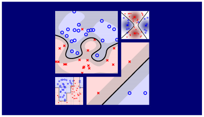
Machine Learning Techniques
(機器學習技法)



Lecture 16: Finale

Hsuan-Tien Lin (林軒田)

htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Roadmap

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 15: Matrix Factorization

linear models of movies on **extracted user features** (or **vice versa**) jointly optimized with **stochastic gradient descent**

Lecture 16: Finale

- Feature Exploitation Techniques
- Error Optimization Techniques
- Overfitting Elimination Techniques
- Machine Learning in Practice

Exploiting Numerous Features via Kernel

numerous features within some $\mathbf{\Phi}$:
embedded in kernel $K_{\mathbf{\Phi}}$ with inner product operation

Polynomial Kernel	Gaussian Kernel	Stump Kernel
'scaled' polynomial transforms	infinite-dimensional transforms	decision-stumps as transforms

Sum of Kernels	Product of Kernels	Mercer Kernels
transform union	transform combination	transform implicitly

	kernel ridge regression	kernel logistic regression
SVM	SVR	probabilistic SVM

possibly **Kernel PCA**, **Kernel *k*-Means**, ...

Exploiting Predictive Features via Aggregation

predictive features within some $\mathbf{\Phi}$:

$$\phi_t(\mathbf{x}) = g_t(\mathbf{x})$$

Decision Stump	Decision Tree	(Gaussian) RBF
simplest perceptron; simplest DecTree	branching (divide) + leaves (conquer)	prototype (center) + influence

---

Uniform	Non-Uniform	Conditional
Bagging; Random Forest	AdaBoost; GradientBoost	Decision Tree; Nearest Neighbor
	probabilistic SVM	

possibly **Infinite Ensemble Learning**, **Decision Tree SVM**, . . .

Exploiting Hidden Features via Extraction

hidden features within some $\mathbf{\Phi}$:

as hidden variables to be 'jointly' optimized with usual weights

—possibly with the help of **unsupervised learning**

Neural Network; Deep Learning	RBF Network	Matrix Factorization
neuron weights	RBF centers	user/movie factors

AdaBoost; GradientBoost	k-Means	Autoencoder; PCA
g_t parameters	cluster centers	'basis' directions

possibly **GradientBoosted Neurons**, **NNet on Factorized Features**, . . .

Exploiting Low-Dim. Features via Compression

low-dimensional features within some $\mathbf{\Phi}$:

compressed from original features

Decision Stump; DecTree Branching

'best' naïve projection to $\mathbb{R}$

Random Forest Tree Branching

'random' low-dim. projection

Autoencoder;PCA

info.-preserving compression

Matrix Factorization

projection from abstract to concrete

Feature Selection

'most-helpful' low-dimensional projection

possibly **other 'dimension reduction' models**

Fun Time

Consider running AdaBoost-Stump on a PCA-preprocessed data set. Then, in terms of the original features $\mathbf{x}$, what does the final hypothesis $G(\mathbf{x})$ look like?

1. a neural network with $\tanh(\cdot)$ in the hidden neurons
2. a neural network with $\text{sign}(\cdot)$ in the hidden neurons
3. a decision tree
4. a random forest

Fun Time

Consider running AdaBoost-Stump on a PCA-preprocessed data set. Then, in terms of the original features $\mathbf{x}$, what does the final hypothesis $G(\mathbf{x})$ look like?

1. a neural network with $\tanh(\cdot)$ in the hidden neurons
2. a neural network with $\text{sign}(\cdot)$ in the hidden neurons
3. a decision tree
4. a random forest

Reference Answer: 2

PCA results in a linear transformation of $\mathbf{x}$. Then, when applying a decision stump on the transformed data, it is *as if* a perceptron is applied on the original data. So the resulting G is simply a linear aggregation of perceptrons.

Numerical Optimization via Gradient Descent

when ∇E 'approximately' defined, use it for **1st order approximation**:

$$\text{new variables} = \text{old variables} - \eta \nabla E$$

SGD/Minibatch/GD

(Kernel) LogReg;

Neural Network [backprop];

Matrix Factorization;

Linear SVM (maybe)

Steepest Descent

AdaBoost;

GradientBoost

Functional GD

AdaBoost;

GradientBoost

possibly **2nd order techniques**,
GD under constraints, ...

Indirect Optimization via Equivalent Solution

when difficult to solve original problem,
seek for **equivalent solution**

Dual SVM	Kernel LogReg Kernel RidgeReg	PCA
equivalence via convex QP	equivalence via representer	equivalence to eigenproblem

some **other boosting models** and **modern solvers of kernel models** rely on such a technique heavily

Complicated Optimization via Multiple Steps

when difficult to solve original problem,
seek for **‘easier’ sub-problems**

Multi-Stage	Alternating Optim.	Divide & Conquer
probabilistic SVM;	k-Means;	decision tree;
linear blending;	alternating LeastSqr;	
stacking;	(steepest descent)	
RBF Network;		
DeepNet pre-training		

useful for **complicated models**

Fun Time

When running the DeepNet algorithm introduced in Lecture 213 on a PCA-preprocessed data set, which optimization technique is used?

1. variants of gradient-descent
2. locating equivalent solutions
3. multi-stage optimization
4. all of the above

Fun Time

When running the DeepNet algorithm introduced in Lecture 213 on a PCA-preprocessed data set, which optimization technique is used?

1. variants of gradient-descent
2. locating equivalent solutions
3. multi-stage optimization
4. all of the above

Reference Answer: 4

minibatch GD for training; equivalent eigenproblem solution for PCA; multi-stage for pre-training

Overfitting Elimination via Regularization

when model too 'powerful':

add brakes somewhere

large-margin

SVM;

AdaBoost (indirectly)

L2

SVR;

kernel models;

NNet [weight-decay]

voting/averaging

uniform blending;

Bagging;

Random Forest

denoising

autoencoder

weight-elimination

NNet

constraining

autoenc. [weights];

RBF [# centers];

pruning

decision tree

early stopping

NNet (any GD-like)

arguably **most important techniques**

Overfitting Elimination via Validation

when model too 'powerful':

check performance carefully and honestly

 # SV

SVM/SVR

OOB

Random Forest

Internal Validation

blending;

DecTree pruning

simple but **necessary**

Fun Time

What is the major technique for eliminating overfitting in Random Forest?

1. voting/averaging
2. pruning
3. early stopping
4. weight-elimination

Fun Time

What is the major technique for eliminating overfitting in Random Forest?

1. voting/averaging
2. pruning
3. early stopping
4. weight-elimination

Reference Answer: 1

Random Forest, based on uniform blending, relies on voting/averaging for regularization.

NTU KDDCup 2010 World Champion Model

Feature engineering and classifier ensemble for KDD Cup 2010,
Yu et al., KDDCup 2010

linear blending of

Logistic Regression +
many rawly encoded features

Random Forest +
human-designed features

yes, you've learned everything! :-)

NTU KDDCup 2011 Track 1 World Champion Model

A linear ensemble of individual and blended models for music rating prediction, Chen et al., KDDCup 2011

NNet, DecTree-like, and then linear blending of

- Matrix Factorization variants, including probabilistic PCA
- Restricted Boltzmann Machines: an 'extended' autoencoder
- *k* Nearest Neighbors
- Probabilistic Latent Semantic Analysis:
  an extraction model that has 'soft clusters' as hidden variables
- linear regression, NNet, & GBDT

yes, you can 'easily' understand everything! :-)

NTU KDDCup 2012 Track 2 World Champion Model

A two-stage ensemble of diverse models for advertisement ranking in KDD Cup 2012, Wu et al., KDDCup 2012

NNet, GBDT-like, and then linear blending of

- Linear Regression variants, including linear SVR
- Logistic Regression variants
- Matrix Factorization variants
- ...

'key' is to **blend properly without overfitting**

NTU KDDCup 2013 Track 1 World Champion Model

Combination of feature engineering and ranking models for paper-author identification in KDD Cup 2013, Li et al., KDDCup 2013

linear blending of

- Random Forest with many many many trees
- GBDT variants

with tons of efforts in designing features

'another key' is to **construct features with domain knowledge**

ICDM 2006 Top 10 Data Mining Algorithms

1. C4.5: another **decision tree**
2. *k*-Means
3. SVM
4. Apriori: for frequent itemset mining
5. EM: **'alternating optimization'** algorithm for some models
6. PageRank: for link-analysis, similar to **matrix factorization**
7. AdaBoost
8. *k* Nearest Neighbor
9. Naive Bayes: a simple **linear model** with 'weights' decided by data statistics
10. C&RT

personal view of five missing ML competitors:
LinReg, LogReg, Random Forest, GBDT, NNet

Machine Learning Jungle

bagging decision tree support vector machine **neural network** *kernel*
AdaBoost aggregation *sparsity* autoencoder **functional gradient**
dual uniform blending deep learning nearest neighbor decision stump
kernel LogReg large-margin *prototype* quadratic programming **SVR**
GBDT **PCA** `random forest` *matrix factorization* **Gaussian kernel**
soft-margin *k-means* OOB error **RBF network** probabilistic SVM

welcome to the **jungle**!

Fun Time

Which of the following is the official lucky number of this class?

1. 9876
2. 1234
3. 1126
4. 6211

Fun Time

Which of the following is the official lucky number of this class?

1. 9876
2. 1234
3. 1126
4. 6211

Reference Answer: 3

May the luckiness always be with you!

Summary

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 16: Finale

- Feature Exploitation Techniques
  kernel, aggregation, extraction, low-dimensional
- Error Optimization Techniques
  gradient, equivalence, stages
- Overfitting Elimination Techniques
  (lots of) regularization, validation
- Machine Learning in Practice
  welcome to the jungle

- **next: happy learning!**