INT104 ARTIFICIAL INTELLIGENCE

L5- Classification & Training models

Fang Kang
fang.kang@xjtlu.edu.cn
Xi'an Jiaotong-Liverpool University
西交利物浦大学

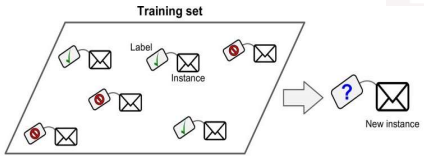
CONTENT

- **Classification**
  - Binary Classifier
  - Performance Measure (Accuracy/Precision/Recall/F1/ROC)
  - Multiclass/Multilabel Classification
- **Training models**
  - Linear Regression
  - Gradient Descent
  - Polynomial Regression
  - Learning Curves
  - Regularized Linear Models
  - Logistic Regression

Binary Classifier

Classification: Classification algorithms find a function that determines which category the input data belongs to.

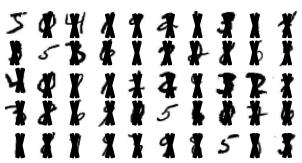
Binary Classification is a supervised learning algorithm that classifies new observations into one of two classes.



Binary Classifier

Let's simplify the problem for now and only try to identify one digit—for example, the number 5.

- If the label is "5"--- Positive
- If the label is not "5" --- Negative



Performance Measures

Why do we need to evaluate machine learning models?

- The primary purpose of machine learning models is often to make a decision or develop insight. And in service of these goals, it is important to know **how much we can really trust that model and data.**
- Once you have built a machine learning framework (e,g. classifier), we should **know its performance** (e,g. accuracy).
- When you have a real-world problem, you would **compare different models** to pick the right one for it.

Performance Measures

Metrics to evaluate **Classification** models

- Accuracy
- Confusion Matrix (not a metric but fundamental to others)
- Precision and Recall
- F1-score
- AUC&ROC

Accuracy

Train/Test split:

We can split the entire dataset into train and test sets (e,g. 70% for training, 30% for testing). However, the generalization performance of a machine learning method relates to its prediction capability on independent test sets.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

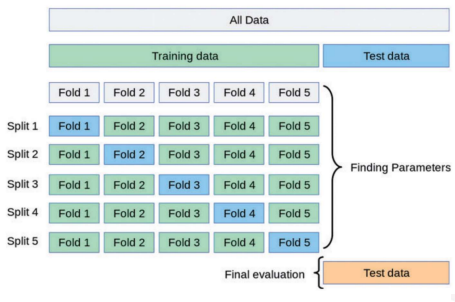
Cross Validation

Train/test/validation split

To avoid selecting the parameters that perform best on the test data but maybe not the parameters that generalize best, we can further split the training set into training fold and validation fold

- **Training fold**: used to fit the model
- **Validation fold**: used to estimate prediction error for model selection
- **Test set**: used for assessment of the prediction error of the final chosen model

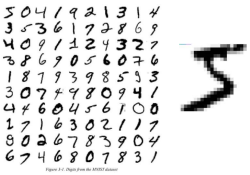
K-fold Cross-Validation



Class imbalance

Identify one digit—for example, the number 5.

- If the label is "5"---Positive
- If the label is not "5" ---Negative



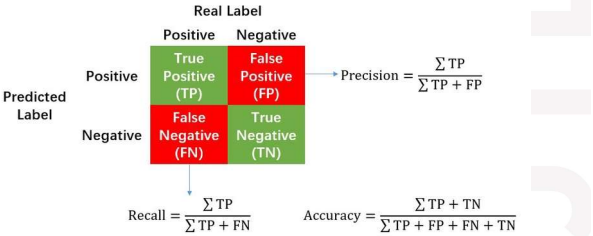
- Can't always use accuracy
  - All false was still > 90%
- Class imbalance
  - 90% of our samples were not 5
  - 10% of our samples were 5
- Try other methods
  - Confusion Matrix
  - Precision & Recall
  - AUC & ROC Curve

Confusion Matrix

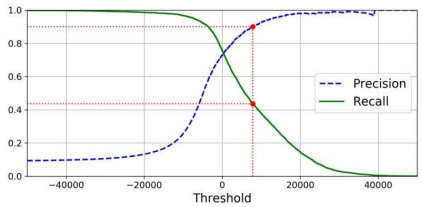
When performing classification or predictions, there are four types of outcomes that could occur:

- True Positive (TP): Predict an observation belongs to a class and it actually does belong to that class.
- True Negative (TN): Predict an observation does not belong to a class and it actually does not belong to that class.
- False Positives (FP): Predict an observation belongs to a class and it actually does not belong to that class.
- False Negatives (FN): Predict an observation does not belong to a class and it actually does belong to that class.

Confusion Matrix



Precision/Recall Trade-off



Note

If someone says, "Let's reach 99% precision," you should ask, "At what recall?

Precision / Recall

- Trade off between precision and recall
  - With precision - make sure what you're saying is positive is actually positive
  - With recall - make sure you're not missing out on positive observations
  - As one increases, the other decreases
  - Metrics like F1 scores average them both

F1 Score

- Harmonic mean of precision and recall
  - Gives more weight to low values
  - Only get a high F1 score if both are high
  - Typically precision & recall are similar

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

F1 Score

```
from sklearn.metrics import f1_score
f1_score(y_train_5, y_train_pred)
```

```
0.7325171197343846
```

F_β score: a more flexible F score that combines precision and recall

$$F_\beta\ score = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$$

$\beta < 1$ focuses more on precision
$\beta > 1$ focuses more on recall

Precision / Recall / F1

Optimization guide

- So do I want to improve F1, precision, or recall?
- Depends on the situation
  - Classifier to detect if videos are safe for kids
    - Reject many good videos (low recall) but keep safe one (high precision)
  - Classifier to detect shoplifters?
    - May give false positives (high recall) but captures all thieves (low precision)

Practice

	Real Label	
Predicted	TP: 63	FN: 37
	FP: 28	TN: 72

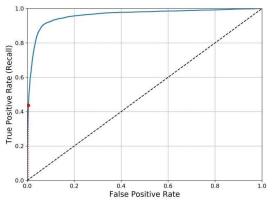
	Real Label	
Predicted	TP: 77	FN: 23
	FP: 77	TN: 23

	Real Label	
Predicted	TP: 24	FN: 76
	FP: 88	TN: 12

Accuracy	Recall	Precision	F1 Score

Receiver Operation Characteristics (ROC)

A **ROC curve** (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. In another word, it presents **Recall** (True Positive Rate) VS **FPR** (**False Positive Rate**)



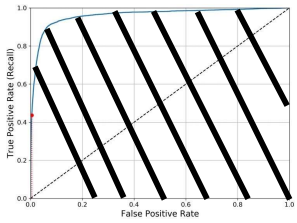
$$Recall = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{TN + FP}$$

The ROC graph summarizes all of the confusion matrices that each threshold produced.

Area Under the Curve(AUC)

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0; one whose predictions are 100% correct has an AUC of 1.



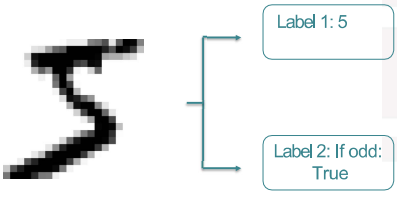
Multiclass Classification

Multiclass classification refers to classification tasks that can distinguish between more than two classes.

- One-versus-the-rest(OvR) strategy: train multiple binary classifiers for each class, select the class whose classifier outputs the highest score.
  - train N times
- One-versus-one (OvO) strategy: train a binary classifier for every pair of classes
  - train N(N-1)/2 times

Multilabel Classification

Multilabel classification refers to classification system that outputs multiple binary tags.

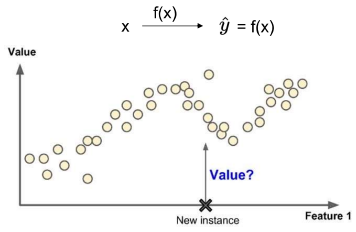


CONTENT

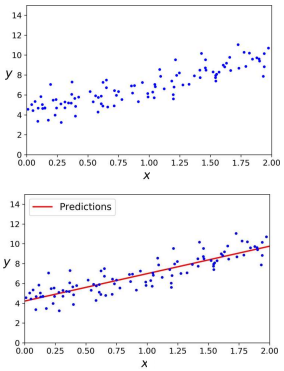
- Classification
  - Binary Classifier
  - Performance Measure (Accuracy/Precision/Recall/F1/ROC)
  - Multiclass/Multilabel Classification
- **Training models**
  - Linear Regression
  - Gradient Descent
  - Polynomial Regression
  - Learning Curves
  - Regularized Linear Models
  - Logistic Regression

Regression

Regression attempts to determine the strength and character of the relationship between one dependent variable (usually denoted by Y) and a series of other variables (known as independent variables).



Simple Linear Regression



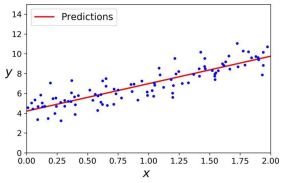
$$\hat{y} = kx + b$$

x: feature value (input)

$\hat{y}$: predicted value (output)

k,b: model parameters (b: bias term, k: weight)

Simple Linear Regression



Usually the predicted value (fitted value) $\hat{y}$ is not perfect. The difference between the fitted value and real value y is known as residuals $\hat{e}$

$$\hat{y}^{(i)} = kx^{(i)} + b$$

$$\hat{e}_i = y^{(i)} - \left(kx^{(i)} + b\right) = y^{(i)} - \hat{y}^{(i)}$$

The regressed value usually pursues a minimum of residual sum of square (RSS)

$$RSS = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

Linear Regression

A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the bias term (also called the intercept term)

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the i^{th} feature value.
- θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

Linear Regression

Linear Regression model prediction (vectorized form)

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \longrightarrow \hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^{\mathrm{T}} \cdot \mathbf{x}$$

- $\boldsymbol{\theta}$ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- $\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.
- $\boldsymbol{\theta} \cdot \mathbf{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\mathbf{x}$, which is of course equal to $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$.
- $h_{\boldsymbol{\theta}}$ is the hypothesis function, using the model parameters $\boldsymbol{\theta}$.

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Note

In Machine Learning, vectors are often represented as column vectors. If $\boldsymbol{\theta}$ and $\mathbf{x}$ are column vectors, then the prediction is $\hat{y} = \boldsymbol{\theta}^{\mathrm{T}} x$, where $\boldsymbol{\theta}^{\mathrm{T}}$ is the transpose of $\boldsymbol{\theta}$ (a row vector instead of a column vector)

Linear Regression

- **Cost function**: Mean Squared error (MSE) for a Linear Regression model

$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\mathrm{T}} \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

Training the model is the process to find the value of θ that minimizes the cost function.

- **Normal Equation:**

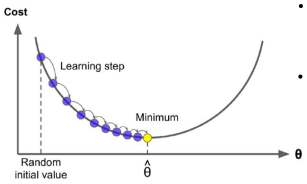
$$\frac{\partial \,\mathrm{MSE}(\mathbf{X}, h_{\theta})}{\partial \theta} = 0 \longrightarrow \hat{\boldsymbol{\theta}} = \left(X^{\top} X \right)^{-1} X^{\top} y$$

  - $\hat{\theta}$ is the value of θ that minimizes the cost function
  - y is the vector of targeted values containing $y^{(1)}$ *to* $y^{(m)}$

Gradient Descent

Minimize the Mean Squared error (MSE) cost function:

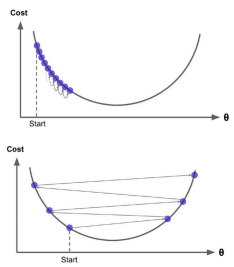
$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^{\top} \mathbf{x}^{(i)} - y^{(i)} \right)^2$$



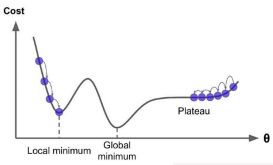
- Starting at the intimal location $\boldsymbol{\theta}_0 = [\theta_0, \theta_1 \ldots \theta_n]_0^{\top}$
- Calculate the gradient $\nabla_{\boldsymbol{\theta}} MES(\boldsymbol{\theta})$
- Iteratively apply $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \nabla_{\boldsymbol{\theta}} MES(\boldsymbol{\theta}_i)$ (η is the learning rate)

Gradient Descent

In appropriate Learning Rate



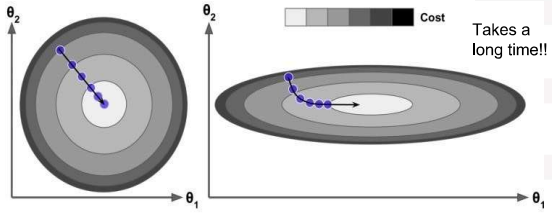
Local minimum and Plateau



The MSE cost function for a Linear Regression model is continuous and convex function.
Gradient Descent is guaranteed to approach arbitrarily close the global minimum.

Gradient Descent

Features with very different scales.



Takes a long time!!

Figure 4-7. Gradient Descent with (left) and without (right) feature scaling

Gradient Descent

- **Batch Gradient Descent (Full Gradient Descent)**

Use the whole training set to compute the gradients at every step.

$\left(\boldsymbol{\theta} = [\theta_0, \theta_1 \ldots \theta_n]^{\top} \right)$

$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^{\top} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

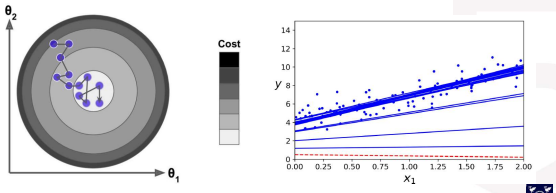
$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$$

Gradient Descent

- **Stochastic Gradient Descent (SGD)**

Stochastic Gradient Descent picks a random instance in the training set at every step and computes the gradients based only on that single instance.

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\theta} MES\left(\boldsymbol{\theta}; x^{(i)}, y^{(i)}\right)$$



Gradient Descent

- Mini-batch Gradient Descent

Mini-batch GD computes the gradients on small random sets of instances called mini-batches.

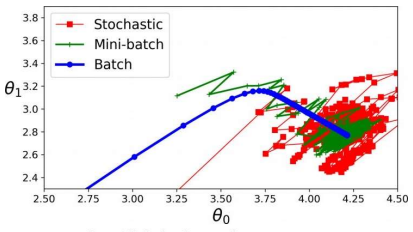


Figure 4-11. Gradient Descent paths in parameter space

Gradient Descent

- Summary

Method	Pros	Cons
Batch Gradient Descent	Guaranteed convergence to the global minimum.	Computationally expensive for large datasets. Requires the entire dataset to be loaded into memory.
Stochastic Gradient Descent	Computationally efficient for large datasets.	Convergence to the global minimum is not guaranteed. The noise in the updates can cause the loss function to oscillate.
Mini-Batch Gradient Descent	Better convergence than stochastic gradient descent. Computationally efficient for large datasets.	Convergence to the global minimum is not guaranteed.

Polynomial Regression

What if your data is more complex than a straight line?

$$\hat{y} = ax_1^2 + bx_1 + c \iff \hat{y} = ax_2 + bx_1 + c$$
$$x_2 = x_1^2$$

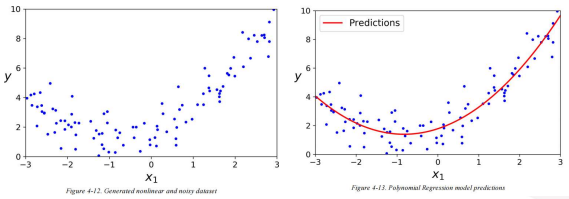
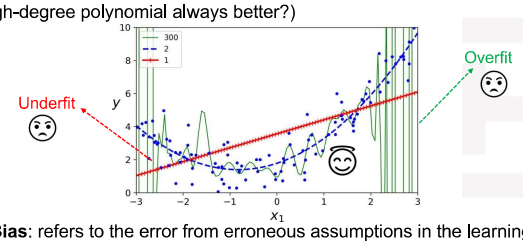


Figure 4-12. Generated nonlinear and noisy dataset

Figure 4-13. Polynomial Regression model predictions

Learning Curves

If you perform high-degree Polynomial Regression, you will likely fit the training data much better than with plain Linear Regression. (Is high-degree polynomial always better?)

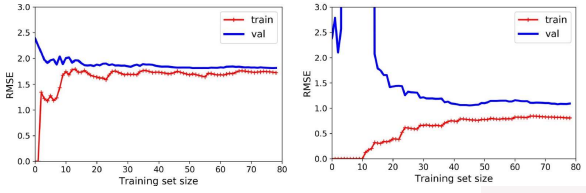


Bias: refers to the error from erroneous assumptions in the learning algorithm. (inability to capture the underlying patterns in the data).

Variance: refers an error from sensitivity to small fluctuations in the training data. (difference in fits between data sets)

Learning Curves

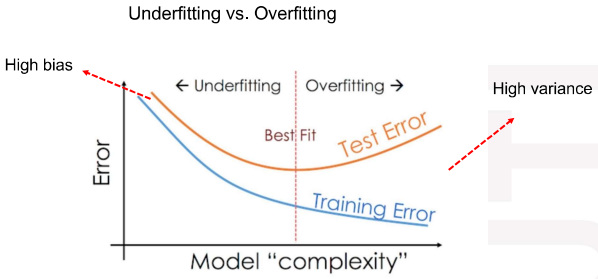
- Underfitting (1d) vs. Overfitting(300d)



Poor performance on training data and poor performance on validation data.

Good performance on training data but poor performance on validation data

Learning Curves

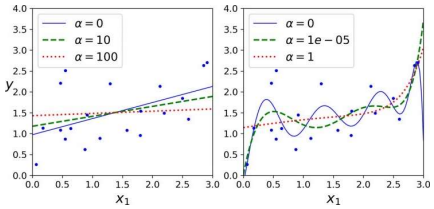


Regularized Linear Models

- **Ridge Regression(L2)**:

Cost function: $J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} {\theta_i}^2$

This forces the learning algorithm to not only fit the data but also keep the model weights as small as possible.



Regularized Linear Models

- **Lasso Regression(L1)**:

Cost function: $J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i|$

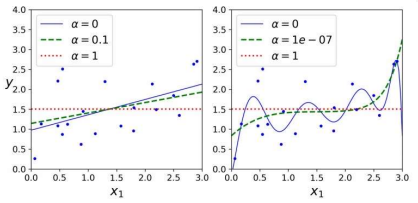


Figure 4-18. A linear model (left) and a polynomial model (right), both using various levels of Lasso regularization

Lasso Regression automatically performs **feature selection** and outputs a *sparse model*

Regularized Linear Models

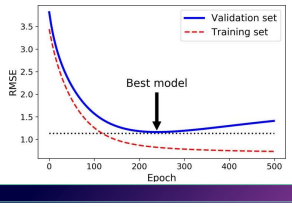
- Elastic Net:

Cost function: $J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} {\theta_i}^2$

It is a middle ground between Ridge Regression and Lasso Regression.

- Early stopping

To stop training as soon as the validation error reaches a minimum.



Logistic Regression

Logistic Regression: is commonly used to **estimate the probability** that an instance belongs to a particular class.

Logistic Regression model

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^{\intercal}\boldsymbol{\theta})$$

Sigmoid function:

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

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

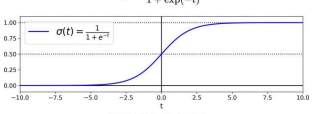


Figure 4-21. Logistic function

Logistic Regression (Softmax Regression)

Softmax Regression: for Multinomial Logistic Regression.

Softmax score for class k: $s_k(\mathbf{x}) = \mathbf{x}^{\intercal}\boldsymbol{\theta}^{(k)}$

Softmax function: $\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))}$

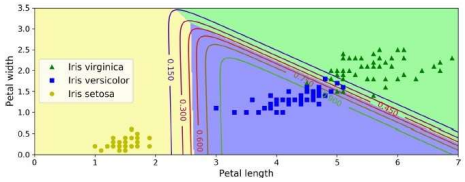


Figure 4-25. Softmax Regression decision boundaries