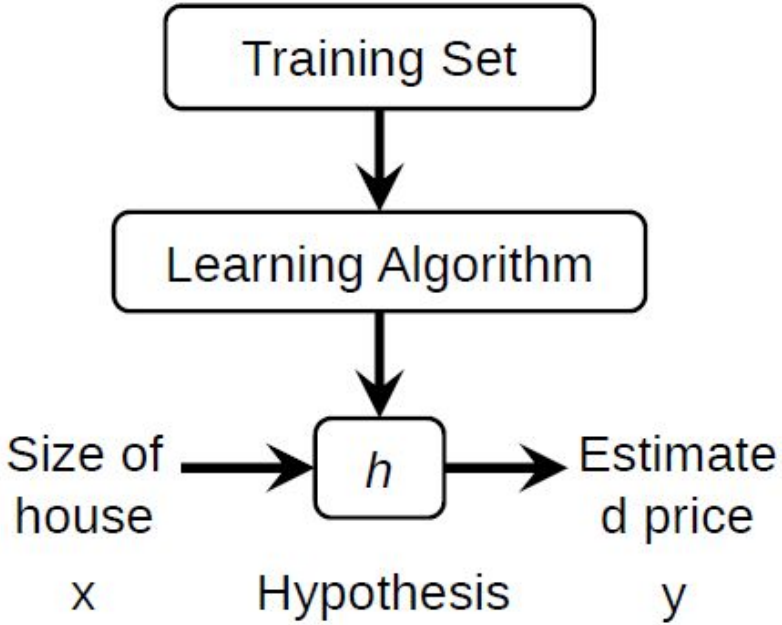
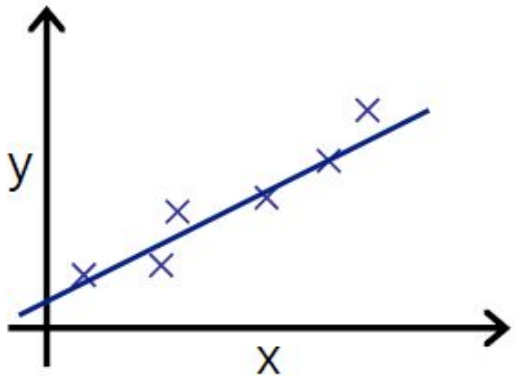
K6312 Information Mining & Analysis

Chen Zhenghua & Zhao Rui

Recap on Linear Regression



How do we represent *h* ?

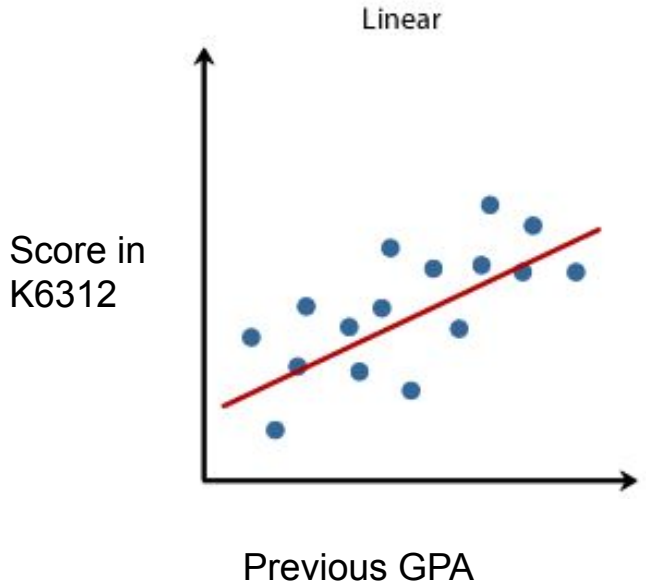


$$h(x) = w_0 + w_1 x$$

Linear regression with one variable.
"**Univariate Linear Regression**"

Continuous Target

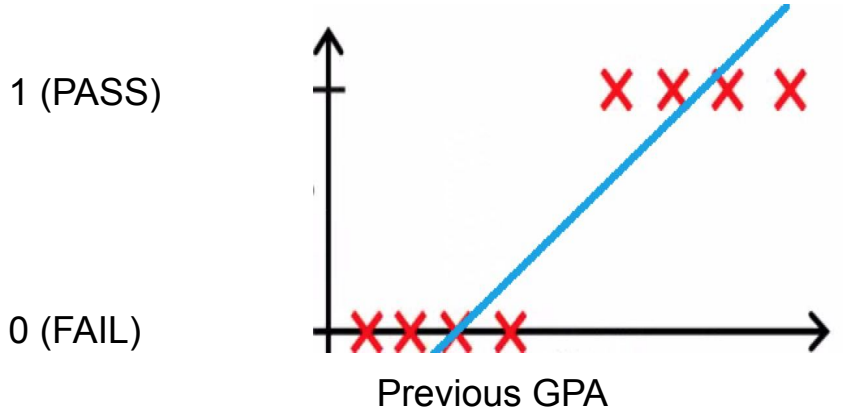
- Let us build an auto-grade algorithms
- Input feature is one scalar: your previous GPA
- Target value is: your score in K6312



Discrete Target

- We only want to predict whether you can pass K6312
- Input feature is one scalar: your previous GPA
- Target value is: a binary value (1: pass, 0: fail)

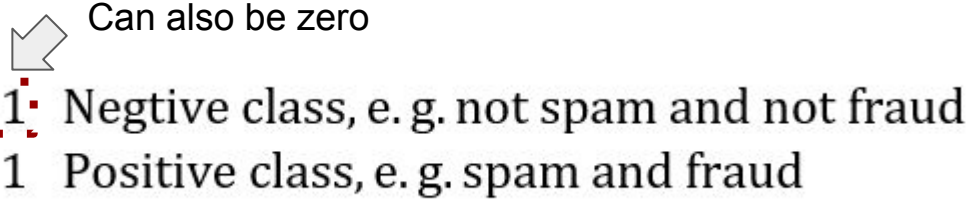
Classification Problem



Classification

Binary Classification

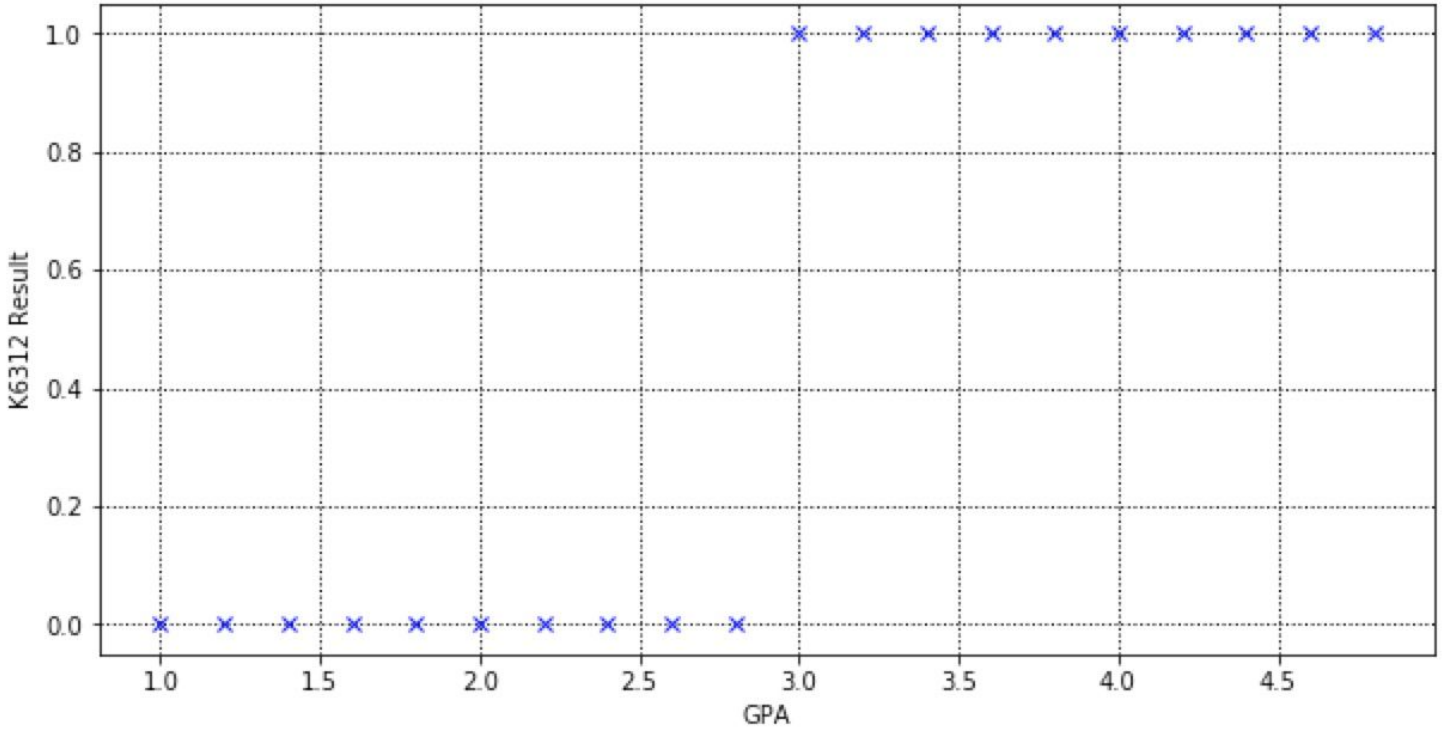
- Email: spam or not spam
- Online Transaction: fraud or not fraud
- ……



Machine Learning is to learn a function from data such that

$$f: X \rightarrow Y$$

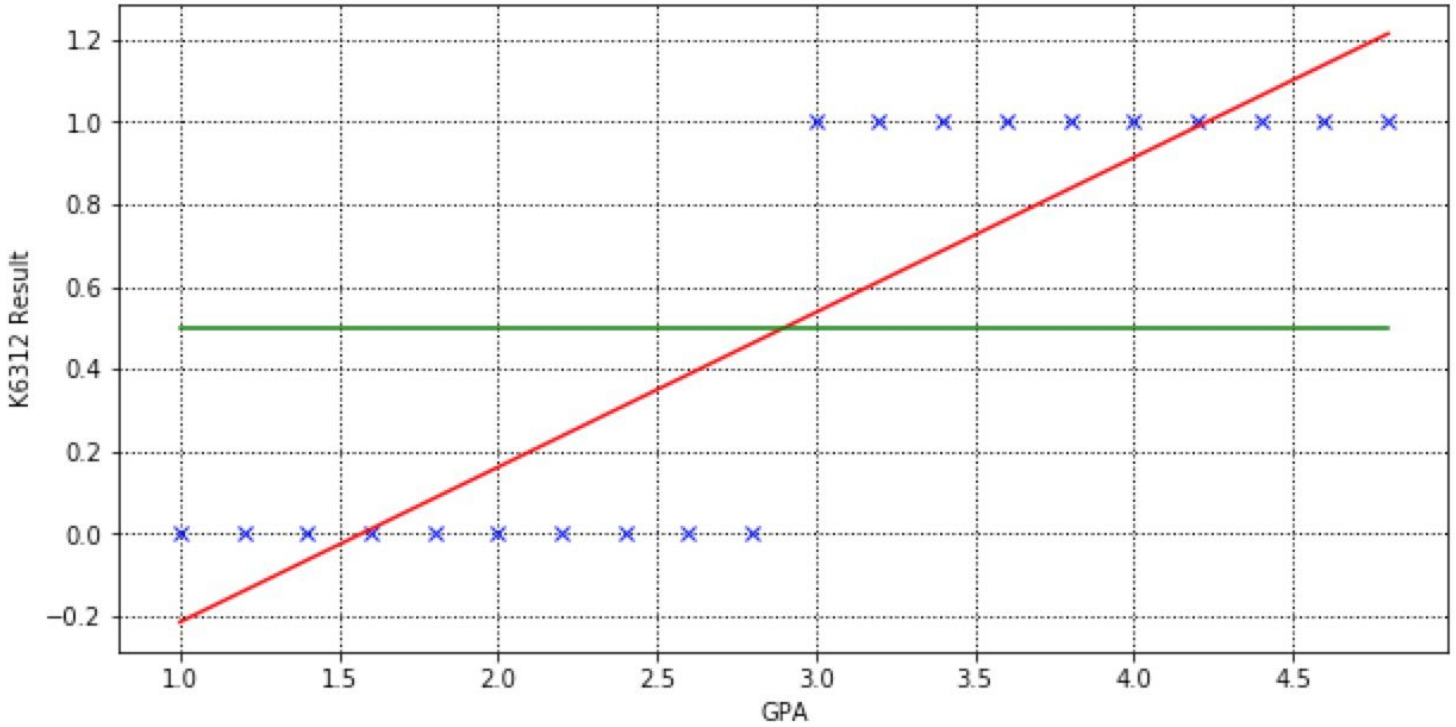
Can we use linear regression for classification?



After fitting,

```
print(lin_regression.coef_)
print(lin_regression.intercept_)
```

```
[0.37593985]
-0.5902255639097744
```



Output Value is continuous

- For classification problem, we want the output value to be probabilistic, which should be in range(0, 1).
- However, the output of linear regression is unbounded

```
print(lin_regression.coef_)
print(lin_regression.intercept_)
```

```
[0.37593985]
-0.5902255639097744
```

y=0.3759*x-0.59

When x = 5, y=1.289

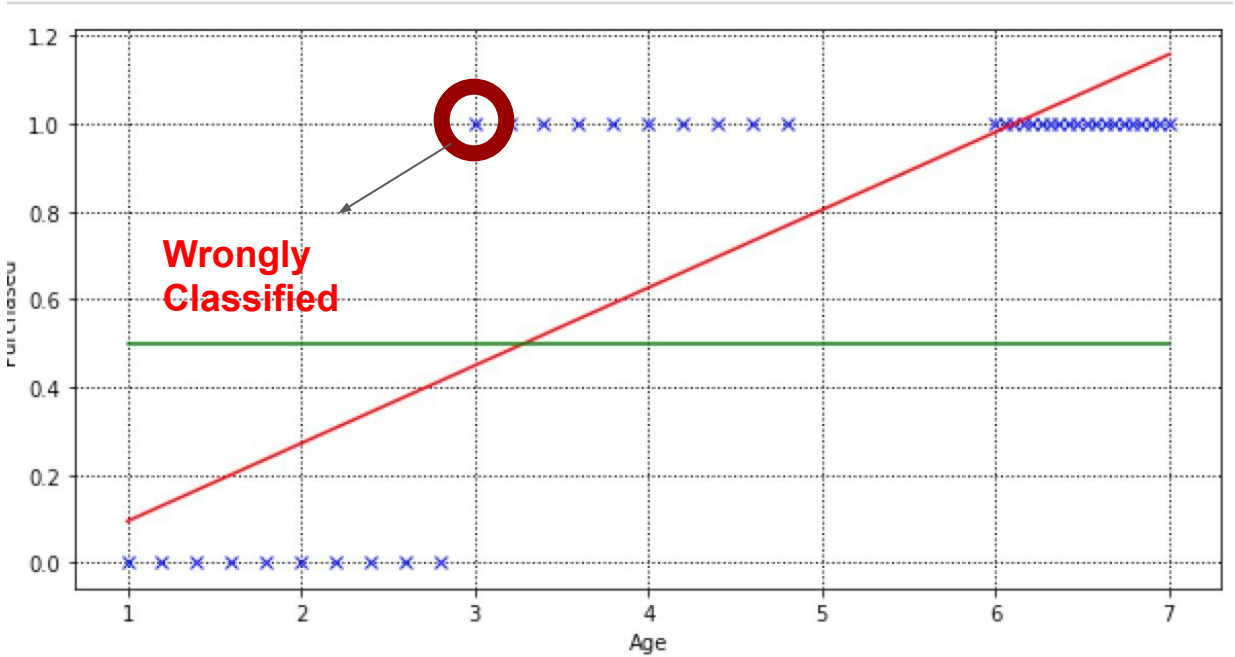
When x = 4.6, y=1.038

When x = 1.2, y=-0.13

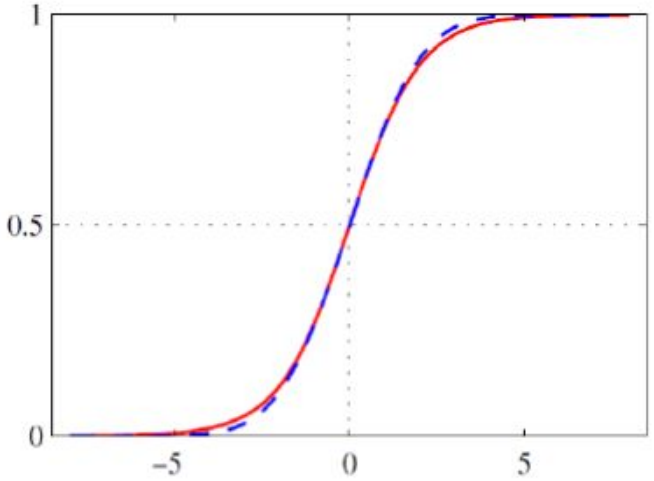
What we want is that the proba. scores of the first two cases is close to 1 and the last case is close to 0.

Imbalanced data

- Let us add 20 students whose GPA are in the range(6, 7) and pass the K6312



Logistic Function



$$\sigma(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t}$$

t : (-infinite, +infinite)

$\sigma(t)$: probabilistic score from 0 to 1

Pierre François Verhulst named the function as logistic function in 1845

Linear Regression

$$f(x) = w * x + b$$

$$\sigma(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t}$$

Logistic Regression

$$f(x) = \frac{1}{1+e^{-(w*x+b)}}$$

Logistic Regression

Logistic Regression

- Uses logistic function to model binary target
- Model the distribution of *p(y = 1|**x**)* given ***x***
- The exact parametric formulation is:

$$p(y=1|x) = \frac{1}{e^{(-wx+b)}+1} = \frac{e^{(wx+b)}}{e^{(wx+b)}+1}$$

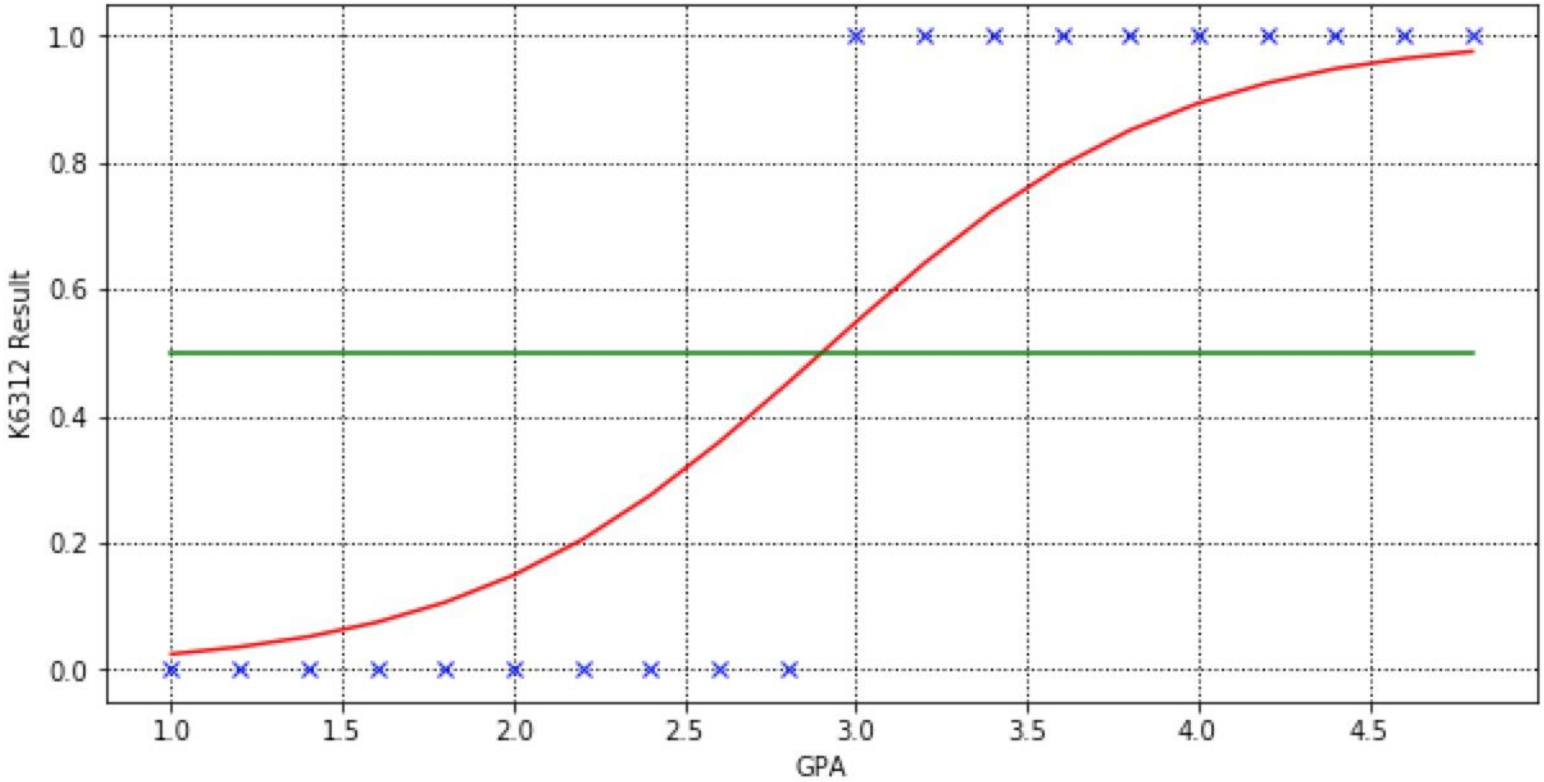
$$p(y=0|x) = 1 - \frac{1}{e^{(-wx+b)}+1} = \frac{1}{e^{(wx+b)}+1}$$

- Let us check the performance of Logistic Regression on K6312 auto-grade system

After fitting

```
print(log_regression.coef_)
print(log_regression.intercept_)
```

```
[[1.93582432]]
[-5.61388646]
```



Output Value is Prob.score

```
print(log_regression.coef_)
print(log_regression.intercept_)
```

```
[[1.93582432]]
[-5.61388646]
```

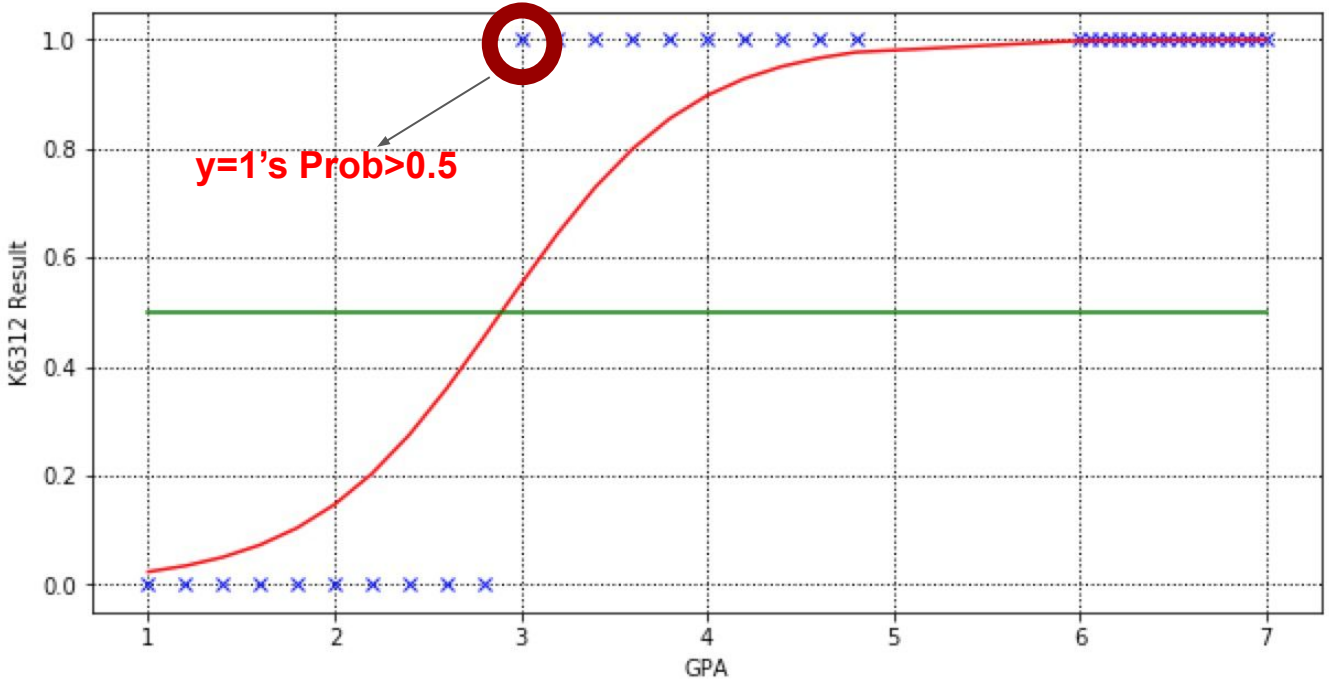
$$p(y=1|x) = \frac{1}{e^{(-wx+b)}+1} = \frac{e^{(wx+b)}}{e^{(wx+b)}+1}$$

p(y=1|x)=e^t/(1+e^t)
p(y=0|x)=1/(1+e^t)
t=1.93*x-5.6

	prob(y=0)	prob(y=1)
x = 5	[[0.01686949	0.98313051]
x = 4.6	[0.03588451	0.96411549]
x = 1.2	[0.96411521	0.03588479]]

Imbalanced data

- Let us add 20 students whose GPA are in the range(6, 7) and pass the K6312



How to learn parameters

- Fitting the data
- In the python code, it is simple

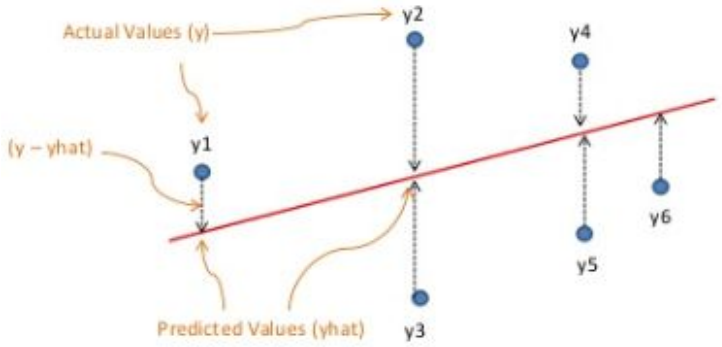
Examples

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
```

- Actually, it is an optimization problem (in math perspective)

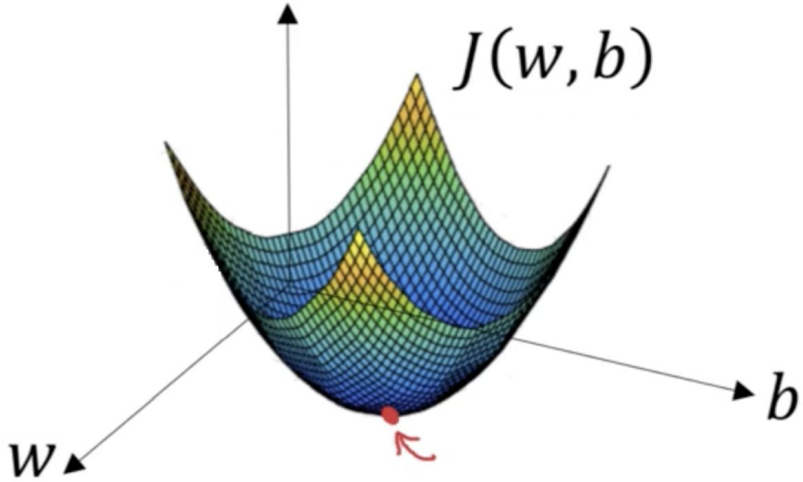
Optimization

- Fitting the data -> define a loss function, which reflects the fitness of the different model parameters over the parameters
- Optimization is the process to search the minimum point



For Linear Regression

https://www.slideshare.net/AndrewFerlitsch/ml-simple-linear-regression



Math Warning

Entropy Loss

- For each single data point: $\tilde{y} = p(y=1|x) = \frac{1}{e^{(-wx+b)}+1} = \frac{e^{(wx+b)}}{e^{(wx+b)}+1}$

$$Loss(y, \tilde{y}) = -[ylog\tilde{y} + (1-y)log(1-\tilde{y})]$$

- To understand this loss function, compute the loss values for these following cases:
  - If y=1, predict prob of y=1 is 0.9,
  - If y=0, predict prob of y=1 is 0.2,
  - If y=1, predict prob of y=1 is 0.2,
  - If y=0, predict prob of y=1 is 0.9,

Whether the model prediction is correct or not?

Entropy Loss

- For each single data point: $\tilde{y} = p(y = 1|x) = \frac{1}{e^{(-wx+b)}+1} = \frac{e^{(wx+b)}}{e^{(wx+b)}+1}$

$$Loss(y, \tilde{y}) = -[ylog\tilde{y} + (1 - y)log(1 - \tilde{y})]$$

- To understand this loss function, compute the loss values for these following cases:
  - **If y=1, predict prob of y=1 is 0.9, -log0.9=0.1**
  - **If y=0, predict prob of y=1 is 0.2, -log0.8=0.22**

  Good Fitness, Low Loss

  - **If y=1, predict prob of y=1 is 0.2, -log0.2=1.6**
  - **If y=0, predict prob of y=1 is 0.9, -log0.1=2.3**

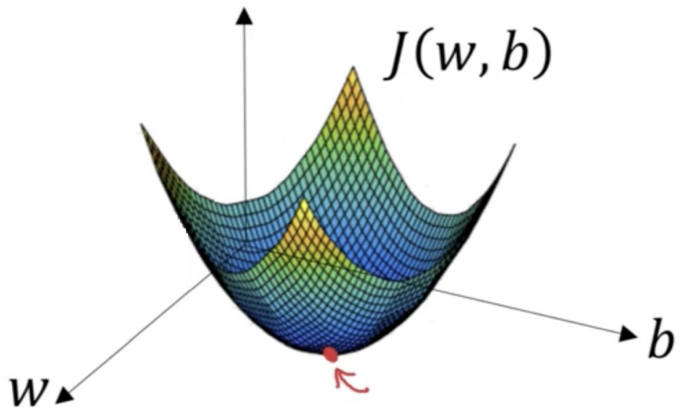
  Bad Fitness, High Loss

Optimization is to reduce Loss

- For training data of m data points (x, y), the loss is the function of model parameters

$$J(w,b) = \frac{\sum_{i=1}^{m} Loss(\tilde{y}^i, y^i)}{m}$$

$$\tilde{y} = p(y=1|x) = \frac{1}{e^{(-wx+b)}+1} = \frac{e^{(wx+b)}}{e^{(wx+b)}+1}$$



Gradient Descent Algorithm

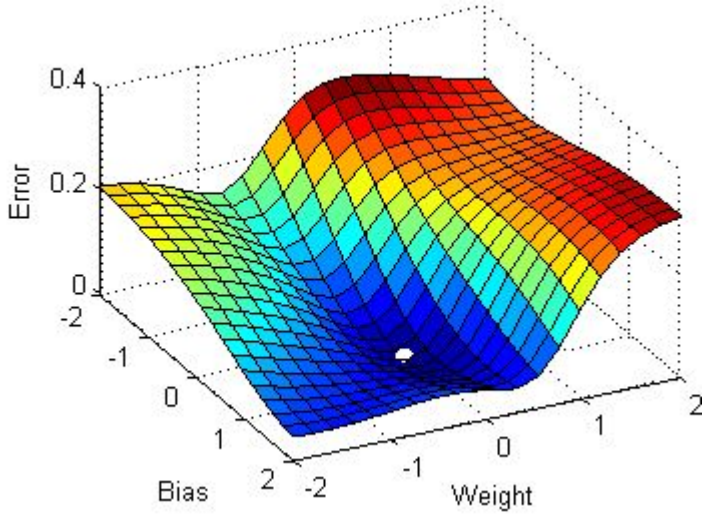
Gradient for loss function f for $\mathbf{x}_n$, which computed by BP algorithm

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \nabla J(\mathbf{w})$$

New Parameters Guess

Current Parameters Guess

Learning Rate



Like hiking down a mountain

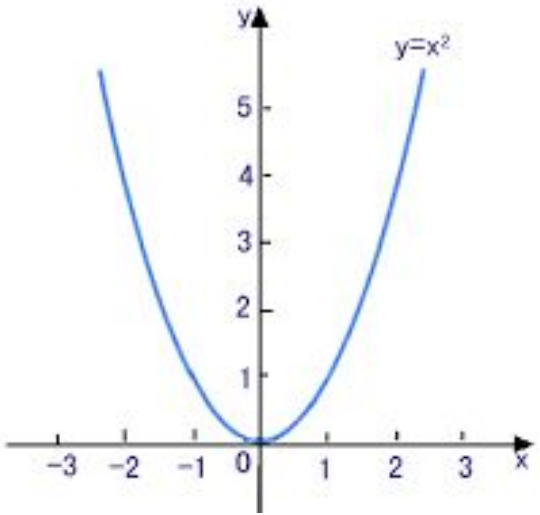
Univariate Gradient Descent

- Assume the model parameter is \theta, the cost function is:

$$J(\theta) = \theta^2$$

$$\bigtriangledown J(\theta) = 2 * \theta$$

$$\theta_{n+1} = \theta_n - \alpha(2 * \theta_n)$$



- The initial guess is 5 and **the learning rate is set to 0.4**
  - Then, we guess: 5 - 0.4*(2*5)=1
  - Then, 1-0.4*(2*1)=0.2
  - Then, 0.2-0.4*(2*0.2)=0.04

Minimum Point: 0

Univariate Gradient Descent

- The initial guess is 5, compute the estimation of the parameter \theta in the first three iterations
  - Large learning rate: 10
  - Small learning rate: 0.01

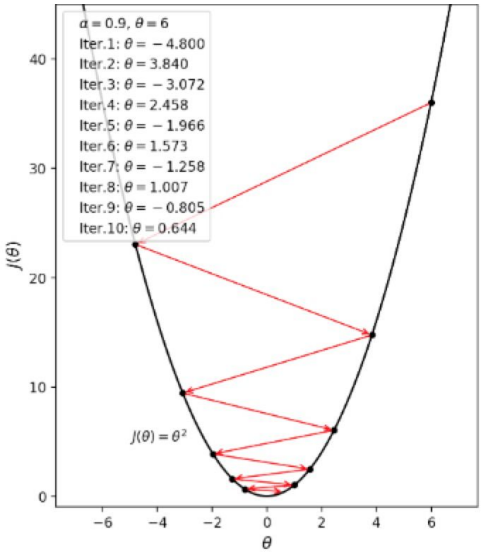
$$J(\theta) = \theta^2$$

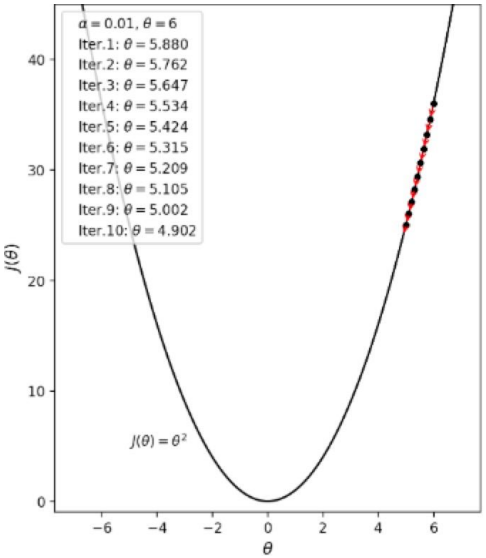
$$\bigtriangledown J(\theta) = 2 * \theta$$

$$\theta_{n+1} = \theta_n - \alpha(2 * \theta_n)$$

Learning Rate

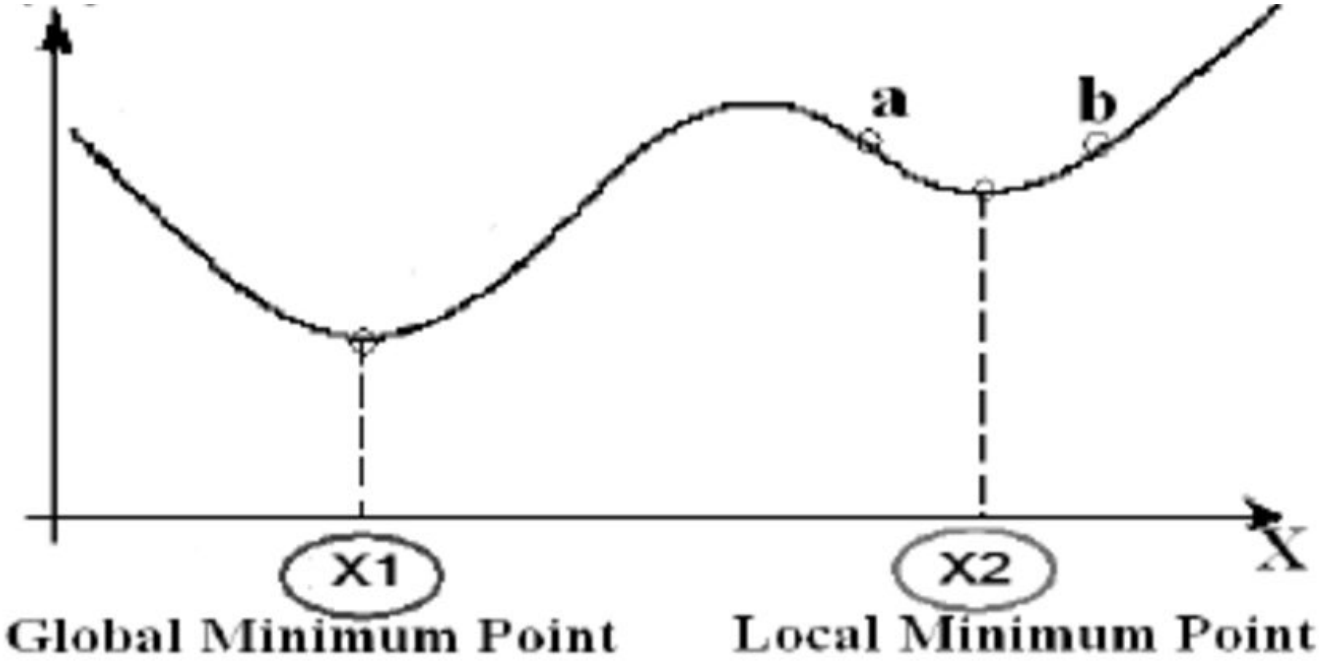
- Too large, too slow
- Too small, too oscillation





Local Minimum

- If our initial guess is a or b,



Decision boundary of Logistic Regression

- Decision is made by comparing the probabilities

$$p(y=1|\mathbf{x}) > p(y=-1|\mathbf{x}) \Leftrightarrow \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} > 1$$

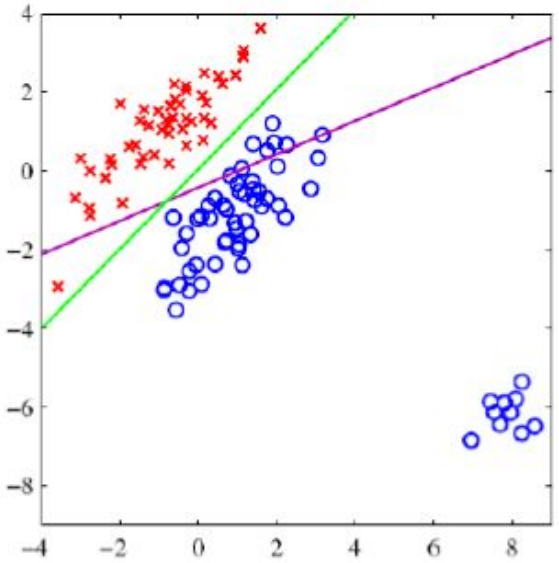
- Take the logarithm

$$\ln \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} = \mathbf{w}^\top \mathbf{x} + b \rightarrow \mathbf{w}^\top \mathbf{x}$$

- Decision boundary is linear

$$\mathbf{w}^\top \mathbf{x} + b = 0$$

$$y = \begin{cases} +1 & \text{if } \mathbf{w}^\top \mathbf{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}$$



* The threshold is tunable

We can have multiple w

- For simplicity, we only have one feature x therefore only one w and bias b in the example.
- In practice, each data sample is represented by a n-dimensional vector and the logistic regression model has n weights and one bias b.
- For Gradient descent algorithm, partial derivative of each parameter is used

output: σ(-1.2*(1) + 1.4*(1) + 2.2*(1) + 0.6*(1) + 0.2)

LR’s Application

- Classifier whether a user will click the advertisement
- Classifier whether your PR application can be approved or not
- Classifier whether these two kinds of people will fall in love or not

Evaluation

How to do we evaluate the model performance?

i.e., how to quantify the matching degree between the ground truth y and the predicted labels y^.

Evaluation of Classification Problems

- Confusion Matrix

	Predicted Positive	Predicted Negative
Positive Label	**TP** True Positive	**FN** False Negative
Negative Label	**FP** False Positive	**TN** True Negative

- Accuracy: How accurate is the prediction?

$$\frac{\text{Correct Prediction}}{\text{Total \#-of-Samples}} = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision and Recall

	Predicted Positive	Predicted Negative
Positive Label	TP	FN
Negative Label	FP	TN

- Precision $= \frac{TP}{TP + FP}$
  - how accurate the positive prediction is?

- Recall $= \frac{TP}{TP + FN}$
  - how many positive cases are detected?

Example: K6312 Auto-grade

	Predicted Positive	Predicted Negative
Positive Label	27	4
Negative Label	1	18

- Accuracy = (27 + 18) / (27 + 1 + 18 + 4) = 0.9
- Precision = 27 / (27 + 1) = 0.964
- Recall = 27/ (27 + 4) = 0.871

- Can we do better?

Precision vs Recall

- Case 1: Accuracy is high, but recall is low.
  - Examples?

Accuracy: TP+TN/(TP+FP+TN+FN)
Precision: TP/(TP+FP)
Recall: TP/(TP+FN)

- Case 2: Accuracy is high, but precision is low.
  - Examples?

Precision vs Recall

- Case 1: Accuracy is high, but recall is low.
  - Examples?

Accuracy: TP+TN/(TP+FP+TN+FN)
Precision: TP/(TP+FP)
Recall: TP/(TP+FN)

- Case 2: Accuracy is high, but precision is low.
  - Examples?

Toy Case

```
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
y_true = [0,0,1,1,1,0,0,1]
```

```
y_pred_prob = [0.45,0.6,0.52,0.55,0.75,0.23,0.27, 0.8]
```

Low Recall

a high threshold of 0.7

```
y_pred = [1 if x>0.7 else 0 for x in y_pred_prob]
```

```
y_true
```

```
[0, 0, 1, 1, 1, 0, 0, 1]
```

```
y_pred
```

```
[0, 0, 0, 0, 1, 0, 0, 1]
```

```
print("accuracy:{}".format(accuracy_score(y_true, y_pred)))
print("precision:{}".format(precision_score(y_true, y_pred)))
print("recall:{}".format(recall_score(y_true, y_pred)))
```

```
accuracy:0.75
precision:0.6666666666666666
recall:1.0
```

Low Precision

a low threshold of 0.3

```
y_pred = [1 if x>0.3 else 0 for x in y_pred_prob]
```

```
y_true
```

```
[0, 0, 1, 1, 1, 0, 0, 1]
```

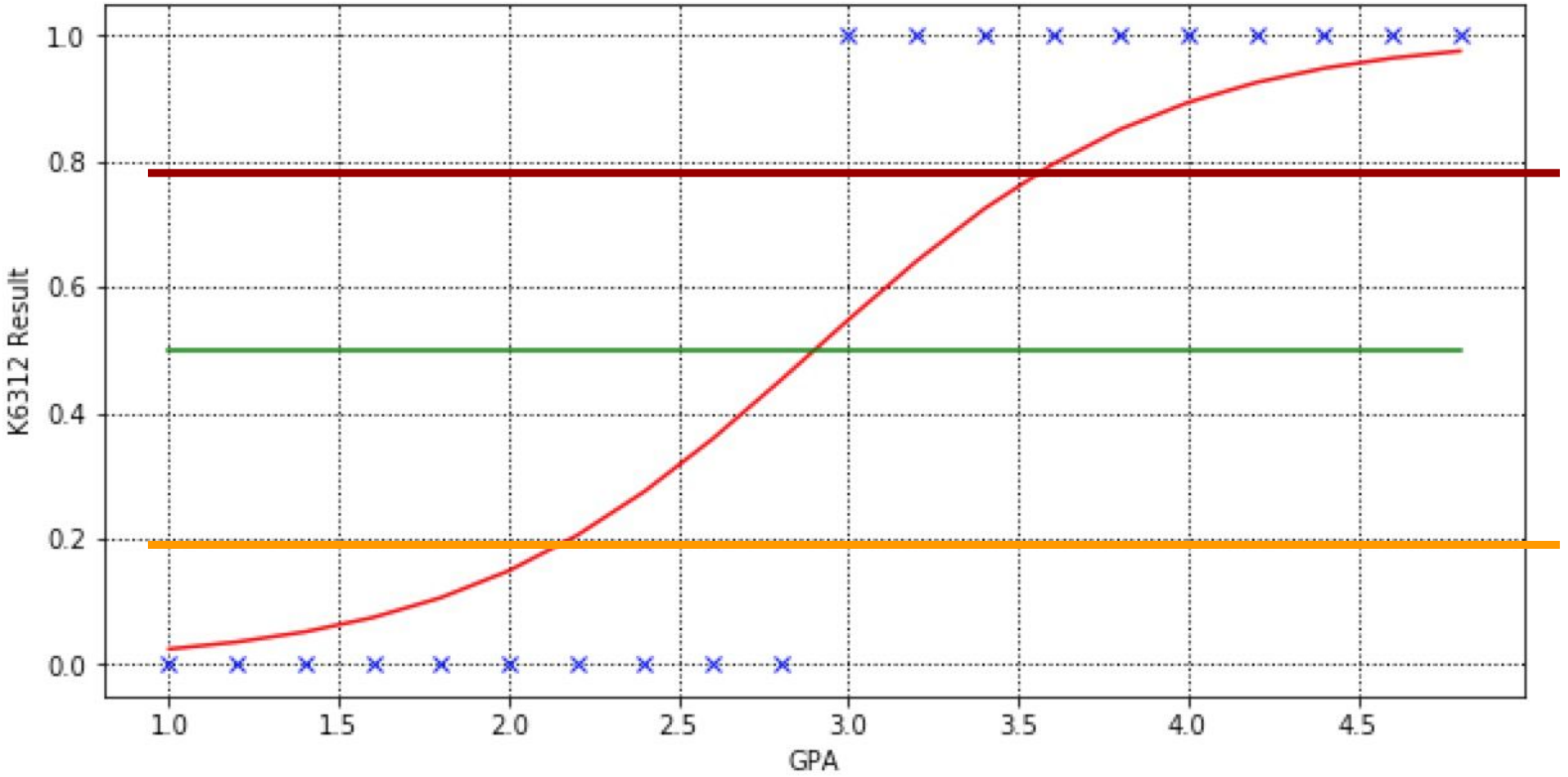
```
y_pred
```

```
[1, 1, 1, 1, 1, 0, 0, 1]
```

```
print("accuracy:{}".format(accuracy_score(y_true, y_pred)))
print("precision:{}".format(precision_score(y_true, y_pred)))
print("recall:{}".format(recall_score(y_true, y_pred)))
```

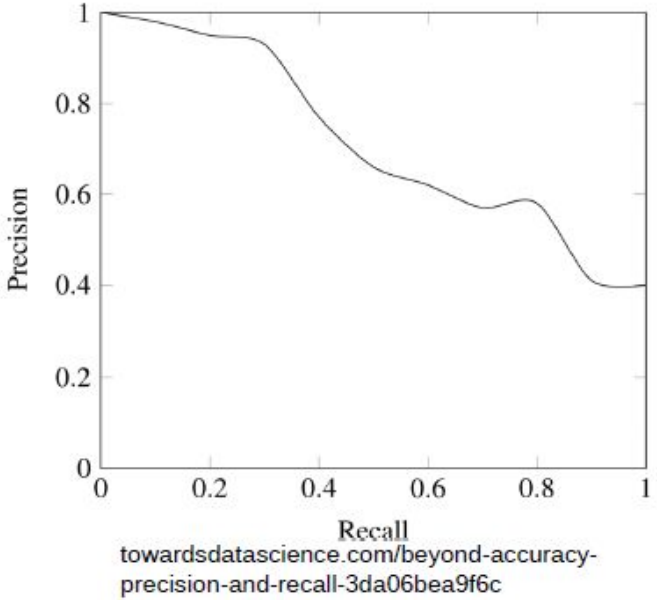
```
accuracy:0.75
precision:0.6666666666666666
recall:1.0
```

Precision vs Recall



Precision v.s. Recall

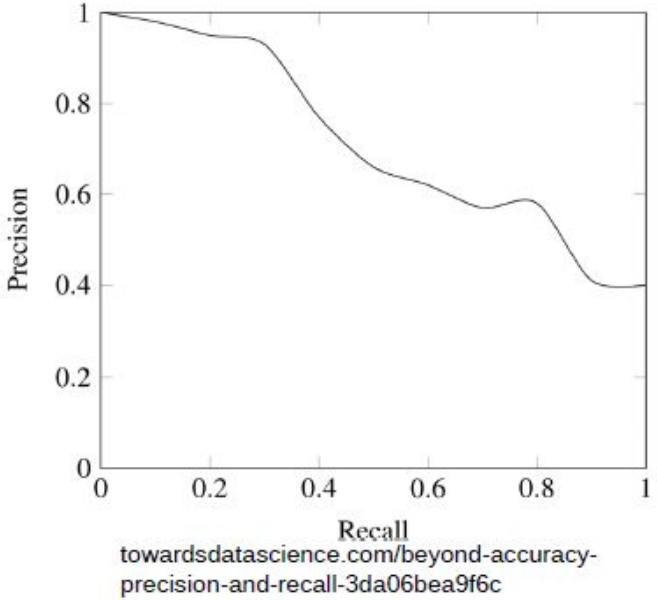
- Under non-trivial situation, precision and recall cannot be optimized at the same time
  - Which one to optimize depends on use cases



towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

F1 Score Vs Accuracy

- Accuracy does not perform well for imbalanced data sets
  - Assume we have 100 transaction, 90 are non-fraud cases and 10 fraud ones
  - High accuracy can be achieved by classifying every transaction as non-fraud
- Precision and Recall can give more insights
- F1 Score conveys the balance between the precision and recall.



towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

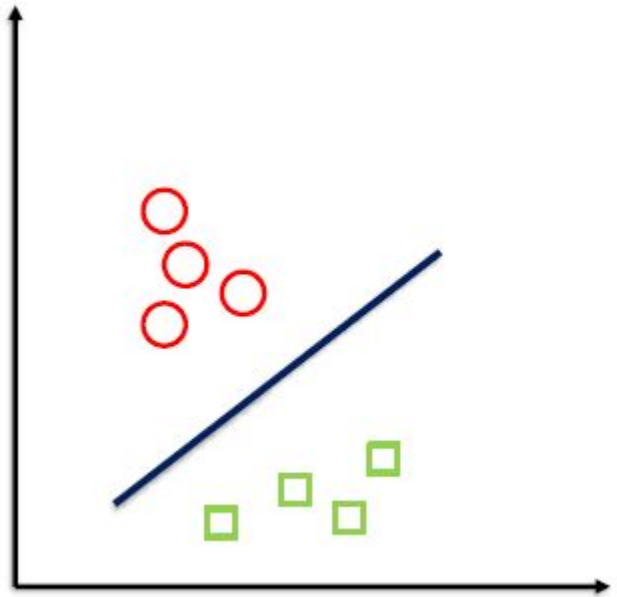
Multiclass Classification

Multiclass Classification

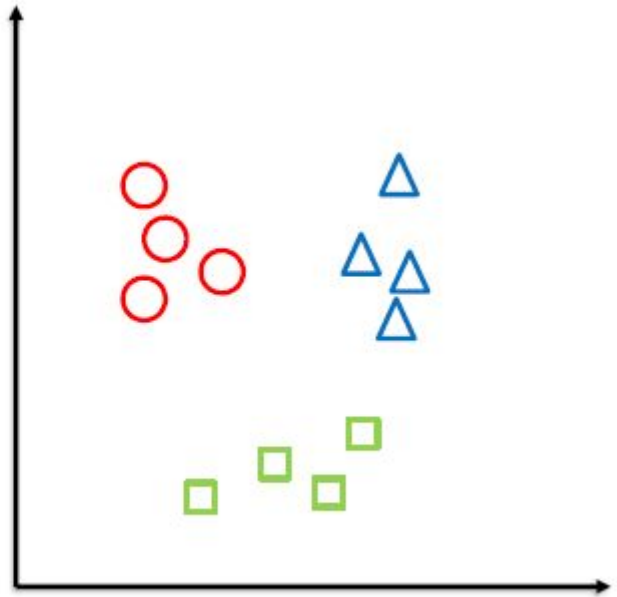
- Weather: Cloudy, Rain, Snow, ...
- Fruit: Apple, Orange, Peach, ...
- Email tagging: Work, Ad, Friends, …

Multiclass Classification

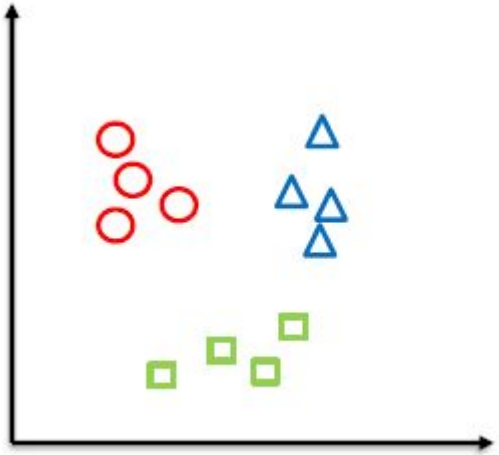
Binary classification

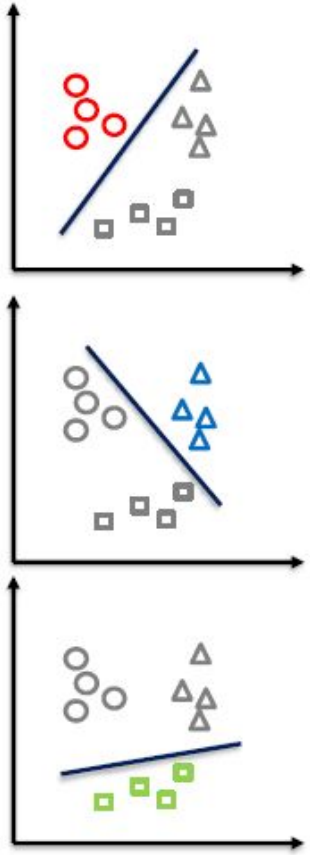


Multiclass classification



One-v.s.-All (One-v.s-Rest)



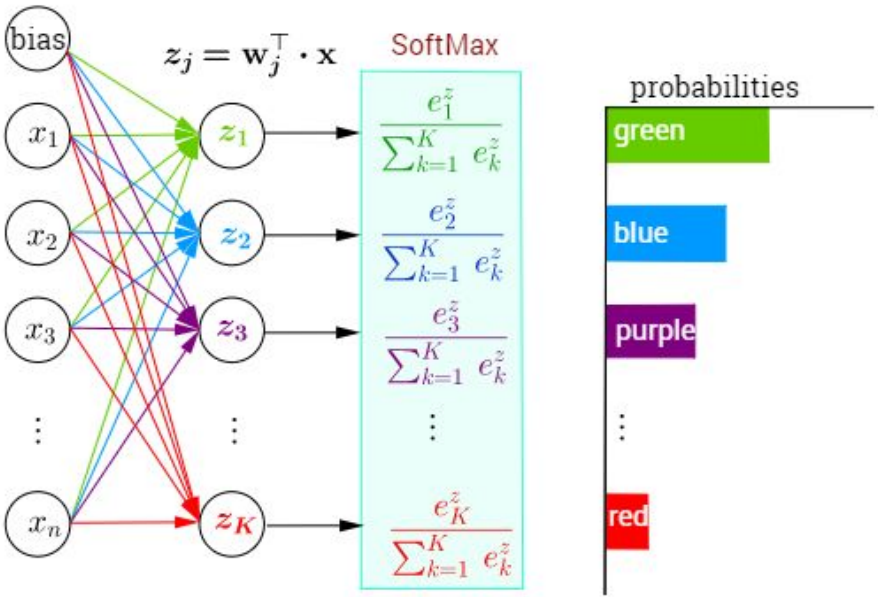


Train a LR classifier $p_i(y = 1|\boldsymbol{x})$ for each class i to predict the probability of $y = i$

$$i^* = \arg\max_i p_i(y = 1|\boldsymbol{x})$$

Softmax Classifier

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



https://stats.stackexchange.com/questions/265905/derivative-of-softmax-with-respect-to-weights