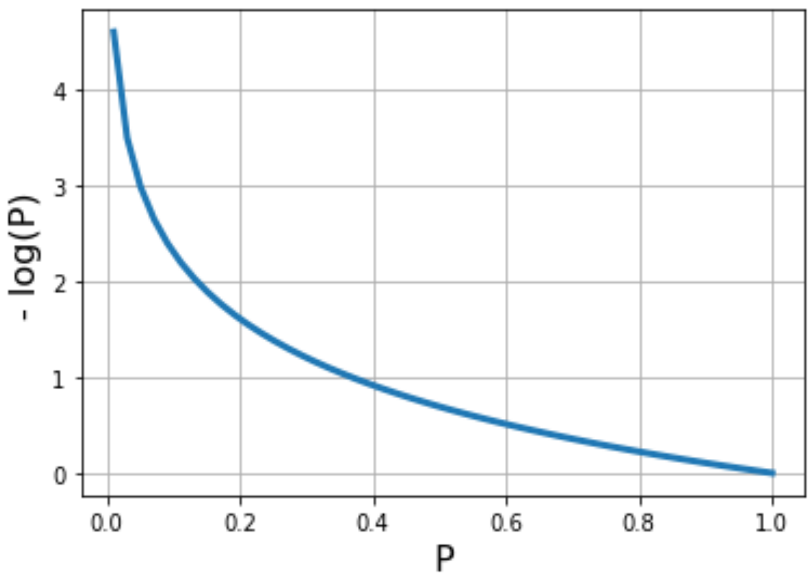
- Assume outputs y_i are independent, depending only on $\boldsymbol{x}_i$
- Then, likelihood factors:

$$P(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w}) = \prod_{i=1}^{N} P(y_i|\boldsymbol{x}_i,\boldsymbol{w})$$

- Define negative log likelihood:

$$L(\boldsymbol{w}) = -\ln P(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})$$
$$= -\sum_{i=1}^{N} \ln P(y_i|\boldsymbol{x}_i,\boldsymbol{w})$$

Negative Log Likelihood



- Maximum likelihood estimator can be re-written as:

$$\widehat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} P(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w})$$

Logistic Loss = Binary Cross Entropy

- Negative log likelihood function: $J(w) = -\sum_{i=1}^{n} \ln P(y_i|\boldsymbol{x}_i, \boldsymbol{w})$

$$P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{w}) = \frac{1}{1 + e^{-z_i}}, \qquad z_i = \boldsymbol{w}_{1:p}^T \boldsymbol{x}_i + w_0$$

- Therefore,

$$P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{w}) = \frac{e^{z_i}}{1 + e^{z_i}}, \qquad P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{w}) = \frac{1}{1 + e^{z_i}}$$

- Hence,

$$\ln P(y_i|\boldsymbol{x}_i, \boldsymbol{w}) = y_i \ln P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{w}) + (1 - y_i) \ln P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{w})$$
$$= y_i z_i - \ln[1 + e^{z_i}]$$

A bit more rearrangement

- Loss function = binary cross entropy:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} (\ln[1 + e^{z_i}] - y_i z_i)$$

Multi-Class Classification

- For multi-class classification, define the "one-hot" vector:

$$r_{ik} = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases}, \qquad i = 1, \dots, N, \qquad k = 1, \dots, K$$

- Then, $\ln P(y_i | \boldsymbol{x}_i, \boldsymbol{W}) = \sum_{k=1}^{K} r_{ik} \ln P(y_i = k | \boldsymbol{x}_i, \boldsymbol{W})$
- Hence, negative log likelihood is:

$$\boxed{J(\boldsymbol{W}) = \sum_{i=1}^{N} \left[\ln \left[\sum_k e^{z_{ik}} \right] - \sum_k z_{ik} r_{ik} \right]}$$

  - Sometimes called the categorial cross-entropy
  - Can prove this with some algebra (derivation removed)

Gradient Calculations

- To minimize take partial derivatives: $\frac{\partial J(W)}{\partial W_{kj}} = 0$ for all W_{kj}
- Define transform matrix: $A_{ij} = \phi_j(\boldsymbol{x}_i)$
- Hence, $z_{ik} = \sum_{j=1}^{p} A_{ij} W_{kj}$
- Estimated class probabilities: $p_{ik} = \frac{e^{z_{ik}}}{\sum_{\ell} e^{z_{i\ell}}}$
- Gradient components are (proof removed): $\frac{\partial L(\boldsymbol{W})}{\partial W_{kj}} = \sum_{i=1}^{N} (p_{ik} - r_{ik}) A_{ij} = 0$
  - $K \times p$ equations and $K \times p$ unknowns
- Unfortunately, no closed-form solution to these equations
  - Nonlinear dependence of p_{ik} on terms in W

Numerical Optimization

- We saw that we can find minima by setting $\nabla f(x) = 0$
  - M equations and M unknowns.
  - May not have closed-form solution

- Numerical methods: Finds a sequence of estimates x^t

$$x^t \to x^*$$

  - Start at some guess point
  - Under some conditions, it converges to some other "good" minima
  - Run on a computer program, like python

- Next lecture: Will discuss numerical methods to perform optimization
- This lecture: Use built-in python routine

Outline

- Motivating Example: Classifying a breast cancer test
- Linear classifiers
- Logistic regression
- Fitting logistic regression models
- Returning to the breast cancer dataset
- Measuring accuracy in classification

Fitting Two Variables

```
xnames =['size_unif','marg']
X = np.array(df[xnames])
print(X.shape)
```

```
(683, 2)
```

```
scal = StandardScaler()
Xtr1 = scal.fit_transform(Xtr)
Xts1 = scal.transform(Xts)
```

```
reg = linear_model.LogisticRegression(C=1e5)
reg.fit(Xtr1, ytr)
```

```
yhat = reg.predict(Xts1)
acc = np.mean(yhat == yts)
print("Accuracy on test data = %f" % acc)
```

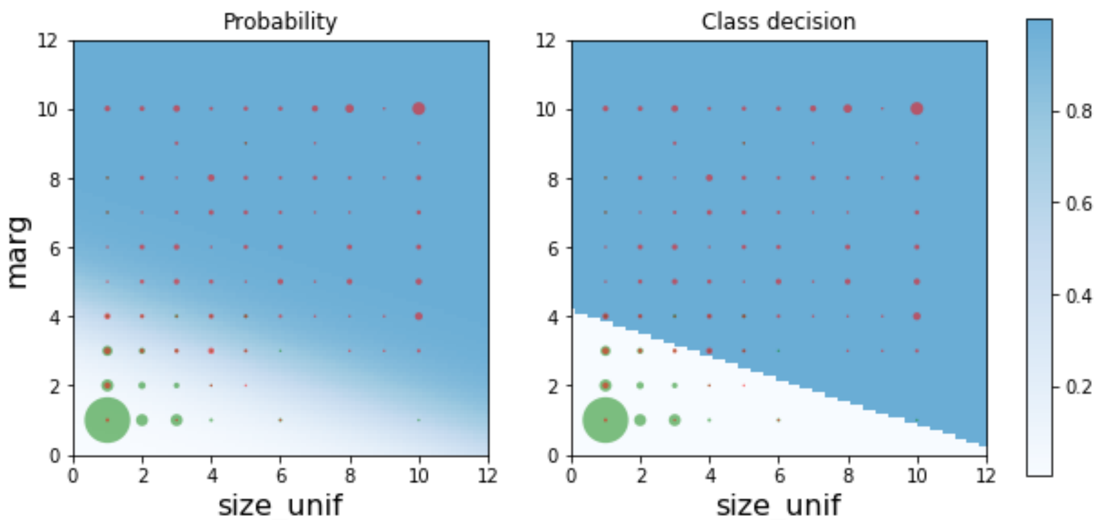
```
Accuracy on test data = 0.917073
```

- Get the two variables
- Scale, fit and test
- Regularization note for sklearn
  - Always performs regularized regression:
  - Minimizes

$$J(w) + \frac{1}{2C}\|w\|^2$$

  - $J(w) =$ Negative log likelihood
  - $\frac{1}{2C}\|w\|^2 =$ regularization term (like Ridge)
  - Used for numerical stability

Visualizing the Decision Regions



- Probability output:
  - Smooth value from 0 to 1
- Class decision:
  - $\hat{y} = 1$ when $P(y = 1|x) > 0.5$
  - $\hat{y} = 0$ when $P(y = 1|x) \leq 0.5$
  - Decision boundary is a line
  - Similar to what we manually selected

Fitting All The Variables

```
# Get array of all the features. These are the columns in the dataframe
# except the last
xnames = names[:-1]
X = np.array(df[xnames])
print(X.shape)

# Split into training and test
Xtr, Xts, ytr, yts = train_test_split(X,y, test_size=0.30)
```

```
# Scale the data
scal = StandardScaler()
Xtr1 = scal.fit_transform(Xtr)
Xts1 = scal.transform(Xts)

# Fit on the scaled trained data
reg = linear_model.LogisticRegression(C=1e5)
reg.fit(Xtr1, ytr)

# Measure accuracy
yhat = reg.predict(Xts1)
acc = np.mean(yhat == yts)
print("Accuracy on training data = %f" % acc)
```

```
Accuracy on training data = 0.970732
```

- With all 9 variables:
  - Get 97% test accuracy
- 10-fold cross validation
  - Also in code
  - Accuracy = 0.967
  - SE = 0.011

	feature	slope
0	id	-0.513702
1	thick	1.498450
2	size_unif	-0.293459
3	shape_unif	0.702795
4	marg	1.207218
5	cell_size	0.145706
6	bare	1.153128
7	chrom	1.459208
8	normal	0.719531
9	mit	0.785216

Outline

- Motivating Example: Classifying a breast cancer test
- Linear classifiers
- Logistic regression
- Fitting logistic regression models
- Returning to the breast cancer dataset
- Measuring accuracy in classification

Errors in Binary Classification

- Two types of errors:
  - Type I error (False positive / false alarm): Decide $\hat{y} = 1$ when $y = 0$
  - Type II error (False negative / missed detection): Decide $\hat{y} = 0$ when $y = 1$
- Implication of these errors may be different
  - Think of breast cancer diagnosis
- Accuracy of classifier can be measured by:

predicted→ real↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

$$\text{TPR (sensitivity)} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

$$\text{FPR (1-specificity)} = \frac{\text{FP}}{\text{TN}+\text{FP}}$$

  - $TPR = P(\hat{y} = 1|y = 1)$
  - $FPR = P(\hat{y} = 1|y = 0)$
  - Accuracy=$P(\hat{y} = 1|y = 1)$+ $P(\hat{y} = 0|y = 0)$
    - (percentage of correct classification)

Many Other Metrics

- From previous slide
  - $TPR = P(\hat{y} = 1|y = 1)$=sensitivity
  - $FPR = P(\hat{y} = 1|y = 0)$=1-specificity

predicted → real ↓	*Class_pos*	*Class_neg*
Class_pos	TP	FN
Class_neg	FP	TN

$$\text{TPR (sensitivity)} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

$$\text{FPR (1-specificity)} = \frac{\text{FP}}{\text{TN}+\text{FP}}$$

- Machine learning often uses (positive=items of interests in retrieval applications)
  - Recall = Sensitivity =TP/(TP+FN) (How many positives are detected among all positive?)
  - Precision =TP/(TP+FP) (How many detected positive is actually positive?)
  - F1-score = $\frac{Precision * Recall}{(Precision+Recall)/2} = \frac{2TP}{2TP+FN+FP} = \frac{TP}{TP+\frac{FN+FP}{2}}$
  - Accuracy=(TP+TF)/(TP+FP+TN+FN) (percentage of correct classification)
- Medical tests:
  - Sensitivity = $P(\hat{y} = 1|y = 1) = TPR$
  - Specificity = $P(\hat{y} = 0|y = 0) = 1 - FPR$ =True negative rate
  - Need a good tradeoff between sensitivity and specificity

Breast Cancer

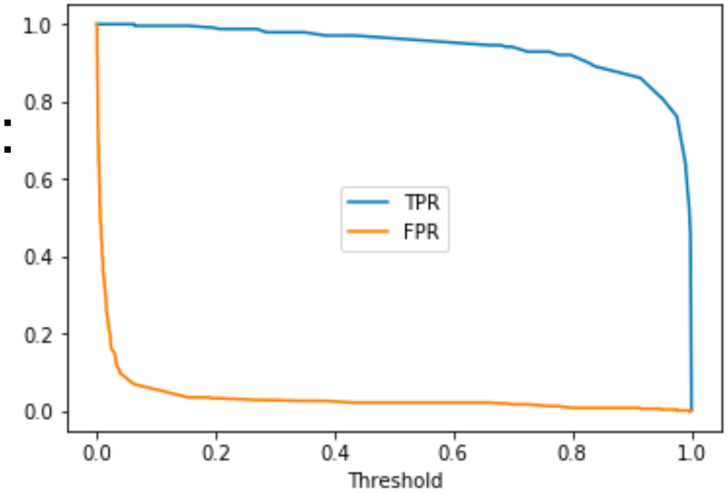
- Measure accuracy on test data
- Use 4-fold cross-validation
- Sklearn has built-in functions for CV

```
Precision =  0.9614
Recall =     0.9554
f1 =         0.9578
Accuracy =   0.9664
```

```python
from sklearn.model_selection import KFold
from sklearn.metrics import precision_recall_fscore_support
nfold = 4
kf = KFold(n_splits=nfold)
prec = []
rec = []
f1 = []
acc = []
for train, test in kf.split(Xs):
    # Get training and test data
    Xtr = Xs[train,:]
    ytr = y[train]
    Xts = Xs[test,:]
    yts = y[test]

    # Fit a model
    logreg.fit(Xtr, ytr)
    yhat = logreg.predict(Xts)

    # Measure
    preci,reci,f1i,_= precision_recall_fscore_support(yts,yhat,average='binary')
    prec.append(preci)
    rec.append(reci)
    f1.append(f1i)
    acci = np.mean(yhat == yts)
    acc.append(acci)

# Take average values of the metrics
precm = np.mean(prec)
recm = np.mean(rec)
f1m = np.mean(f1)
accm= np.mean(acc)

print('Precision = {0:.4f}'.format(precm))
print('Recall =    {0:.4f}'.format(recm))
print('f1 =        {0:.4f}'.format(f1m))
print('Accuracy =  {0:.4f}'.format(accm))
```

Hard Decisions

- Logistic classifier outputs a soft label: $P(y=1|x) \in [0,1]$
  - $P(y=1|x) \approx 1 \Rightarrow y=1$ more likely
  - $P(y=0|x) \approx 1 \Rightarrow y=0$ more likely
- Can obtain a hard label by thresholding:
  - Set $\hat{y}=1\ if\ \ P(y=1|x) > t$
  - t = Threshold

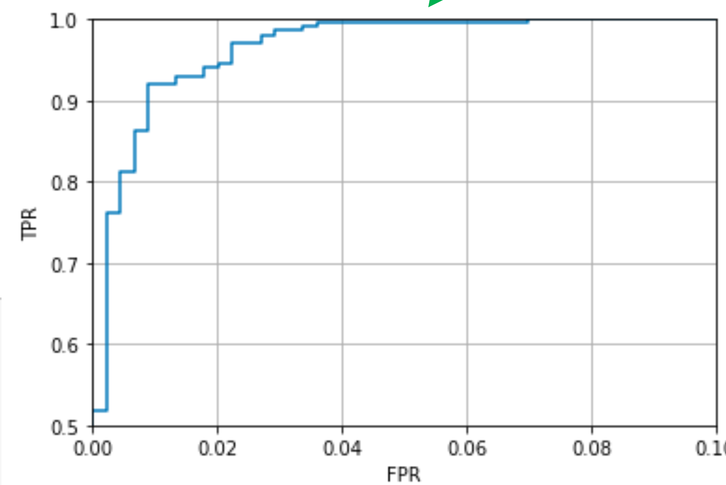


- How to set threshold?
  - Set $t=\frac{1}{2} \Rightarrow$ Minimizes overall error rate
  - Increasing $t \Rightarrow$ Decreases false positives, but also reduces sensitivity
  - Decreasing $t \Rightarrow$ Increases sensitivity, but also increases false positive

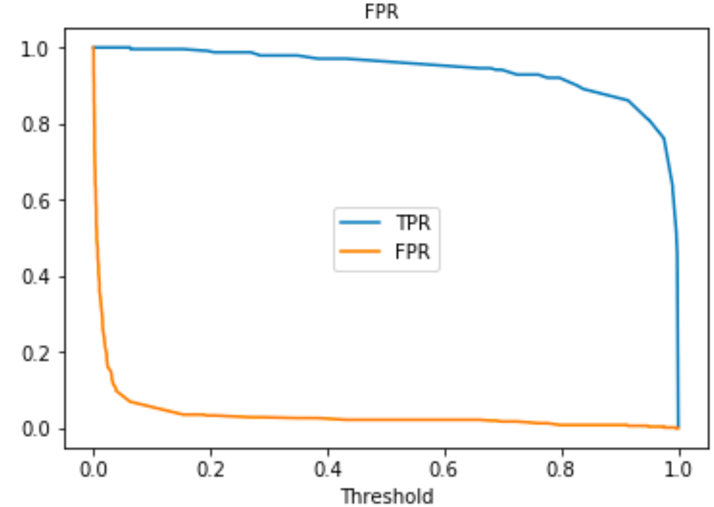
ROC Curve

Values for different thresholds

- Varying threshold obtains a set of classifier
- Trades off FPR (1-specificity) and TPR (sensitivity)
- Can visualize with ROC curve
  - Receiver operating curve
  - Term from digital communications




```
from sklearn import metrics
yprob = logreg.predict_proba(Xs)
fpr, tpr, thresholds = metrics.roc_curve(y,yprob[:,1])

plt.plot(fpr,tpr)
plt.grid()
plt.xlabel('FPR')
plt.ylabel('TPR')
```



https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

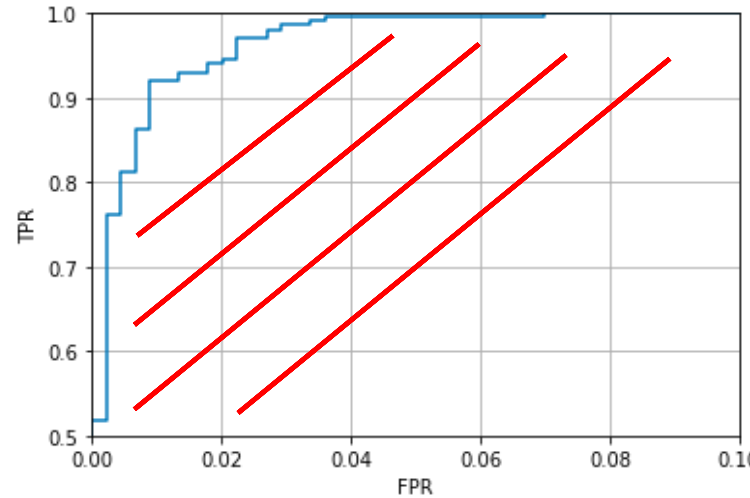
Area Under the Curve (AUC)

- As one may choose a particular threshold based on the desired trade-off between the TPR and FPR, it may not be appropriate to evaluate the performance of a classifier for a fixed threshold.

- AUC is a measure of goodness for a classifier that is independent of the threshold.

- A method with a higher AUC means that under the same FPR, it has higher PPR.

- Should report average AUC over cross validation folds




```
uac=metrics.roc_auc_score(y,yprob[:,1])
print("UAC=%f" % uac)
```

UAC=0.996315

Multi-Class Classification in Python

- Two options
- One vs Rest (OVR)
  - Solve a binary classification problem for each class k
  - For each class k, train on modified binary labels (indicates if sample is in class or not)

$$\tilde{y}_i = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{if } y_i \neq k \end{cases}$$

  - Predict based on classifier that yields highest score
- Multinomial
  - Directly solve weights for all classes using the multi-class cross entropy

Confusion Matrix for Multi-Class Classification

Vowel produced \ Perceived vowel	i	e	a	o	u
i	15		1		
e	1		1		
a			79	5	
o			4	15	3
u				2	2

Metrics for Multiclass Classification

- Using a $K \times K$ confusion matrix
- Should normalize the matrix:
  - Sum over each row =1
- Can compute accuracy:
  - Per class: This is the diagonal entry
  - Average: The average of the diagonal entries

Pred--> Real↓	1	2	…	K
1				
2				
…				
K				

LASSO Regularization for Logistic Regression

- Similar to linear regression, we can use LASSO regularization with logistic regression
  - Forces the weighting coefficients to be sparse.
- Add L1 penalty $L(\boldsymbol{W}) = \sum_{i=1}^{N} [\ln[\sum_k e^{z_{ik}}] - z_{ik} r_{ik}] + \lambda \, ||W||_1$
- The regularization level λ should be chosen via cross validation as before
- Sklearn implementation:

```
logreg = linear_model.LogisticRegression(penalty='l1')
logreg.C = c
```

  - Default use l2 penalty, to reduce the magnitude of weights
- C is the inverse of regularization strength $(C = 1/\lambda)$; must be a positive float.
  - Should use a large C if you do not want to apply regularization
- Go through the LASSO part of the code