SWE404/DMT413
BIG DATA ANALYTICS

Lecture 8: Classification and Regression Algorithms I

Lecturer: Dr. Yang Lu

Email: luyang@xmu.edu.my

Office: A1-432

Office hour: 2pm-4pm Mon & Thur

Outlines

- Linear Regression
- Logistic Regression
- Neural Networks
- Support Vector Machines
- Machine Learning Related Issues

LINEAR REGRESSION

Data Representation

- For a given dataset, we usually use x to represent the features and y to represent the label. For the ith sample:

$$\boldsymbol{x}_i = \left[x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \dots, x_d^{(i)}\right]^T \in \mathbb{R}^d$$
$$y_i \in \mathbb{R}$$

- A dataset can be represented as:

$$X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \dots, \boldsymbol{x}_n] \in \mathbb{R}^{n \times d}$$
$$\boldsymbol{y} = [y_1, y_2, y_3, \dots, y_n] \in \mathbb{R}^n$$

- $\mathbb{R}$ is the domain of real number, d is the feature dimension and n is the number of samples.
- We use bold font to represent vector, and uppercase letter to represent matrix.
  - x_i is the ith feature in $\boldsymbol{x}$, while $\boldsymbol{x}_i$ is the ith sample in X.

Linear Regression

- Linear regression model can be represented by

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$
$$= w_1 x_1 + w_2 x_2 + \cdots + w_d x_3 + b$$

- $\boldsymbol{w}$ is called the model *weights* or *coefficients*, and b is called the *bias* or *intercept*. Together they are called the model *parameters*.
- The goal of linear regression is to find w and b such that the following *cost function* (aka *loss function*) is minimized:

$$J = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

- This cost function is also known as the Mean Squared Error (MSE) function.

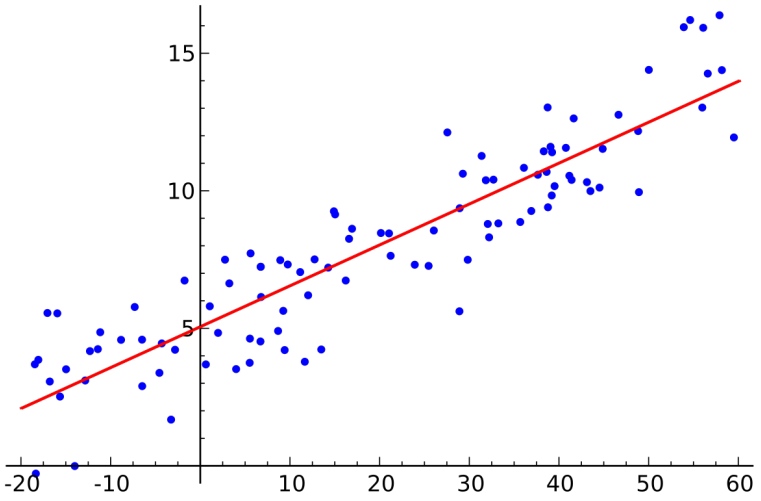


Image source: https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a

Gradient Descend

- The *gradient* vector is orthogonal to the tangent of a plane towards the greater value.
- Thus, the direction of negative gradient heads to the local minimum.
- We can update our model parameter by iteratively adding the negative gradient.

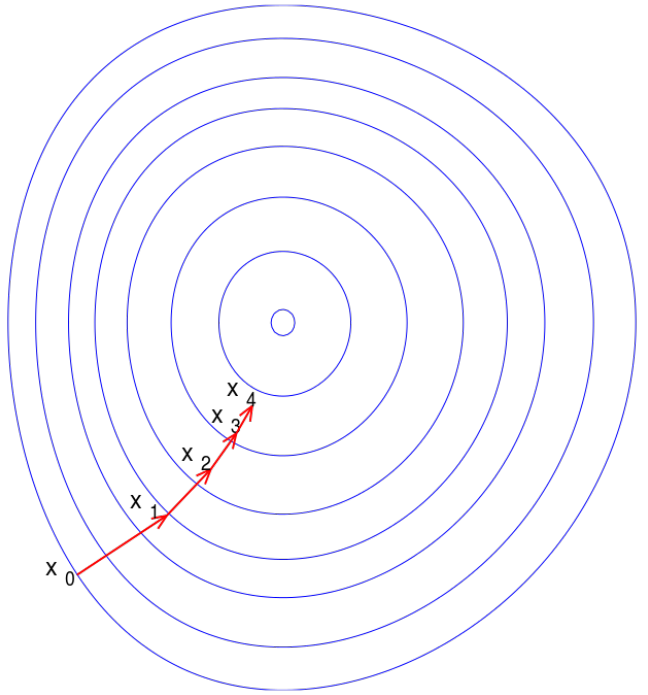


Image source: https://en.wikipedia.org/wiki/Gradient_descent

Gradient Descent

- To solve this minimization problem, we calculate its partial derivatives:

$$\frac{\partial J}{\partial w_i} = \frac{2}{n}\sum_{i=1}^{n}(f(x_i) - y_i)x_i$$

$$\frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(f(x_i) - y_i)$$

- Putting partial derivatives together in a vector is the gradient $\nabla J(\boldsymbol{w})$.
- Thus, the model weights can be iteratively updated by:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla J(\boldsymbol{w})$$

$$b \leftarrow b - \eta \frac{\partial J}{\partial b}$$

The cost function is *convex* such that gradient descent is able to find the global minimum

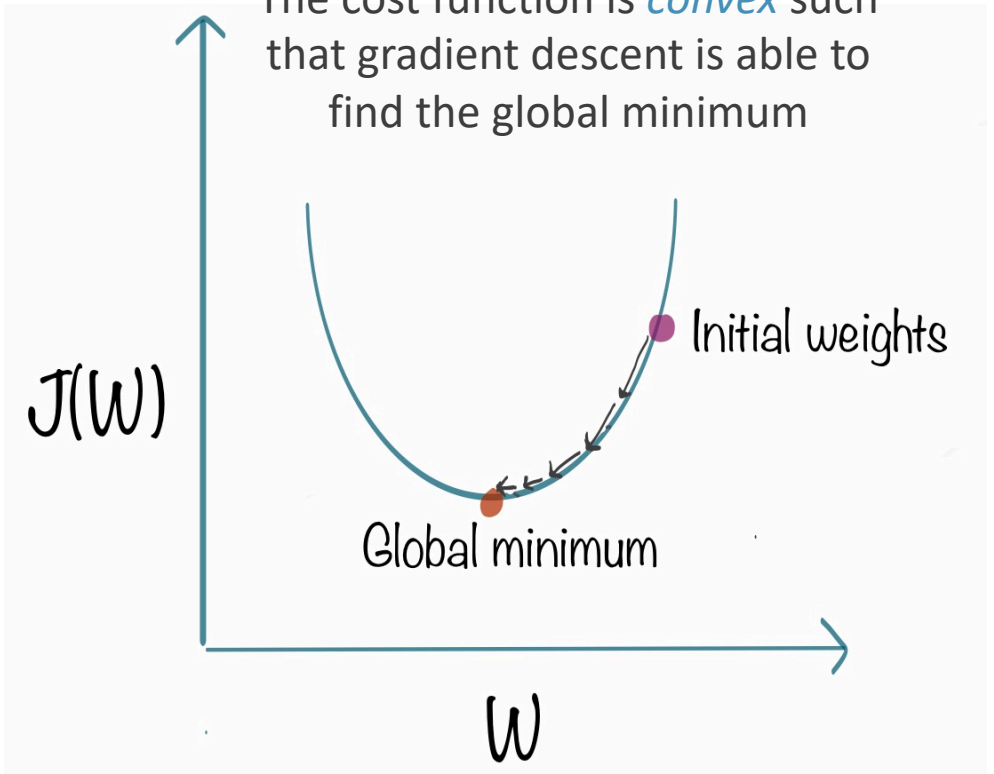
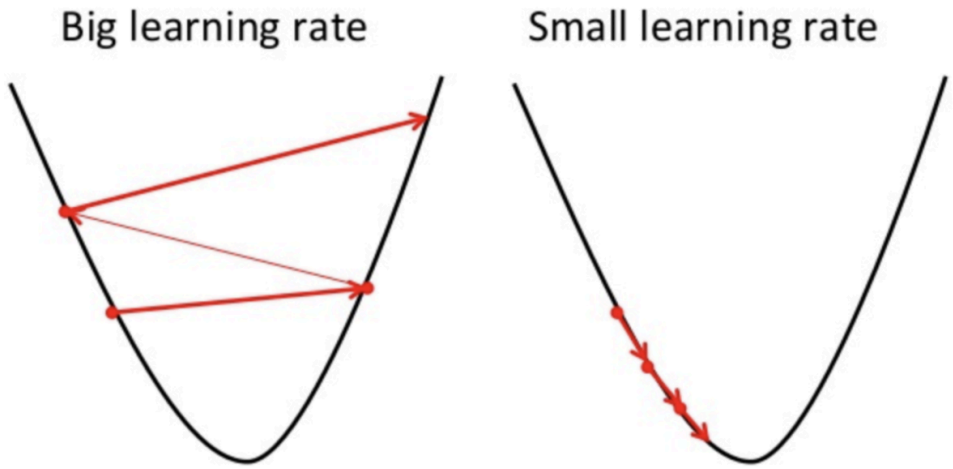


Image source: https://www.kdnuggets.com/2018/06/intuitive-introduction-gradient-descent.html

Learning Rate

- In the above updating formula, The size of these steps η is called the *learning rate*.
  - With a high learning rate, we can go with large step, but we risk overshooting the lowest point and resulting in non-convergence.
  - With a very low learning rate, we can confidently move in the right direction, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom.
- One strategy is to decrease the learning rate gradually on iteration.



Advantages and Disadvantages

- Advantages:
  - The modeling speed is fast, does not require very complicated calculations, and runs fast when the amount of data is large.
  - The understanding and interpretation of each variable can be given according to the model weight.
- Disadvantages:
  - Non-linear data cannot be well fitted. So you need to first determine whether the variables are linear. In real application, the target is seldomly linear with the features.

MLlib API

class `pyspark.ml.regression.`**`LinearRegression`**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, standardization=True, solver='auto', weightCol=None, aggregationDepth=2, loss='squaredError', epsilon=1.35*) [source]

- Commonly used hyperparameters:
  - **maxIter:** max number of iterations (>= 0).
  - **tol:** the convergence tolerance for iterative algorithms (>= 0).
  - **regParam:** regularization parameter (>= 0).
  - **elasticNetParam:** the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty.

Source: https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.regression.LinearRegression

LIBSVM Data Format

- LIBSVM data format is one of the most commonly used data format for machine learning.
  - label 1:feature_1 2:feature_2 …

```
training = spark.read.format("libsvm").load("sample_linear_regression_data.txt")
training.collect()
```

```
[Row(label=-9.490009878824548, features=SparseVector(10, {0: 0.4551, 1: 0.3664, 2: -0.3826, 3: -0.4458, 4: 0.3311, 5: 0.8067, 6: -0.2624, 7: -0.4485, 8: -0.0727, 9: 0.5658})),
 Row(label=0.2577820163584905, features=SparseVector(10, {0: 0.8387, 1: -0.127, 2: 0.4998, 3: -0.2269, 4: -0.6452, 5: 0.1887, 6: -0.5805, 7: 0.6519, 8: -0.6556, 9: 0.1749})),
 Row(label=-4.438869807456516, features=SparseVector(10, {0: 0.5026, 1: 0.1421, 2: 0.16, 3: 0.505, 4: -0.9372, 5: -0.2842, 6: 0.6356, 7: -0.1646, 8: 0.9481, 9: 0.4268})),
 Row(label=-19.782762789614537, features=SparseVector(10, {0: -0.0389, 1: -0.4167, 2: 0.8997, 3: 0.641, 4: 0.2733, 5: -0.2618, 6: -0.2795, 7: -0.1307, 8: -0.0854, 9: -0.0546})),
 Row(label=-7.966593841555266, features=SparseVector(10, {0: -0.062, 1: 0.6546, 2: -0.6979, 3: 0.6677, 4: -0.0794, 5: -0.4389, 6: -0.6081, 7: -0.6415, 8: 0.7314, 9: -0.0268})),
```

```
1 -9.490009878824548 1:0.4551273600657362 2:0.36644694351969087
  3:-0.38256108933468047 4:-0.4458430198517267 5:0.33109790358914726
  6:0.8067445293443565 7:-0.2624341731773887 8:-0.44850386111659524
  9:-0.07269284838169332 10:0.5658035575800715
2 0.2577820163584905 1:0.8386555657374337 2:-0.1270180511534269
  3:0.499812362510895 4:-0.22686625128130267 5:-0.6452430441812433
  6:0.18869982177936828 7:-0.5804648622673358 8:0.651931743775642
  9:-0.6555641246242951 10:0.17485476357259122
3 -4.438869807456516 1:0.5025608135349202 2:0.14208069682973434
  3:0.16004976900412138 4:0.505019897181302 5:-0.9371635223468384
  6:-0.2841601610457427 7:0.6355938616712786 8:-0.1646249064941625
  9:0.9480713629917628 10:0.42681251564645817
4 -19.782762789614537 1:-0.0388509668871313 2:-0.4166870051763918
  3:0.8997202693189332 4:0.6409836467726933 5:0.273289095712564
  6:-0.26175701211620517 7:-0.2794902492677298 8:-0.1306778297187794
  9:-0.08536581111046115 10:-0.05462315824828923
5 -7.966593841555266 1:-0.06195495876886281 2:0.6546448480299902
  3:-0.6979368909424835 4:0.6677324708883314 5:-0.07938725467767771
  6:-0.43885601665437957 7:-0.608071585153688 8:-0.6414531182501653
  9:0.7313735926547045 10:-0.026818676347611925
6 -7.896274316726144 1:-0.15805658673794265 2:0.26573958270655806
  3:0.3997172901343442 4:-0.3693430998846541 5:0.14324061105995334
  6:-0.25797542063247825 7:0.7436291919296774 8:0.6114618853239959
  9:0.2324273700703574 10:-0.25128128782199144
```

LIBSVM data format

MLlib Example

```
from pyspark.ml.regression import LinearRegression

# Load training data
training = spark.read.format("libsvm").load("sample_linear_regression_data.txt")

lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)
```

```
# Print the coefficients and intercept for linear regression
print("Coefficients: %s" % str(lrModel.coefficients))
print("Intercept: %s" % str(lrModel.intercept))
```

```
Coefficients: [0.0,0.32292516677405936,-0.3438548034562218,1.9156017023458414,0.05288058680386263,0.765962720459771,
0.0,-0.15105392669186682,-0.21587930360904642,0.22025369188813426]
Intercept: 0.1598936844239736
```

MLlib Example

```
# Summarize the model over the training set and print out some metrics
trainingSummary = lrModel.summary
print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show(5)
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

```
numIterations: 7
objectiveHistory: [0.4999999999999994, 0.4967620357443381, 0.4936361664340463, 0.4936351537897608, 0.493635121417787
1, 0.49363512062528014, 0.4936351206216114]
+-------------------+
|          residuals|
+-------------------+
| -9.889232683103197|
| 0.5533794340053554|
| -5.204019455758823|
|-20.566686715507508|
|   -9.4497405180564|
+-------------------+
only showing top 5 rows

RMSE: 10.189077
r2: 0.022861
```

LOGISTIC REGRESSION

Logistic Regression

- How can we use linear regression to do classification?
- The range of linear regression model is $(-\infty, +\infty)$.
- Can we map it into the range $[0, 1]$?

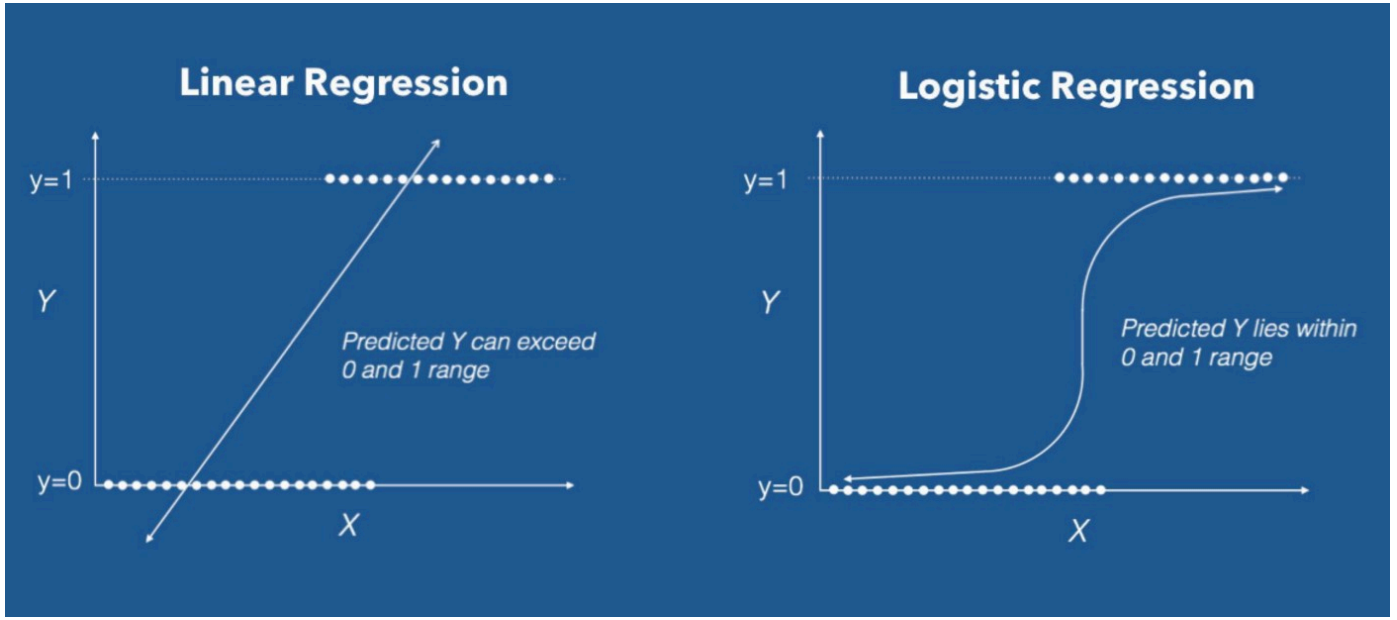


Image source: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Sigmoid Function

- We can make a new model by using the sigmoid function which maps $(-\infty, +\infty)$ to $[0, 1]$:

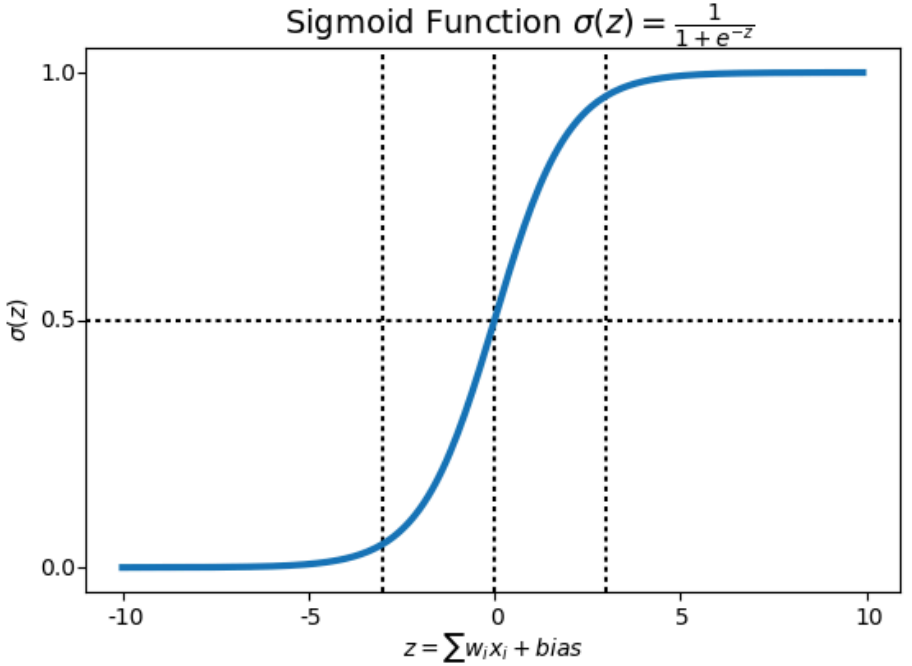
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  while $z = \boldsymbol{w}^T \boldsymbol{x} + b$.

- The sigmoid function can be used to represent the probability of each class:

$$P(y = 1|z) = \sigma(z)$$
$$P(y = 0|z) = 1 - \sigma(z)$$

- Now, if $\sigma(z)$ is in $[0, 1]$.
  - If $\sigma(z) < 0.5$, we classify $\boldsymbol{x}$ as 0.
  - If $\sigma(z) \geq 0.5$, we classify $\boldsymbol{x}$ as 1.



The sigmoid function is also called logistic function

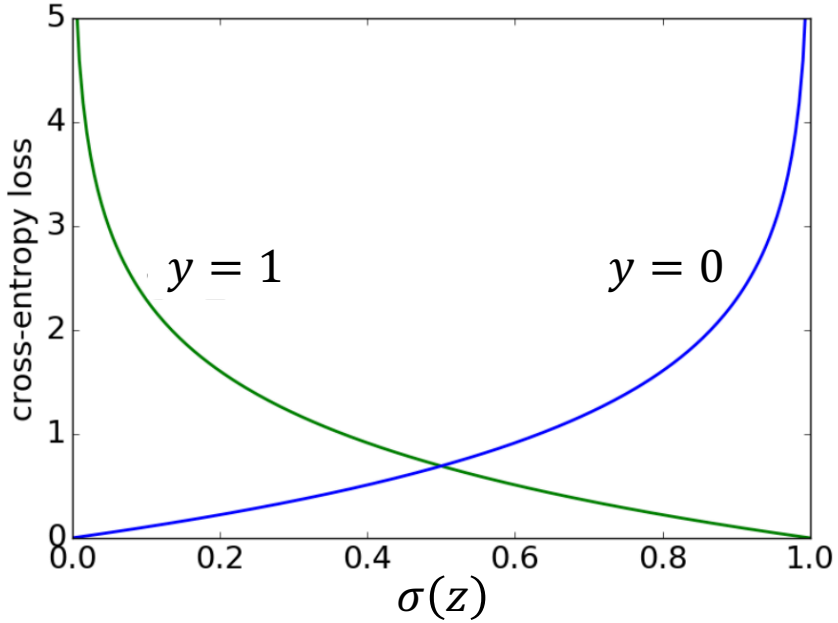
Image source: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Cross-Entropy Cost Function

- MSE is no longer suitable for measuring the error for a classification problem.
- Instead, we use *cross-entropy cost function* (aka *log loss*):

$$J(z) = \begin{cases} -\log \sigma(z_i) & \text{if } y_i = 1 \\ -\log(1 - \sigma(z_i)) & \text{if } y_i = 0 \end{cases}$$
$$= -y_i \log \sigma(z_i) - (1 - y_i) \log(1 - \sigma(z_i))$$

- If you are interested in how this formula is derived, more details can be found here: https://peterroelants.github.io/posts/cross-entropy-logistic/



Derivative of the Cross-Entropy Cost Function

- Calculate partial derivatives:

$$\frac{\partial J}{\partial \sigma} = \frac{\partial(-y \log \sigma - (1-y)\log(1-\sigma))}{\partial \sigma} = \frac{y}{\sigma} + \frac{1-y}{1-\sigma} = \frac{\sigma - y}{\sigma(1-\sigma)}$$

$$\frac{\partial \sigma}{\partial z} = \frac{\partial \frac{1}{1+e^{-z}}}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^2} = \sigma(1-\sigma).$$

- By the chain rule, we have:

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial \sigma}\frac{\partial \sigma}{\partial z} = \frac{\sigma - y}{\sigma(1-\sigma)}\sigma(1-\sigma) = \sigma - y.$$

- Then, we can easily get $\partial J/\partial w_i$ and $\partial J/\partial b$ by using chain rule again with $\partial z/\partial w_i$ and $\partial z/\partial b$.

Iteration with Gradient Descend

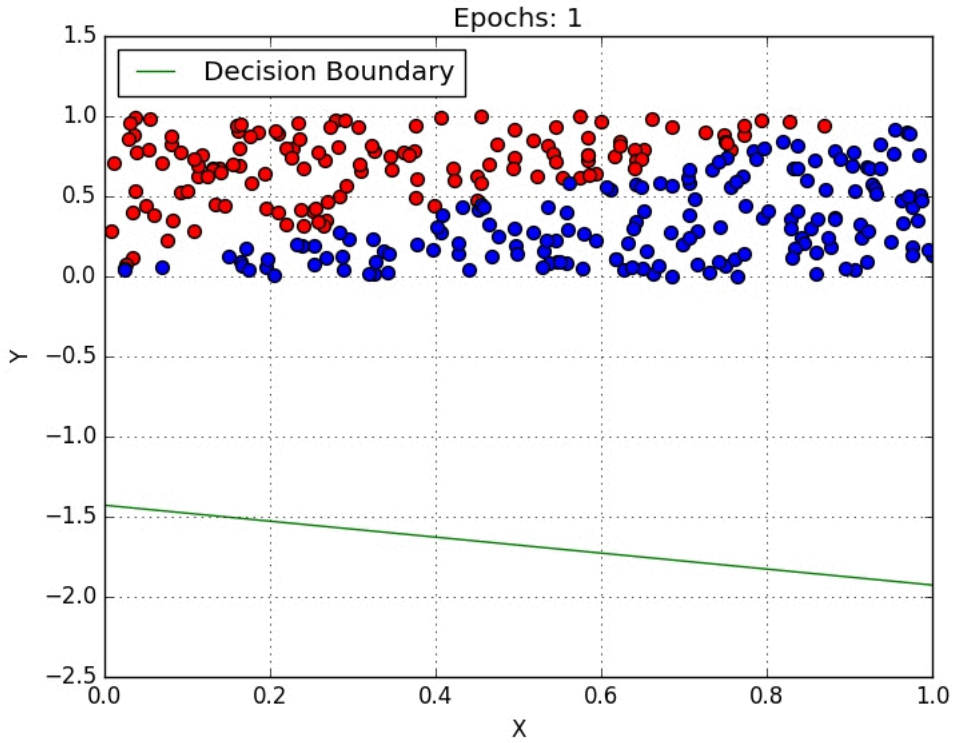


Image source: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Advantages and Disadvantages

- Advantages:
  - Easy to implement, interpret and very efficient to train.
  - Can be used to train extremely large dataset.
- Disadvantages:
  - Sometimes too simple to capture the complex relationships between features.
  - Does poorly with correlated features.

MLlib API

class `pyspark.ml.classification.`**`LogisticRegression`**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, elasticNetParam=0.0, tol=1e-06, fitIntercept=True, threshold=0.5, thresholds=None, probabilityCol='probability', rawPredictionCol='rawPrediction', standardization=True, weightCol=None, aggregationDepth=2, family='auto', lowerBoundsOnCoefficients=None, upperBoundsOnCoefficients=None, lowerBoundsOnIntercepts=None, upperBoundsOnIntercepts=None*) [source]

- Commonly used hyperparameters:
  - **maxIter**, **regParam**, **elasticNetParam**, **tol** are same as linear regression.
  - **family:** The name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial.
  - **threshold:** Threshold in binary classification prediction, in range [0, 1].

Source: https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.LogisticRegression

MLlib Example

```
from pyspark.ml.classification import LogisticRegression

# Load training data
training = spark.read.format("libsvm").load("sample_libsvm_data.txt")

lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)
```

```
# Print the coefficients and intercept for logistic regression
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

```
Coefficients: (692,[244,263,272,300,301,328,350,351,378,379,405,406,407,428,433,434,455,456,461,462,483,484,489,490,4
96,511,512,517,539,540,568],[-7.353983524188197e-05,-9.102738505589466e-05,-0.0001946743054690429 8,-0.000203006424734
86668,-3.1476183314863995e-05,-6.842977602660743e-05,1.5883626898239883e-05,1.4023497091372047e-05,0.0003543204752496
8605,0.00011443272898171087,0.00010016712383666666,0.0006014109303795481,0.0002840248179122762,-0.000115410847365088 3
7,0.000385996886312906,0.000635019557424107,-0.00011506412384575676,-0.0001527186586498680 8,0.000280493380899421 4,0.0
006070117471191634,-0.0002008459663247437,-0.0001421075579290126,0.0002739010341160883,0.0002773045624496811 5,-9.8380
27027269332e-05,-0.0003808522443517704,-0.0002531519800855503 3,0.0002774771477075430 7,-0.000244361976391919 9,-0.00153
94744687597765,-0.00023073328411331293])
Intercept: 0.22456315961250325
```

NEURAL NETWORKS

XOR Problem

- XOR is short for exclusive or operation:

$$XOR(0,0) = 0 \quad XOR(1,1) = 0$$
$$XOR(1,0) = 1 \quad XOR(0,1) = 1$$

- Using a linear model (a line in 2d or a plane in 3d) can never correctly classify the XOR problem.

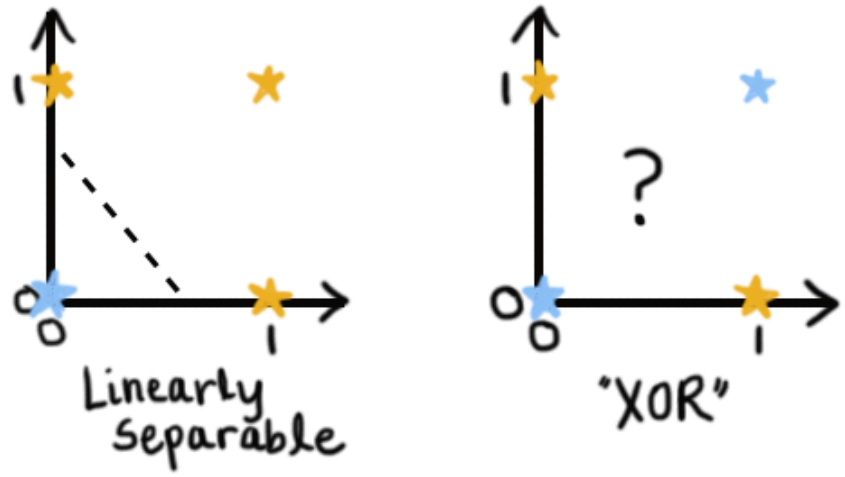
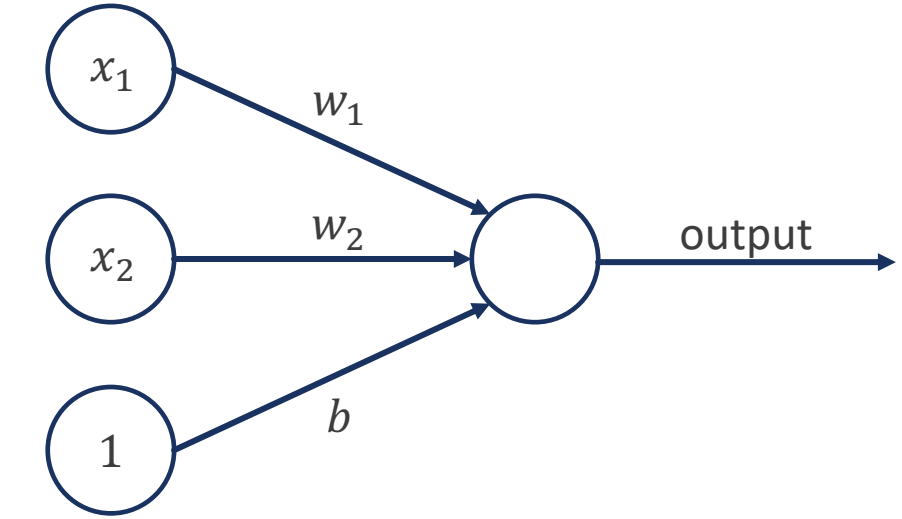


Image source: https://blog.insightdatascience.com/a-quick-history-of-neural-nets-from-inglorious-to-incredible-46e115c38b95

Perceptron Model

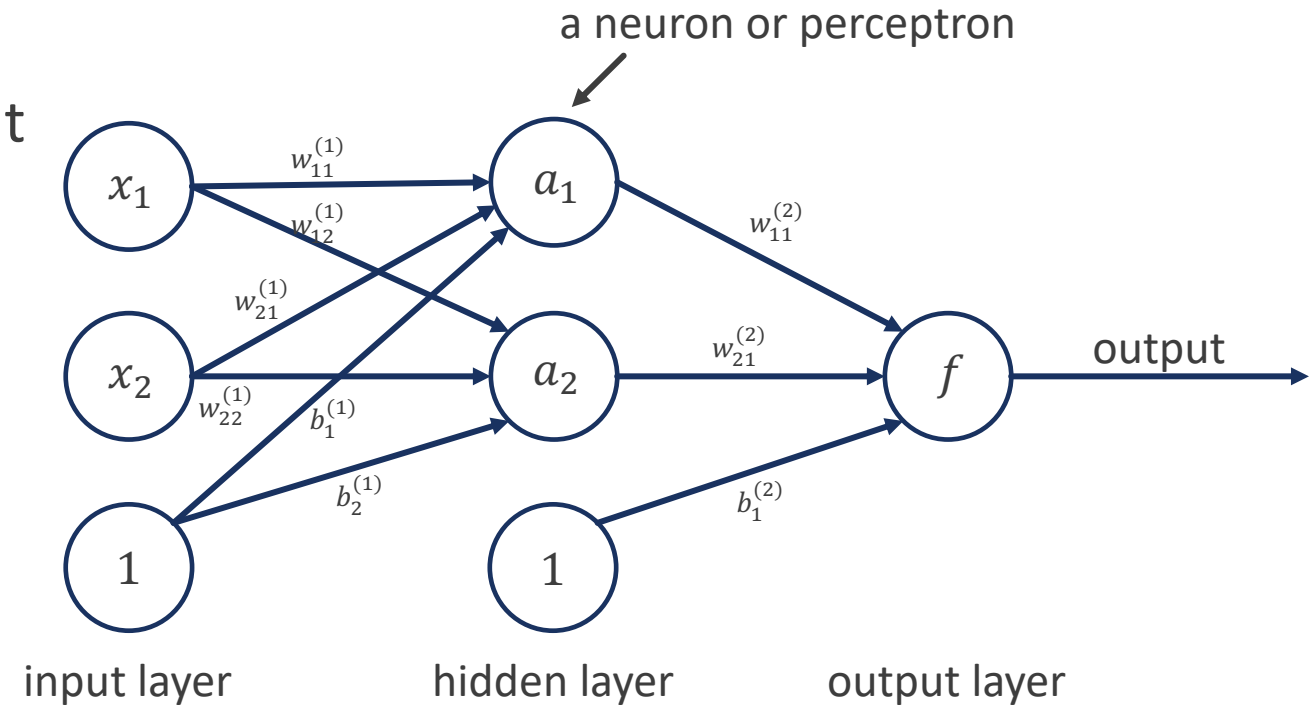
- The previous linear model is also called perceptron model.
- This model has an input layer and an output layer.



Multilayer Perceptrons

- The hidden layer is used as the input of output layer.
- However, this model is still linear because

$$
\begin{aligned}
f(\boldsymbol{x}) &= w_{11}^{(2)} a_1 + w_{21}^{(2)} a_2 + b_1^{(2)} \\
&= w_{11}^{(2)} \left(w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + b_1^{(1)} \right) \\
&\quad + w_{21}^{(2)} \left(w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)} \right) \\
&= (\ldots) x_1 + (\ldots) x_2 + b
\end{aligned}
$$



Non-Linearity

- For the output of each layer, we add an function to make it non-linear. This function is called *activation function*.
- Activation function is required to be derivable such that it will not influence the use of gradient descend.
- We can use sigmoid function as the activation function.

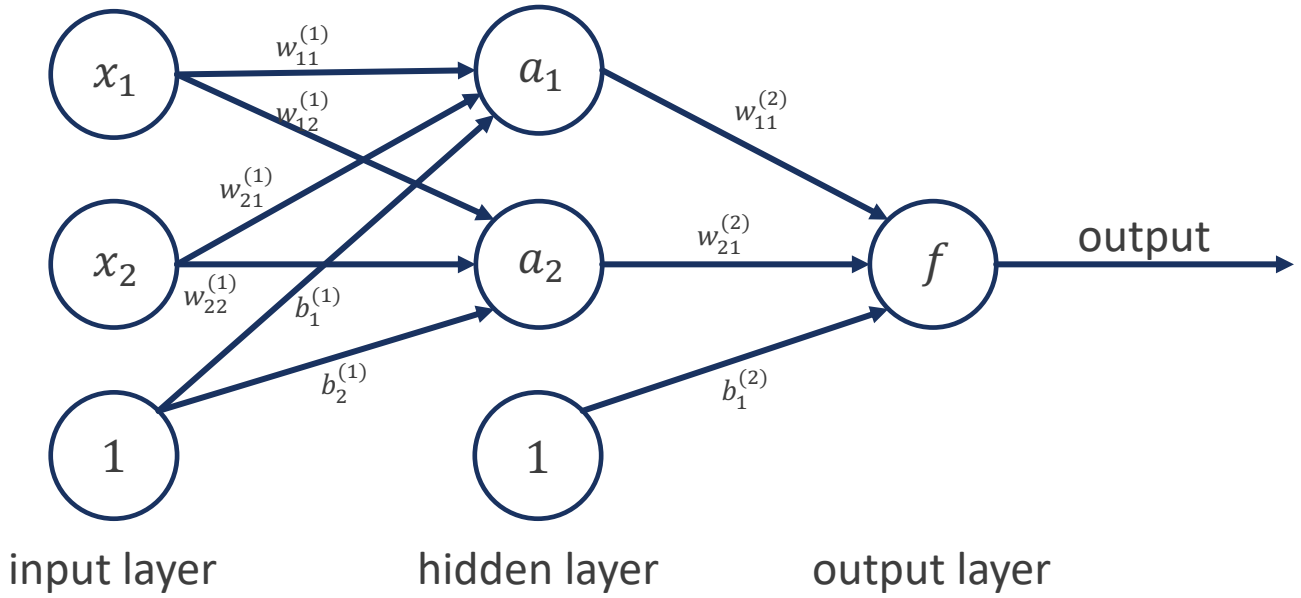
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- The alternatives are tanh and ReLU, which are commonly adopted in deep neural networks.

Non-Linearity

- Thus, we have:

$$a_1 = \sigma\left(w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + b_1^{(1)}\right)$$
$$a_2 = \sigma\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + b_2^{(1)}\right)$$
$$f(\boldsymbol{x}) = \sigma\left(w_{11}^{(2)}a_1 + w_{21}^{(2)}a_2 + b_1^{(2)}\right).$$



Non-Linearity

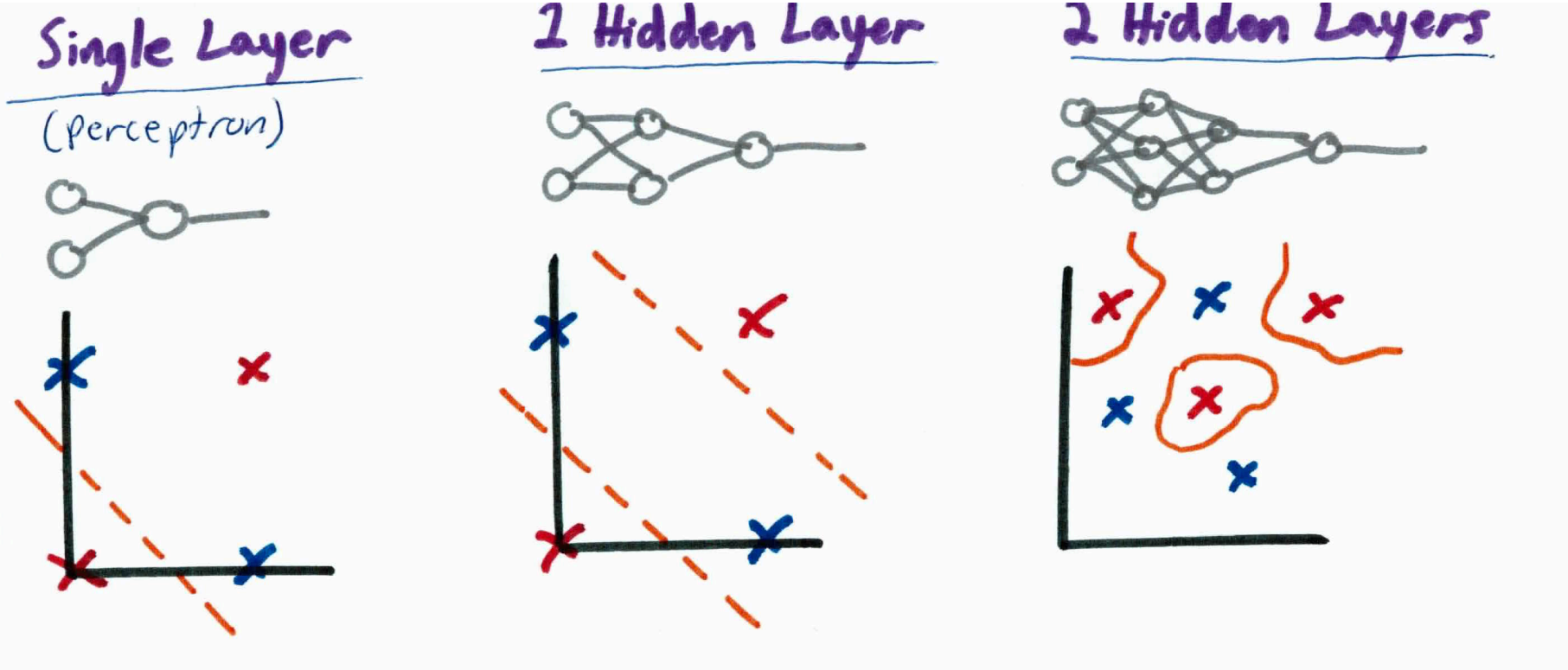


Image source: https://kseow.com/nn

Backpropagation

- We *feedfoward* the information from one layer to another layer, to produce an output.
- We pass the errors backwards so the network can learn by adjusting the weights of the network.
  - *Backpropagation* stands for *backward propagation of errors*.

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial w_{11}^{(2)}}$$

$$\frac{\partial L}{\partial w_{21}^{(2)}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial w_{21}^{(2)}}$$

$$\frac{\partial L}{\partial b_{1}^{(2)}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial b_{1}^{(2)}}$$

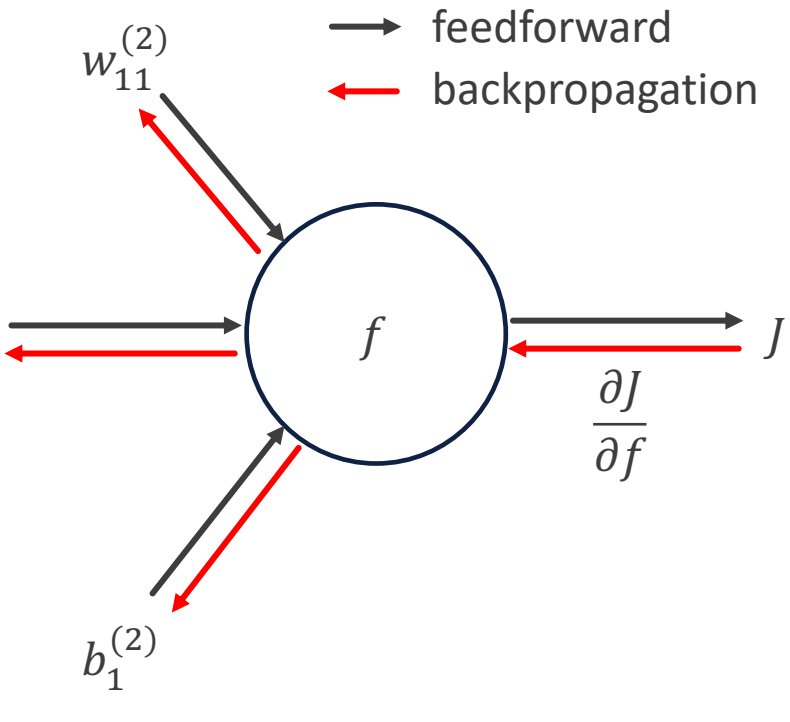


Image source: https://www.kaggle.com/romaintha/an-introduction-to-backpropagation

Backpropagation

- Making use of the chain rule of calculus, we can express the gradient of J with respect to the weights and biases as.
- For a multilayer perceptron model with one hidden layer.
  - $w_{ij}^{(1)}$ is the weight connecting the ith feature in the input layer and the jth neuron in the hidden layer.
  - $w_{ij}^{(2)}$ is the weight connecting the ith neuron in the hidden layer and the jth neuron in the output layer.

$$\frac{\partial J}{\partial w_{ij}^{(2)}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial w_{ij}^{(2)}}$$

$$\frac{\partial J}{\partial w_{ij}^{(1)}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial w_{ij}^{(1)}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial a_i}\frac{\partial a_i}{\partial w_{ij}^{(1)}}$$

Multiclass Classification by Neural Networks

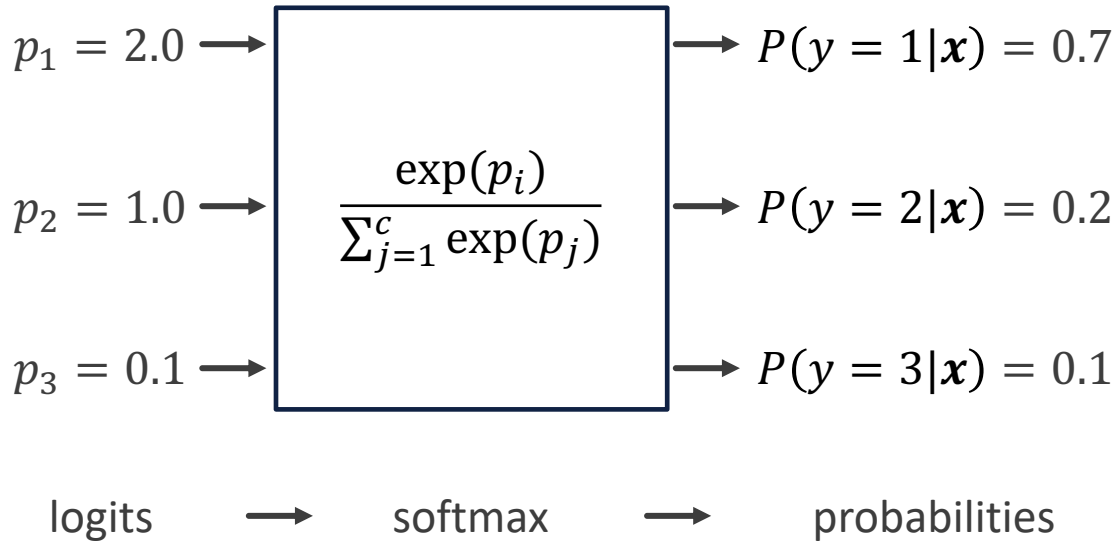
- For a binary classification problem, only one neuron in the output layer is enough.
  - It generates the probability of 0/1.
- For multiclass classification, we may have multiple neurons in the output layer. Each of them generates a score of one class.
  - Then we take the one with maximum score as the predicted class.
- However, the maximum operator is not derivable.

Softmax Function

- The *softmax function* is calculated by:

$$P(y = i|\boldsymbol{x}) = \frac{\exp(p_i)}{\sum_{j=1}^{c} \exp(p_j)}$$

- p_i is the score of the ith class. They are called the *logits*.
  - E.g. $p_i = w_{1i}^{(2)} a_1 + w_{2i}^{(2)} a_2 + b_i^{(2)}$ for the previous example.
- When there are only two classes, softmax function reduces to sigmoid function.



Advantages and Disadvantages

- Advantages:
  - Can handle extremely complex tasks, e.g. image recognition.
  - It has the ability to learn any non-linear functions, if the network is deep enough.
- Disadvantages:
  - Difficult to interpret. The model is like a black box.
  - Very high demand of computational resources.
  - There is no specific rule for determining the structure of artificial neural networks. The appropriate network structure is achieved through experience and trial and error.

MLlib API

class `pyspark.ml.classification.`**`MultilayerPerceptronClassifier`**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, tol=1e-06, seed=None, layers=None, blockSize=128, stepSize=0.03, solver='l-bfgs', initialWeights=None, probabilityCol='probability', rawPredictionCol='rawPrediction'*) [source]

- Each layer has sigmoid activation function, output layer has softmax.
- Number of inputs has to be equal to the size of feature vectors. Number of outputs has to be equal to the total number of labels.
- Commonly used hyperparameters:
  - **layers:** Sizes of layers from input layer to output layer E.g., [780, 100, 10] means 780 inputs, one hidden layer with 100 neurons and output layer of 10 neurons.
  - **blockSize:** Block size for stacking input data in matrices. Data is stacked within partitions. Recommended size is between 10 and 1000, default is 128.
  - **stepSize:** Step size to be used for each iteration of optimization (>= 0).

Source: https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.MultilayerPerceptronClassifier

MLlib Example

```
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Load training data
data = spark.read.format("libsvm").load("sample_multiclass_classification_data.txt")

# Split the data into train and test
train, test = data.randomSplit([0.6, 0.4], 1234)

# specify layers for the neural network:
# input layer of size 4 (features), two intermediate of size 5 and 4
# and output of size 3 (classes)
layers = [4, 5, 4, 3]

# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)

# train the model
model = trainer.fit(train)

# compute accuracy on the test set
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))
```

```
Test set accuracy = 0.9019607843137255
```

SUPPORT VECTOR MACHINES

Optimal Classification Hyperplane

- For the same training data, we may find several different classification hyperplane that has the same error rate.
  - They have the same training error, but when given unknown test data, the test error is different.
- Is there a criterion to select the best hyperplane, such that it has highest probability to correctly classify the unknown test data?

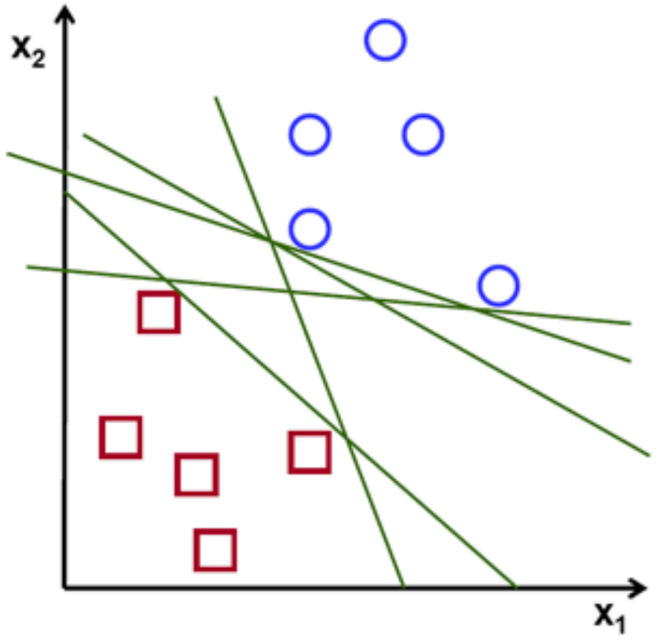


Image source: https://towardsdatascience.com/svm-feature-selection-and-kernels-840781cc1a6c

Optimal Classification Hyperplane

- One criterion is to maximize the margin between the hyperplane and the nearest samples.
- A classification model with such optimal hyperplane will have good *generalization ability*.
  - A model with poor generalization ability performs well on the training data but poorly on the test data.
  - A model with good generalization ability performs well on both the training data and the test data.

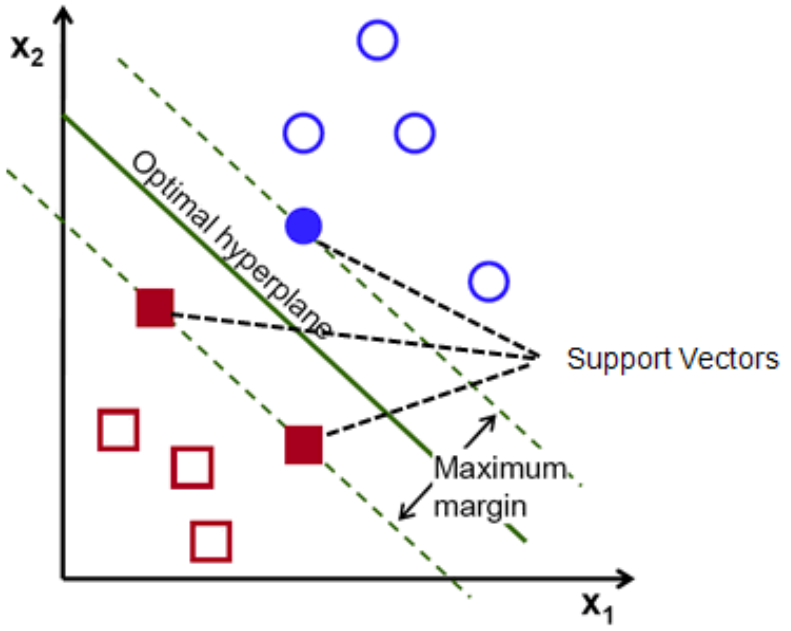


Image source: https://dimensionless.in/introduction-to-svm/

SVM Optimization

- The hyperplane can be represented as $\boldsymbol{w}^T \boldsymbol{x} + b = 0$.
- The optimization of maximizing margin can be derived as:

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\ s.t. \quad & y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 \\ & \text{for } i = 1, \ldots, n \end{aligned}$$

where $\|\boldsymbol{w}\|^2 = w_1^2 + w_2^2 + \cdots + w_d^2$.

- y_i needs to be converted to +1/-1 from 1/0.
- This is a quadratic programming problem.
- However, it the training data is not linear separable, we will not be able to find a hyperplane satisfying the condition.

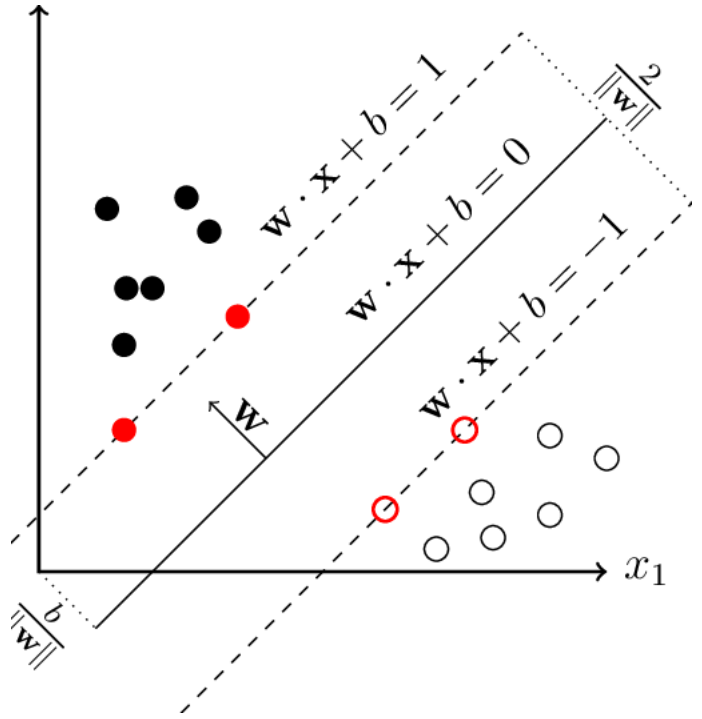


Image source: https://www.researchgate.net/profile/Victor_Suarez-Paniagua/publication/334643403/figure/fig5/AS:783985462484992@1563928108122/An-SVM-separating-two-classes-by-an-hyperplane-wx-b-0.png

Soft Margin SVM

- For every data point $\boldsymbol{x}_i$, we introduce a *slack variable* ξ_i.
- The value of ξ_i is the distance of $\boldsymbol{x}_i$ from its corresponding class's margin if $\boldsymbol{x}_i$ is on the wrong side of the margin, otherwise zero.
- The points that are far away from the margin on the wrong side would get more penalty.

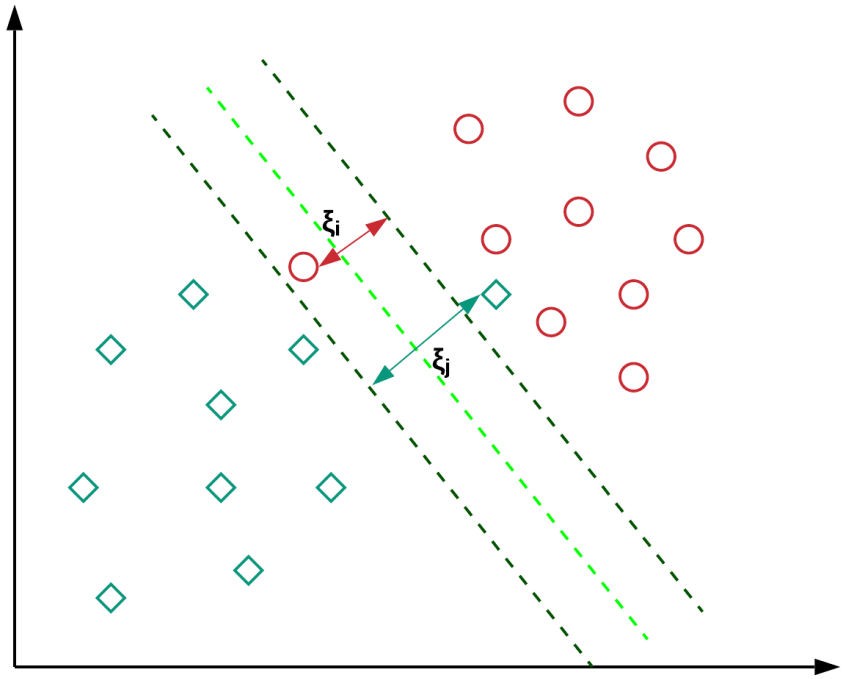
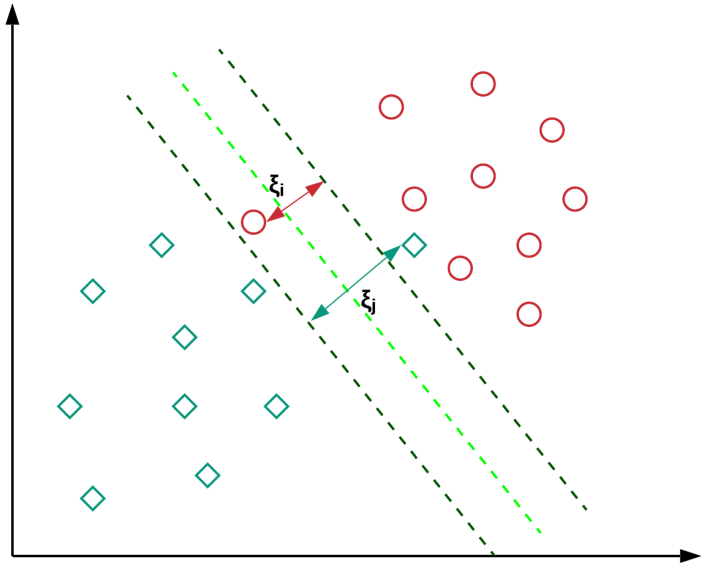


Image source: https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

Soft Margin SVM

- The optimization of maximizing margin can be modified to the soft margin version:

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ s.t. \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \\ & \text{for } i = 1, \dots, n \end{aligned}$$



- C is a hyperparameter that decides the trade-off between maximizing the margin and minimizing the mistakes.
  - Small C gives less importance to classification mistakes and focuses more on maximizing the margin.
  - Large C focuses more on avoiding misclassification at the expense of keeping the margin small.

Image source: https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

Kernel SVM

- The previous version of SVM is still a linear model.
- It will never correctly classifies the data like this.

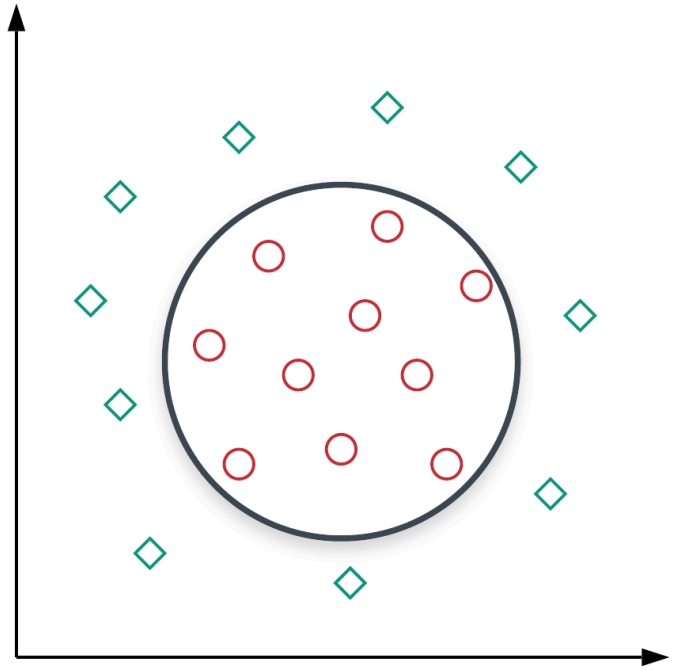


Image source: https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe

Kernel SVM

- The previous optimization problem is solved with Lagrange multiplier. Its dual optimization problem is:

$$\max_{\alpha} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$
$$s.t. \quad 0 \leq \alpha_i \leq C, \text{for } i = 1, \dots, n$$
$$\sum_{i=1}^{n} \alpha_i \, y_i = 0$$

- $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ is the inner product between the ith and jth sample, also called the linear kernel.
- Replacing $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ to a kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ will produce non-linear hyperplane.

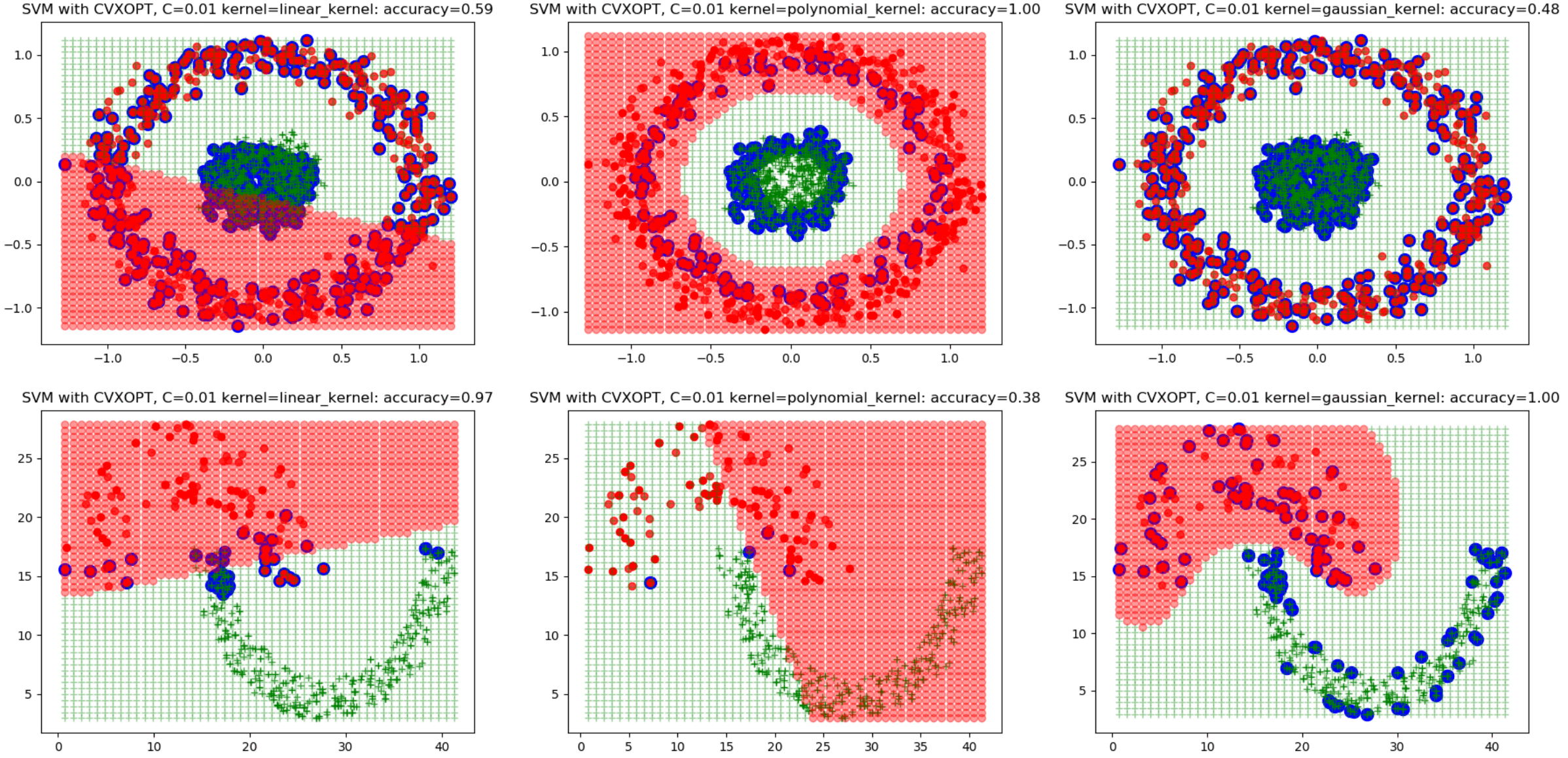
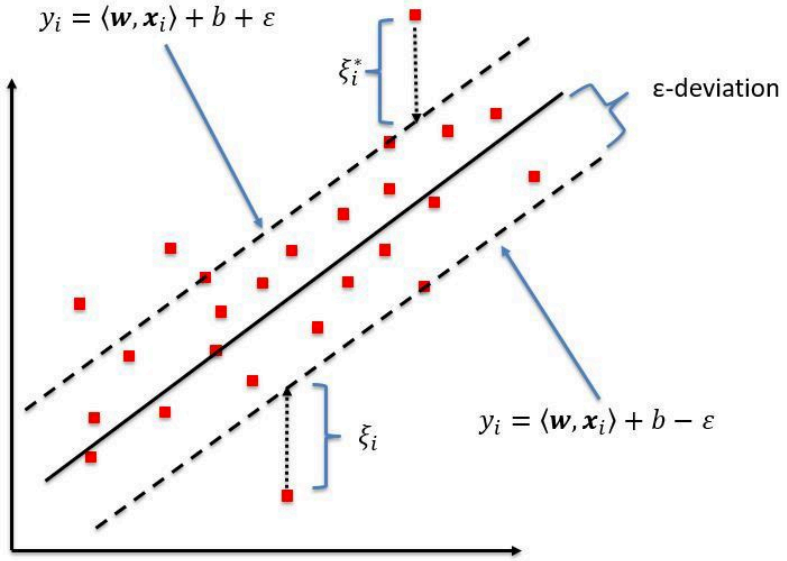


Image source: https://www.datasciencecentral.com/profiles/blogs/implementing-a-soft-margin-kernelized-support-vector-machine

Support Vector Regression

- Use the same idea as SVM.
- The goal is to find a function $f(x)$ that has at most ε deviation from the actually obtained targets y_i for all the training data, and at the same time is as flat as possible.

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ s.t. \quad & y_i - (\boldsymbol{w}^T\boldsymbol{x}_i + b) \le \varepsilon + \xi_i \\ & (\boldsymbol{w}^T\boldsymbol{x}_i + b) - y_i \le \varepsilon + \xi_i^* \\ & \xi_i, \xi_i^* \ge 0 \end{aligned}$$



- It is also called ε-SVR.

Image source: https://medium.com/coinmonks/support-vector-regression-or-svr-8eb3acf6d0ff

Advantages and Disadvantages

- Advantages:
  - SVM works relatively well when there is clear margin of separation between classes.
  - With kernel trick, SVM is able to capture complex feature relationship.
- Disadvantages:
  - SVM algorithm is not suitable for large data sets. Training is very time-consuming.
  - SVM does not perform very well, when the data set has more noise i.e. target classes are overlapping.
  - No probabilistic explanation for the classification.

MLlib

class `pyspark.ml.classification.`**`LinearSVC`**(*featuresCol='features', labelCol='label', predictionCol='prediction', maxIter=100, regParam=0.0, tol=1e-06, rawPredictionCol='rawPrediction', fitIntercept=True, standardization=True, threshold=0.0, weightCol=None, aggregationDepth=2*) ¶ [source]

- MLlib only supports simple linear SVM.
- Kernel SVM and SVR are not supported in MLlib.

Source: https://spark.apache.org/docs/latest/ml-classification-regression.html#linear-support-vector-machine

MACHINE LEARNING RELATED ISSUES

Overfitting

- Is a model the more complex the better?
- No. It will overfit to the training data and perform poorly on the test data.
  - Too complex to be generalized.

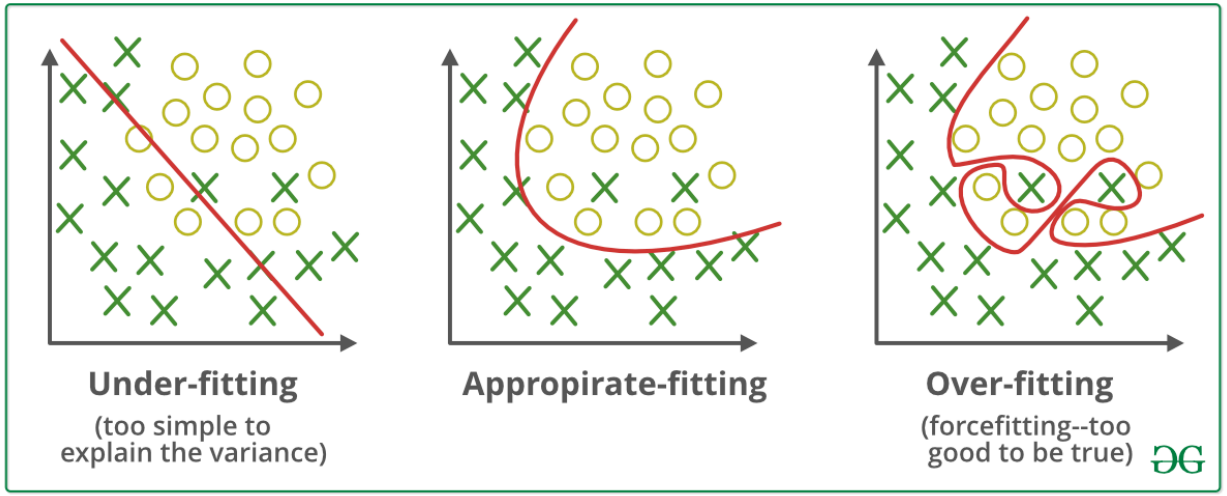


Image source: https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf

Overfitting

- As we increse the model complexity (e.g. add a bunch of hidden layers to neural networks), the training error will decrease, but the test error will increase.

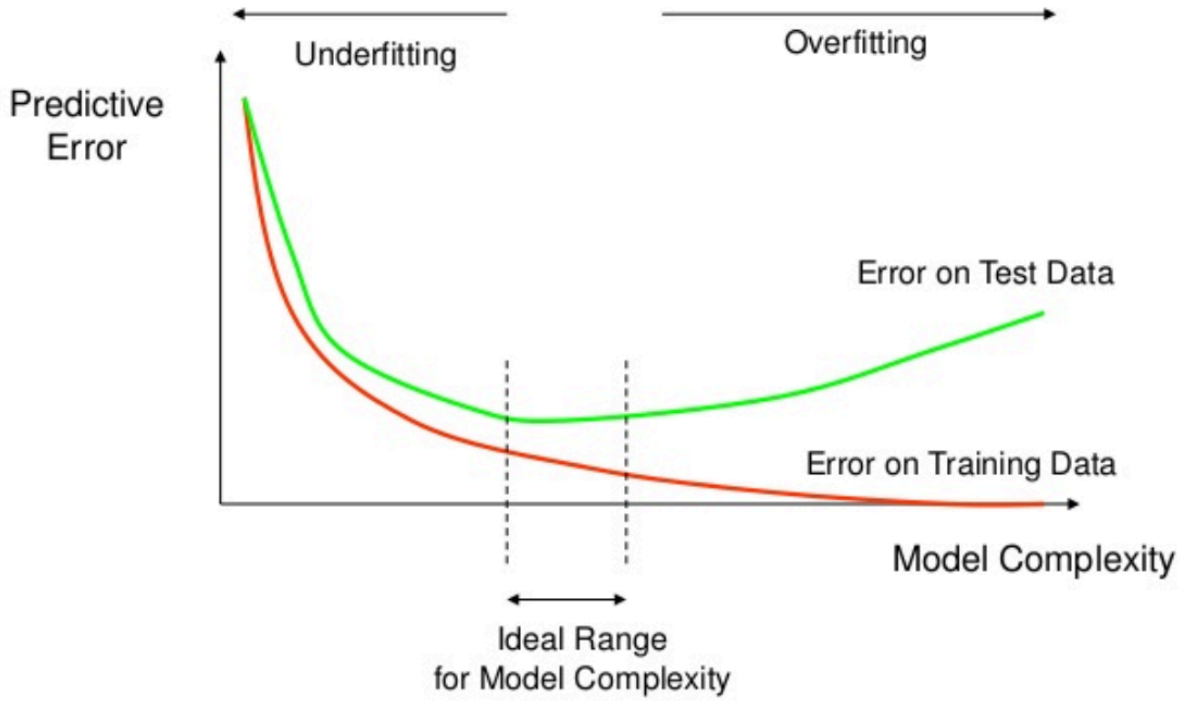


Image source: https://stats.stackexchange.com/questions/292283/general-question-regarding-over-fitting-vs-complexity-of-models

Regularization

- One solution it to control the model complexity by *regularization*.
- Add regularization panelty to the cost function.
- Take linear regression as an example:

$$J = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{w}^T\boldsymbol{x}_i - y_i)^2 + \lambda\|\boldsymbol{w}\|^2$$

- The model complexity is measure by $\|\boldsymbol{w}\|^2$, aka l^2 regularization. λ is a trade-off hyperparameter to balance the model accuracy and complexity.

Conclusion

After this lecture, you should know:

- What is linear and non-linear models.
- What is gradient descent.
- How to use gradient descent to update the model.
- What are the advantages and disadvantages of each model.

Thank you!

- Any question?
- Don’t hesitate to send email to me for asking questions and discussion. ☺