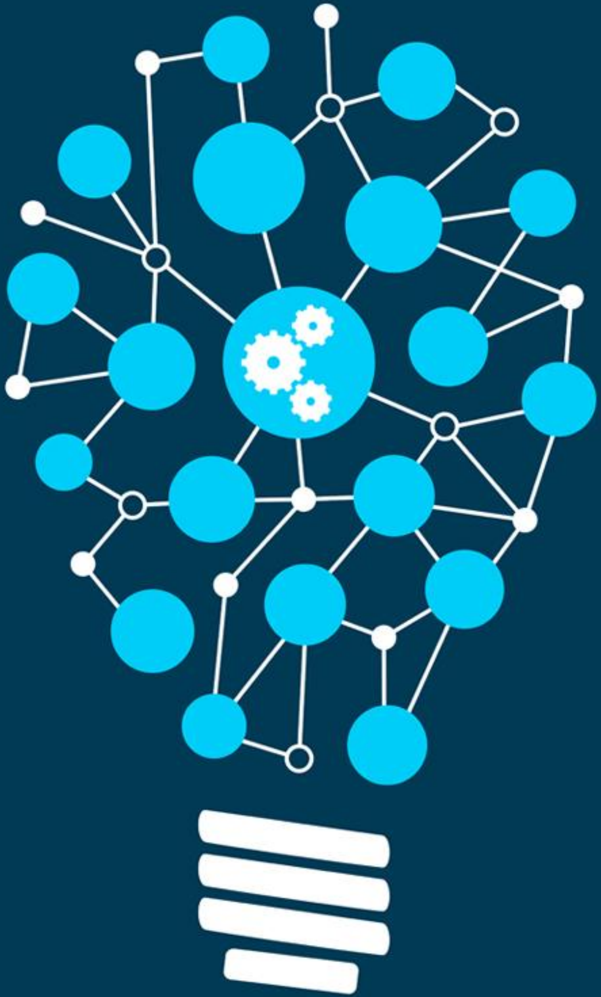
NYU

NYU Summer Machine Learning Program

Presenter Name Here
Date Here



NYU

Linear Classification

Day 4

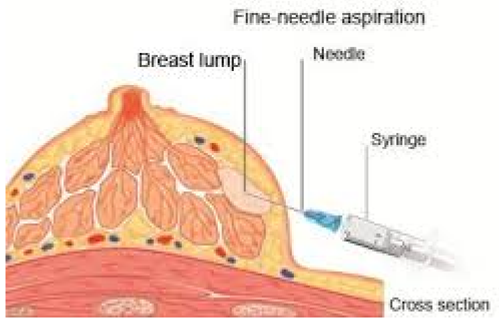
Learning Objectives

- ❑Formulate a machine learning problem as a classification problem
    - ❑ Identify features, class variable, training data
- ❑Visualize classification data using a scatter plot.
- ❑Describe a linear classifier as an equation and on a 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
- ❑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.

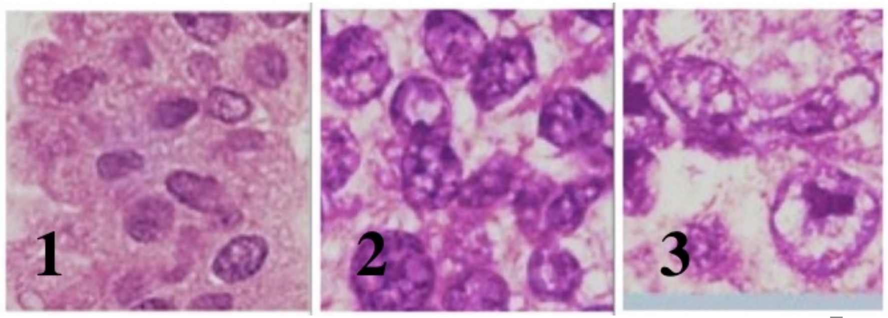
Outline

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

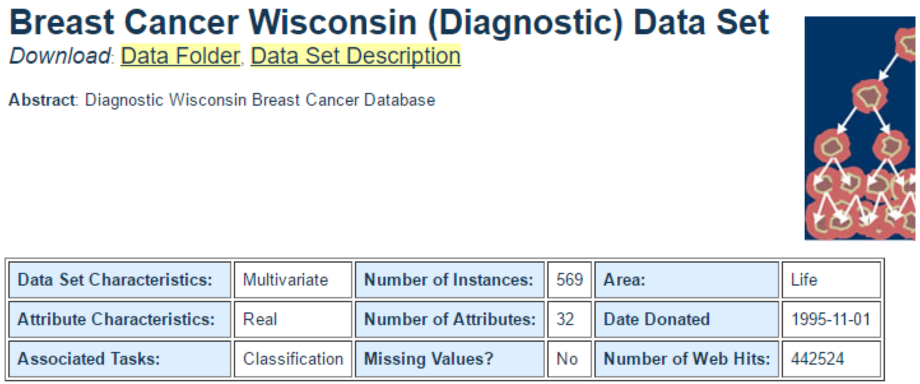
Example - Diagnosing Breast Cancer



- ❑Fine needle aspiration of suspicious lumps
- ❑Cytopathologist visually inspects cells
  - ❑ Sample is stained and viewed under microscope
- ❑Uses many features:
  - ❑ Size and shape of cells, degree of mitosis, differentiation, …
- ❑Diagnosis is not exact and if uncertain, use a more comprehensive biopsy
  - ❑ Additional cost and time
  - ❑ Stress to patient
- ❑Determines if cells are benign or malignant
- ❑Can machine learning provide better rules?



Data



Breast Cancer Wisconsin (Diagnostic) Data Set

Download: Data Folder, Data Set Description

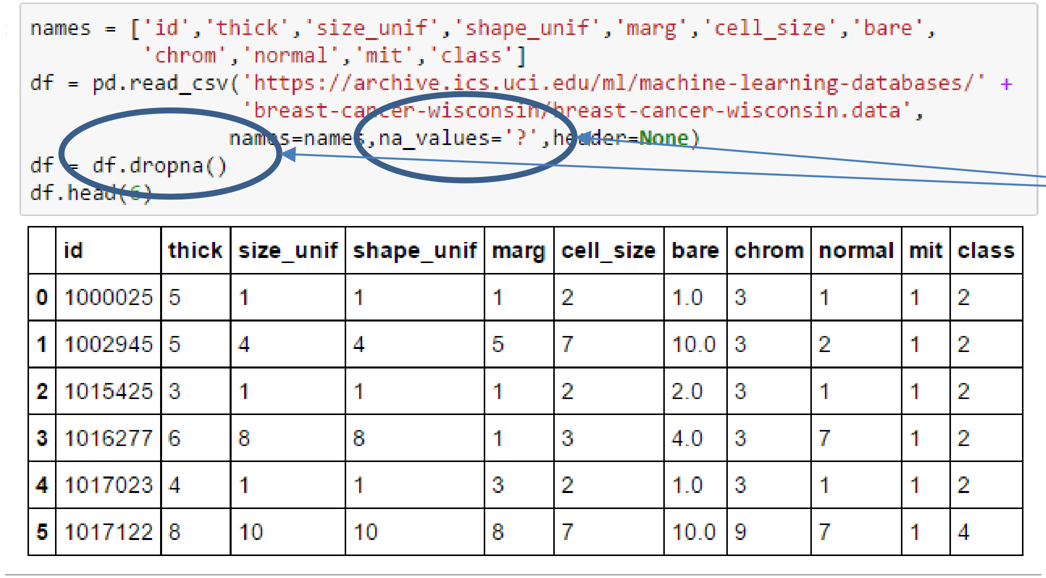
Abstract: Diagnostic Wisconsin Breast Cancer Database

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

- ❑Univ. Wisconsin study, 1994
- ❑569 samples
- ❑10 visual features for each sample
- ❑Ground truth determined by biopsy

Sample ID (code number)
Clump thickness
Uniformity of cell size
Uniformity of cell shape
Marginal adhesion
Single epithelial cell size
Number of bare nuclei
Bland chromatin
Number of normal nuclei
Mitosis

Loading The Data



```
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

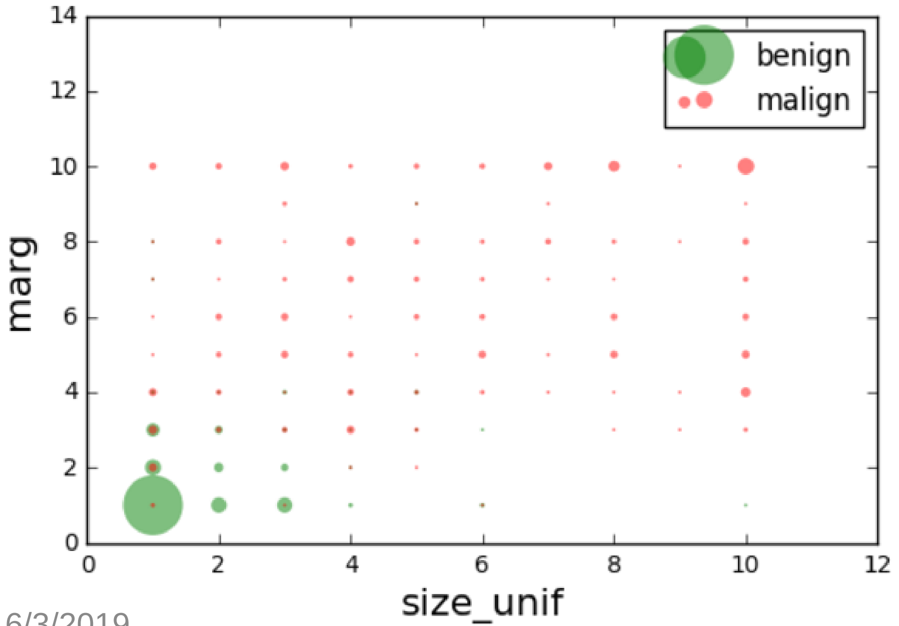
❑Follow standard Pandas routine

Drop missing values

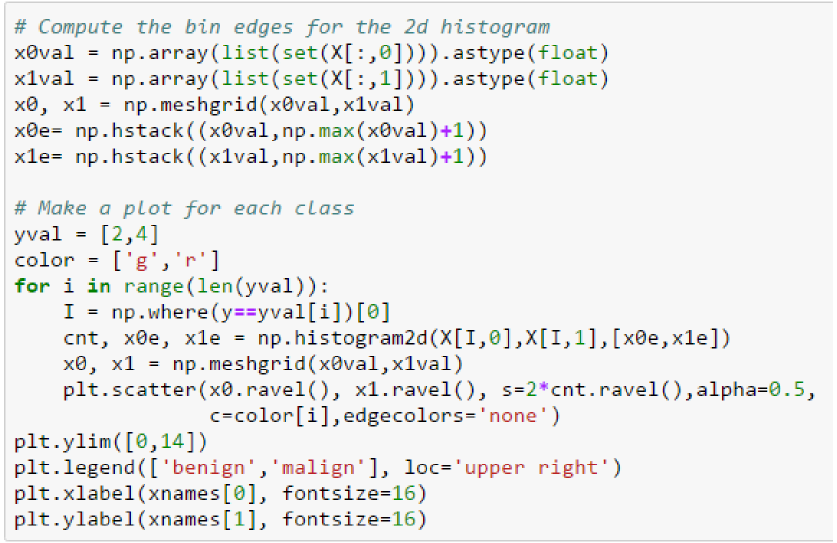
Class 2 is benign

Class 4 is malignant

Visualizing the Data – In Google Colab Notebook



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

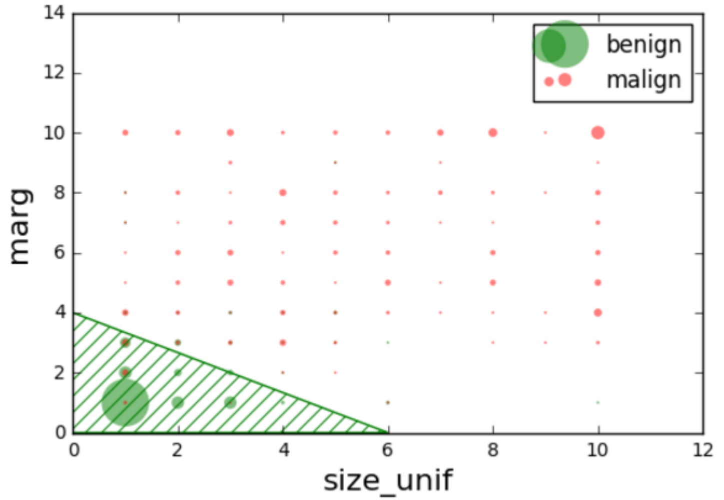


```
# Compute the bin edges for the 2d histogram
x0val = np.array(list(set(X[:,0]))).astype(float)
x1val = np.array(list(set(X[:,1]))).astype(float)
x0, x1 = np.meshgrid(x0val,x1val)
x0e= np.hstack((x0val,np.max(x0val)+1))
x1e= np.hstack((x1val,np.max(x1val)+1))

# 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)
```

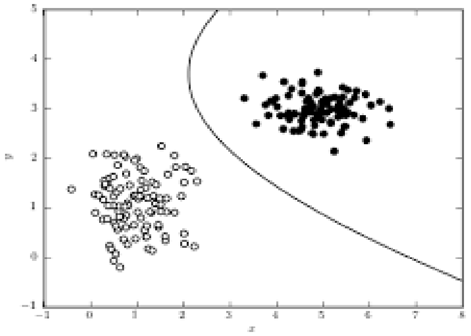
A Possible Rule

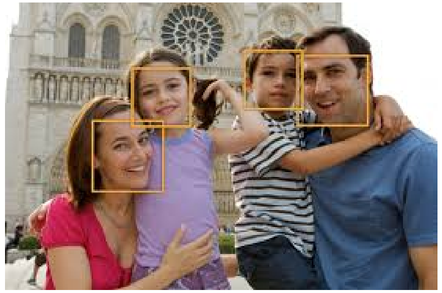
- From inspection, benign if:
  - $marg + \frac{2}{3} * (size\_unif) < 4$
- Classification rule from linear constraint
- What are other possible classification rules?
- Every rule misclassifies some points
- What is optimal?



What is Classification?

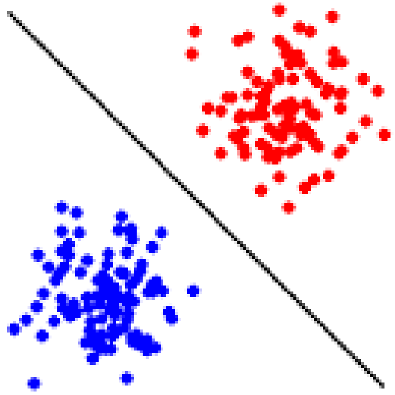
- Given features $\mathbf{x}$, determine its class label, $y = 1,\ldots,K$
- Many applications:
  - Face detection: Is a face present or not?
  - Reading a digit: Is the digit 0,1,…,9?
  - Are the cells cancerous or not?
  - Is the email spam?
- Equivalently, determine classification function: $y' = f(\mathbf{x}) \in \{1,\ldots,K\}$
  - Like regression, but with a discrete response
  - May index $\{1,\ldots,K\}$ or $\{0,\ldots,K-1\}$





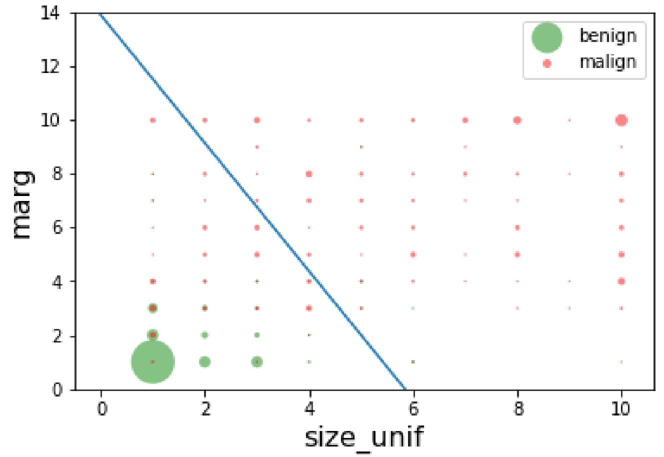
Linear Classifier

- General binary classification rule: y_hat = $f(x)$ = 0 or 1
- Linear classification rule:
  - Take linear combination $\mathrm{z} = w_0 + \Sigma_{j=1}^{\mathrm{d}} \mathrm{w_d} * \mathrm{x\_d}$
- Predict class from z
  - $y_{hat} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$
- Decision regions described by a half-space.
- $\mathrm{w} = (w_0, w_1, \ldots\ldots w_d)$ is called the weight vector



Bad Idea – Using Linear Regression for Classification

- Bad idea: Use linear regression
  - Labels are $y \in \{0,1\}$
  - Use linear model $z = w_0 + \Sigma_{j=1}^{d} w_d * x\_d$
  - Find linear fit so that $\Sigma_i \, (y_i - z_i)^2$ is minimized
  - Then threshold the linear fit: $y_{hat} = \begin{cases} 1 & z > 0.5 \\ 0 & z < 0.5 \end{cases}$
- This yields the line: $w_0 + w_1 * x_1 + w_2 * x_2 = 0.5$
- Why the line is not as expected?
- Should not use MSE as the optimization criterion!
  - Squared error is not related to classifier accuracy

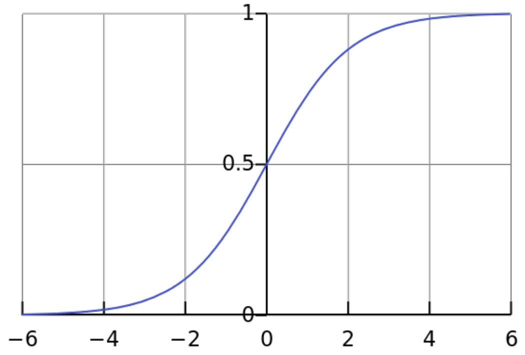


```
yhat=regr.predict(X)
yhati= (yhat >=0).astype(int)
acc = np.mean(yhati == y)
print("Accuracy on training data using two features = %f" % acc)
```

```
Accuracy on training data using two features = 0.922401
```

Logistic Model for Binary Classification

- Binary classification problem: y = 0, 1
- Consider probabilistic model
  - $P(y=1|x) = \frac{1}{1+e^{-z}}$ & $P(y=0|x) = \frac{e^{-z}}{1+e^{-z}}$
  - $z = w_0 + \Sigma_{j=1}^{k} w_k x_k$
- Logistic function: $f(z) = \frac{1}{1+e^{-z}}$
  - Classical "S"-shape. Also called sigmoidal
- Value of f($\boldsymbol{x}$) does not perfectly predict class y.
  - Only a probability of y



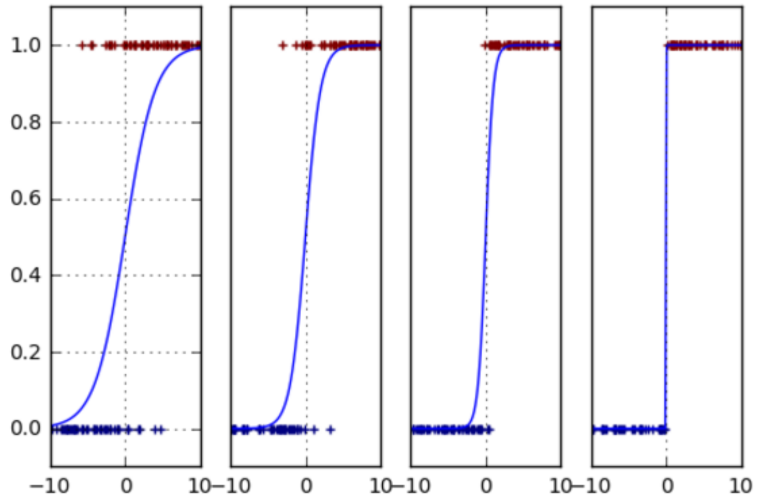
Logistic Model as a “Soft” Classifier

- Plot of

  $P(y = 1|x) = \frac{1}{1+e^{-z}}; \qquad z = w_1 x$

  - Markers are random samples
- Higher w_1: probability transition becomes sharper
  - Fewer samples occur across boundary
- As w_1 -> infinity, logistic becomes “hard” rule

$$P(y = 1|x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$



Multi-Class Logistic Regression

- Suppose $y \in 1, \ldots, K$
  - K possible classes (e.g. digits, letters, spoken words, …)
- Multi-class regression:
  - $\boldsymbol{W} \in R^{(K \times d)}$, $\boldsymbol{w}_0 \in R^K$ is the slope matrix and bias
  - $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{w}_0$: Creates K linear functions
- Then, class probabilities given by:

$$P(y = k|x) = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}}$$

SoftMax Operation

- Consider soft-max function:

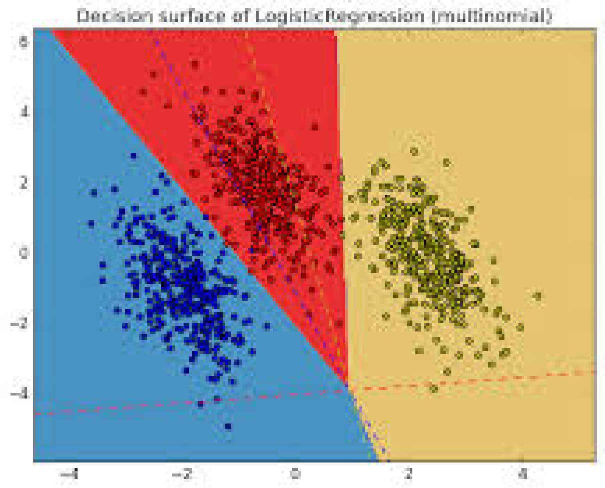
$$g_k(z) = \frac{e^{z_k}}{\Sigma_{l=1}^{K} e^{z_l}}$$

  - K inputs $\mathbf{z} = (z_1, \ldots, z_K)$, K outputs $f(\mathbf{z}) = (f(\mathbf{z})_1, \ldots, f(\mathbf{z})_K)$
- Properties: $f(\mathbf{z})$ is like a PMF on the labels $[0,1,\ldots, K-1]$
  - $g_k(\mathbf{z}) \in [0,1]$ for each component k
  - $\Sigma_{k=1}^{K} g_k(\mathbf{z}) = 1$
- SoftMax property: When $z_k \gg z_\ell$ for all $\ell \neq k$:
  - $g_k(z) \approx 1$
  - $g_\ell(\mathbf{z}) \approx 0$ for all $\ell \neq k$
- Multi-class logistic regression: Assigns highest probability to class k when z_k is largest

$$z_k = \boldsymbol{w}_k^T \boldsymbol{x} + w_{0k}$$

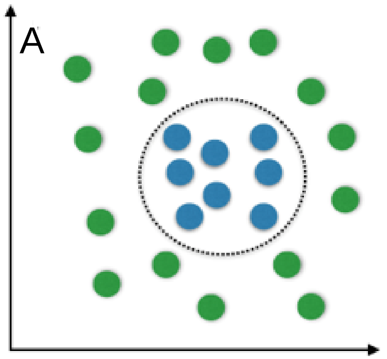
Multi-Class Logistic Regression Decision Regions

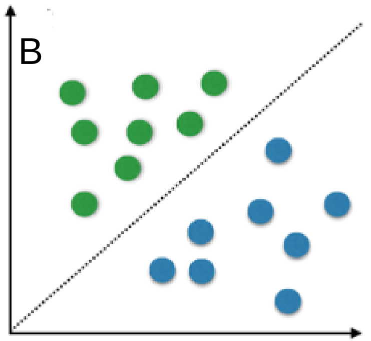
- ❑Each decision region defined by set of hyperplanes
- ❑Intersection of linear constraints
- ❑Sometimes called a polytope



Using Transformed Features

- Enables richer class boundaries
- Example: Fig A is not linearly separable
- But, consider nonlinear features
  - $\phi(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_2^2\,]^T$
- Then we can discriminate classes with linear function (shown in A)
  - $z = [-r^2, 0, 0, 1, 1]\phi(x) = x_1^2 + x_2^2 - r^2$
- Blue when $z \le 0$ and Green when $z > 0$





Learning the Logistic Model Parameters

- Consider general three-part logistic model:
  - Transform to features: $\boldsymbol{x} \mapsto \phi(\boldsymbol{x})$
  - Linear weights: $\boldsymbol{z} = \boldsymbol{W}\phi(\boldsymbol{x})$, $\boldsymbol{W} \in R^{K \times p}$
  - SoftMax: $P(y = k \mid \boldsymbol{x}) = g_k(\boldsymbol{z}) = g_k(\boldsymbol{W}\phi(\boldsymbol{x}))$
- Weight matrix $\boldsymbol{W}$ represents unknown model parameters
- Learning problem:
  - Given training data, $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, N$
  - Learn weight matrix $\boldsymbol{W}$
  - What loss function to minimize?

Logistic Loss Function for Binary Classification

- From previous discussion, we know

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

- Logistic loss function for binary classification is defined as

$$Loss = y_i \ln P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{w}) + (1 - y_i) \ln P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{w})$$

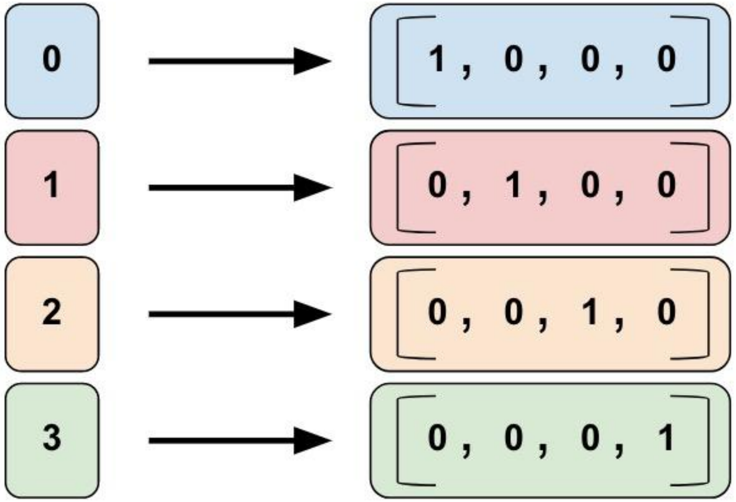
Therefore, $Loss = y_i \ln\left[\frac{e^{z_i}}{(1+e^{z_i})}\right] + (1 - y_i) \ln\left[\frac{1}{1+e^{z_i}}\right]$

One-Hot Log Likelihood for Multi-Class Classification

- Define the "one-hot" vector:

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

- Then, $Loss = \Sigma_{k=1}^{K} r_{ik} \ln P(y_i = k | \boldsymbol{x}_i, \boldsymbol{W})$
  - Sometimes called the cross-entropy

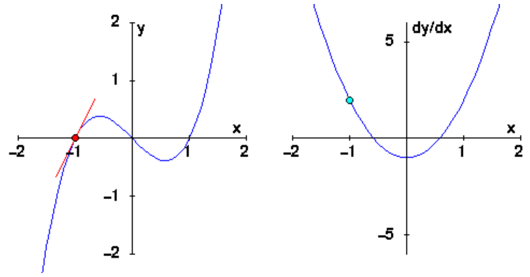


Non-linear Optimization

Motivation:

- ❑ We have been using closed-form solution to solve regression/classification problems
- ❑ Closed form solution is not always the best:
    - ❑ Computation efficiency: operations like inverting a matrix is not efficient
        - ❑ Example: Closed-form solution to linear regression
            - ❑ Calculating the inverse is expensive when there are many variables
    - ❑ For more complex problems, like neural network, a closed form solution is not always available
- ❑ Need an optimizer to find an optimal solution

Introduction to Derivatives

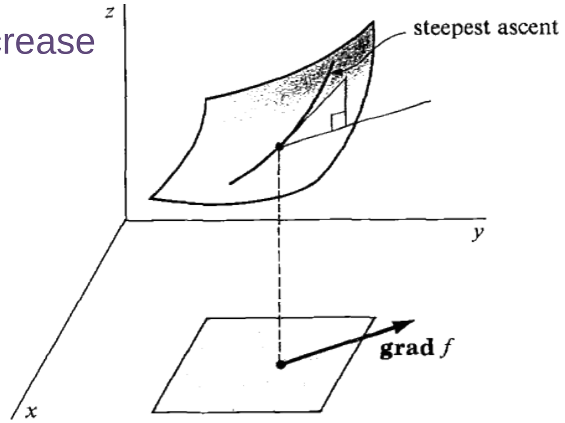


- ❑ The slope of a function tells us the rate of change:
  - ❑ Formula for average rate of change :
  - ❑ Derivative of a function gives us the instantaneous rate of change
  - ❑ For a 1-dimensional function,

Gradient

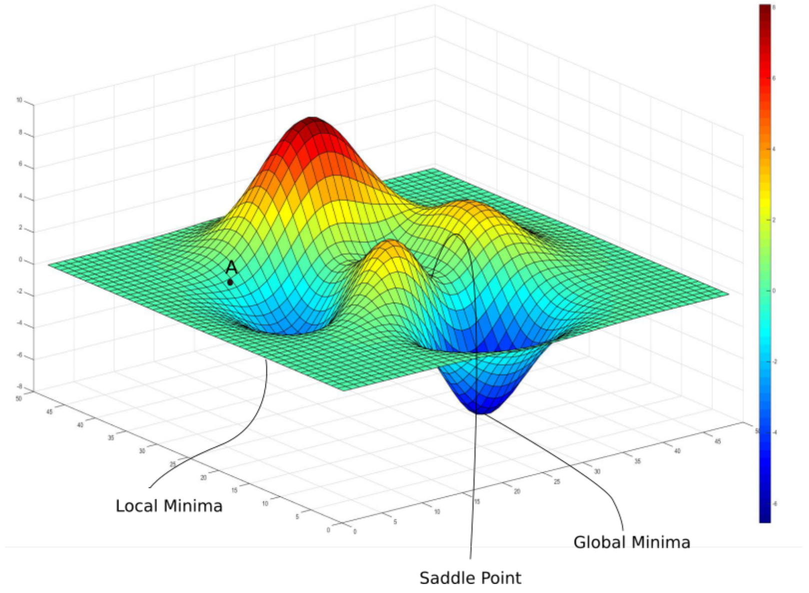
- ❑ In a two dimensional function , we can take derivative in more than one direction
  - ❑ Directional derivative: rate of change of a function in one particular direction

    Example: ,
- ❑ Gradient: A vector that points to the direction of maximum increase



Visualizing Weight Space

- ❑ We may visualize the loss function as surface in a multi-dimensional space
- ❑ Locally, the function may be viewed as a paraboloid
- ❑ There are local minima that we would want to avoid because they are not the optimal solution



Gradient Descent

- ❑ A common non-linear optimization algorithm
- ❑ Key Idea: For each iteration

1. Calculate the gradient of the objective function with respect to the weights

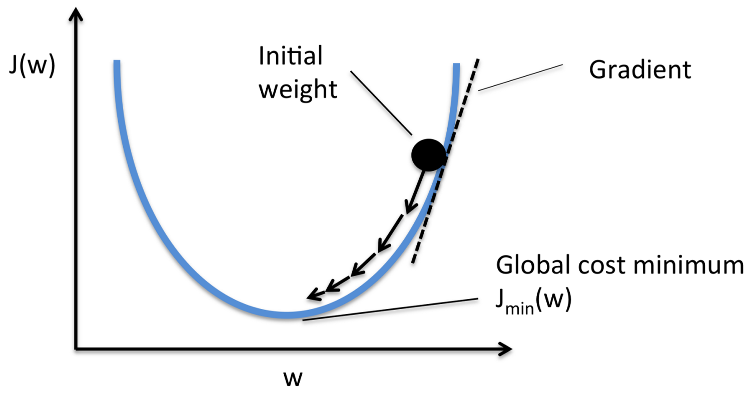
$$\nabla_w f(\mathbf{w}) = \begin{bmatrix} \partial f(\mathbf{w})/\partial w_1 \\ \vdots \\ \partial f(\mathbf{w})/\partial w_N \end{bmatrix}$$

2. Update the weights by taking a step opposite to the gradient

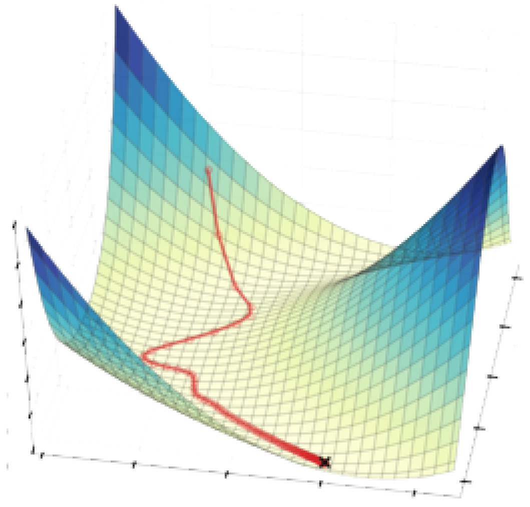
$$w^{k+1} = w^k - \alpha_k \nabla f(w^k)$$

where is called the learning rate: how big of a step you take

Gradient Descent Illustrated



1-Dimensional

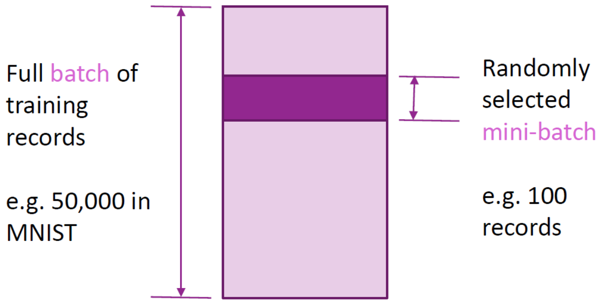


2-Dimensional

NYU

Stochastic Gradient Descent (SGD)

- ❑ A variant of the standard gradient descent
- ❑ In standard gradient descent, gradient is calculated using all training examples
  - ❑ Gradient computation is expensive when data size is very large
- ❑ Stochastic Gradient Descent
  - ❑ In each step:
    1. Select random small "mini-batch"
    2. Evaluate gradient on mini-batch
- ❑ Has the added benefit that:
  - ❑ Manifold is changing: do not get stuck in a local minimum



Errors in Binary Classification

- ❑ There are two types of errors here:
  - ❑ Type I error (False Positive or false alarm): Decide $\hat{y}$=1 when y=0
  - ❑ Type II error (False Negative or missed detection): Decide $\hat{y}$=0 when y=1
- ❑ To keep the errors in check we measure the accuracy of the classifier, with the help of a confusion matrix, True Positive Rate and False Positive Rate.
  - ❑ Accuracy of classifier can be measured by:
    - $TPR = P(\hat{y} = 1|y = 1)$
    - $FPR = P(\hat{y} = 1|y = 0)$
    - Accuracy=$P(\hat{y} = 1|y = 1)$+ $P(\hat{y} = 0|y = 0)$
      - (percentage of correct classification)

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

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

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

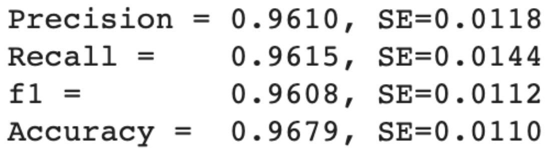
Other Metrics to measure the error rates:

- The most commonly used ones are:
  - Recall/Sensitivity/TPR = 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)
- Implications:
  - Sensitivity $= P(\hat{y} = 1 | y = 1) = TPR$
  - Specificity $= P(\hat{y} = 0 | y = 0) = 1 - FPR$ =True negative rate
  - We need to find a good trade off between Sensitivity and Specificity

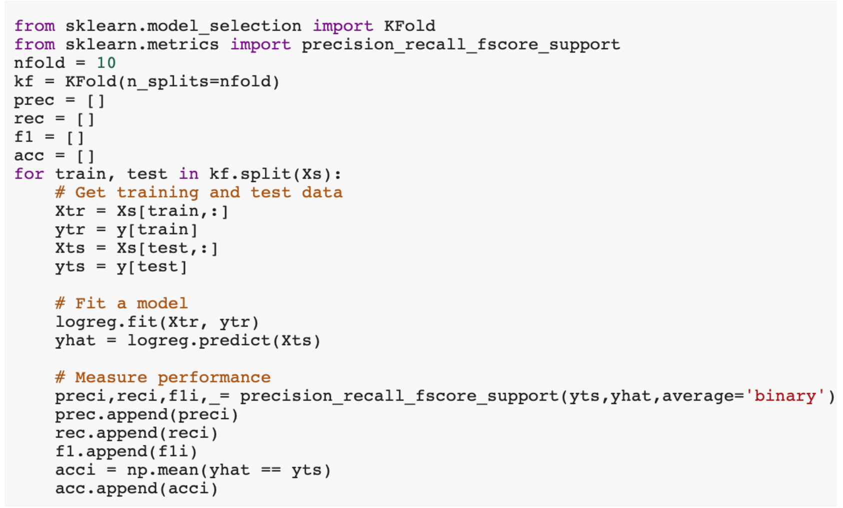
NYU

Breast Cancer Example:

- ❑ Measuring the accuracy on test data
- ❑ Using 10 fold cross-validation
- ❑ Sklearn has built-in functions for CV
- ❑ The following is the output:



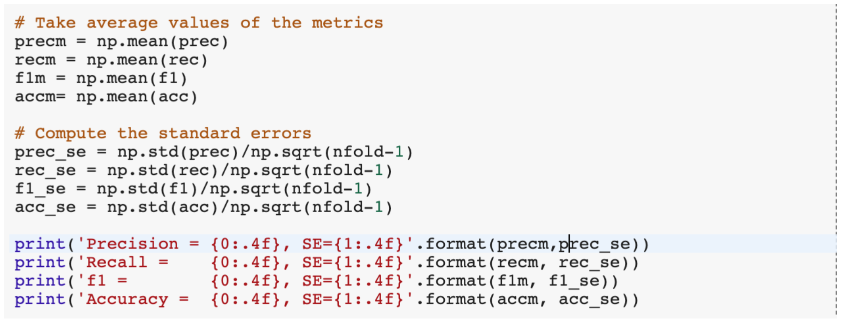
```
Precision =  0.9610, SE=0.0118
Recall =     0.9615, SE=0.0144
f1 =         0.9608, SE=0.0112
Accuracy =   0.9679, SE=0.0110
```



```python
from sklearn.model_selection import KFold
from sklearn.metrics import precision_recall_fscore_support
nfold = 10
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 performance
    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)
```



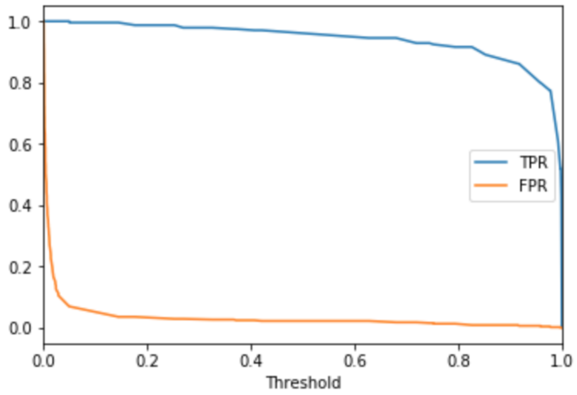
```python
# Take average values of the metrics
precm = np.mean(prec)
recm = np.mean(rec)
f1m = np.mean(f1)
accm= np.mean(acc)

# Compute the standard errors
prec_se = np.std(prec)/np.sqrt(nfold-1)
rec_se = np.std(rec)/np.sqrt(nfold-1)
f1_se = np.std(f1)/np.sqrt(nfold-1)
acc_se = np.std(acc)/np.sqrt(nfold-1)

print('Precision =  {0:.4f}, SE={1:.4f}'.format(precm,prec_se))
print('Recall =     {0:.4f}, SE={1:.4f}'.format(recm, rec_se))
print('f1 =         {0:.4f}, SE={1:.4f}'.format(f1m, f1_se))
print('Accuracy =   {0:.4f}, SE={1:.4f}'.format(accm, acc_se))
```

Hard Decisions:

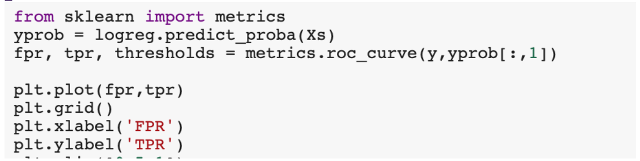
- Logistic classifier outputs a **soft** lab $P(y=1|x) \in [0,1]$
  - $P(y=1|x) \approx 1 \Rightarrow y = 1$ more likely
    $P(y=0|x) \approx 1 \Rightarrow y = 0$ more likely
- We can obtain a **hard label** by **thresholding**:
  Set $\hat{y} = 1 \; if \;\; P(y=1|x) > t$
  - t = Threshold



- 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

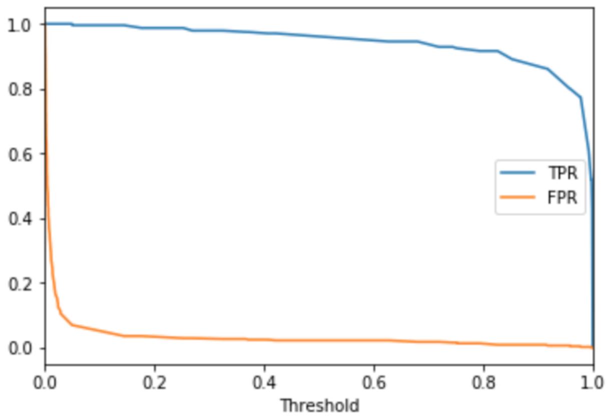
ROC (Receiver Operating Characteristics) Curve

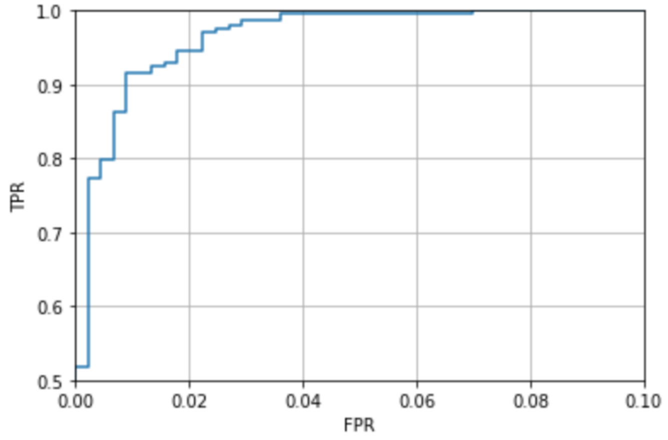
- ❑ Varying threshold obtains a set of classifier
- ❑ Trade off between FPR (1-specificity) and TPR (sensitivity)



```
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')
```





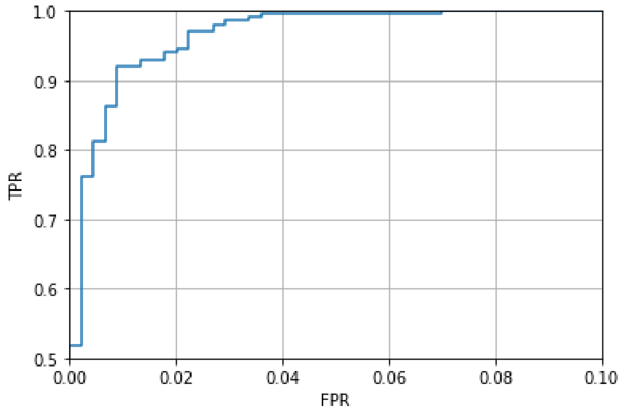
NYU

AUC (Area Under the Curve)

- ❑ We can choose a particular threshold based on the desired trade-off between the TPR and FPR, but 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 TPR.
- ❑ AUC ranges between 0 and 1. 1 being the best and the highest.
- ❑ We should report average AUC over cross validation folds.

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

```
AUC=0.996297
```



Thank You!