Machine Learning Final Project Report

Golden Sword
電機五 B96502132 蘇培豪
電機所 R00921033 溫宗憲

1. Introduction

a) Data Preview:

The given dataset contains 20000 training data and 23907 testing data, in our first thought, this dataset shouldn't suffer much from the insufficient training data problem. Each data contains a 120-dimensional feature vector with all element scale from 0 to 1, showing that each feature can be treated equally without any normalization beforehand. 101 binary labels are introduced in each data with average number of 1-label equals to 4.48 and 0-label 96.52. This information signs us that this multi-label dataset is a binary classification problem with extremely unbalanced 0/1 label. Feature weighting and data selection are very common and useful solution for this classification problem.

b) Missing Feature

There is a very important characteristic of the given dataset, that is, the missing feature problem. The assumption made in this problem is that the maintenance of the system is not perfect thus there are approximately 10% features missed in the training set. As a consequence, methods to handle missing feature should be adopted to get a better prediction result. In our experiments, we also show that missing feature handling is a very important issue in this problem. Those algorithms without good handling may result in not-so-good prediction result.

2. Previous Work Survey

Some very similar work and dataset has been found on the Internet [1,2]. In their past experiences, some state-of-the-art multi-label classifiers such as ECC[3] and RAKEL[4] in which label correlation is modeled has no or little advantage over baseline binary relevance (BR) method. They also introduce that combining simple multi-label learning algorithm such as BR with strong single-label learning techniques such as Random Forest lead to a very competitive experiment result. In our following work, these references act as an important guideline in our final project algorithm selection.

3. Our Algorithms

In order to result in better performance, we try some various classification algorithms to form models for our task. First of all, we implement simple PLA algorithms, pocket PLA. The initial prediction result isn't as good as we expected, therefore we further implement

some popular and useful algorithms taught in class, e.g. Support Vector Machine, Random Forest. The following are the details of our algorithm description:

a) Perceptron Learning Algorithm

Since it is one of the simplest learning models in machine learning, we want to test its performance on the given dataset.

b) SVM

When using SVM, we find the support vectors to represent and calculate the model. Here we implement linear and Gaussian kernel:

1. Linear Kernel $\mathrm{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{x_i} \cdot \boldsymbol{x_j}$

   By using linear kernel we get a "rough" result of prediction with hamming score 18.44 and F1 score 0.26, which is pretty bad.

2. Gaussian RBF Kernel $\mathrm{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\gamma \parallel \boldsymbol{x_i} - \boldsymbol{x_j} \parallel^2)$

   Here the kernel is changed in to Gaussian RBF, which can result in better performance

c) Random Forest

Random Forest is the combination of bagging and Decision Tree. We pick samples randomly as well as features from original dataset to get bootstrapped datasets, decision tree is then implemented on these sets. Then the individual predictions are combined together by voting to generate the final result.

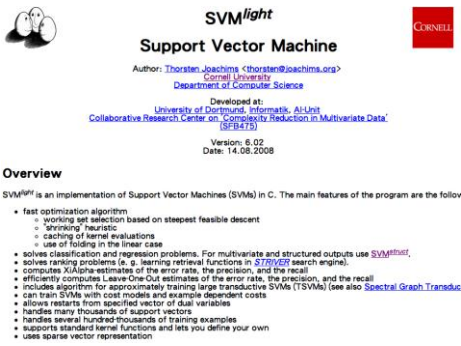
d) Aggregation

Aggregating hypotheses above improves the prediction slightly. Here we use uniform and non-uniform voting of the best

4. Package Used

a) SVM-Light:

SVM-Light [5] is an implementation of SVM in C developed and documented by and University of Dortmund and University of Cornell. It supports many state-of-the-art SVM operations and features as well and cross-platform. Our commands of SVM-Light training and testing are listed as follows:



SVM*light*

Support Vector Machine

Author: Thorsten Joachims <thorsten@joachims.org>
Cornell University
Department of Computer Science

Developed at:
University of Dortmund, Informatik, AI-Unit
Collaborative Research Center on 'Complexity Reduction in Multivariate Data' (SFB475)

Version: 6.02
Date: 14.08.2008

Overview

SVM*light* is an implementation of Support Vector Machines (SVMs) in C. The main features of the program are the following:

- fast optimization algorithm
  - working set selection based on steepest feasible descent
  - 'shrinking' heuristic
  - caching of kernel evaluations
  - use of folding in the linear case
- solves classification and regression problems. For multivariate and structured outputs use SVM*struct*
- solves ranking problems (e. g. learning retrieval functions in STRIVER search engine).
- computes XiAlpha-estimates of the error rate, the precision, and the recall
- efficiently computes Leave-One-Out estimates of the error rate, the precision, and the recall
- includes algorithm for approximately training large transductive SVMs (TSVMs) (see also Spectral Graph Transducer)
- can train SVMs with cost models and example dependent costs
- allows restarts from specified vector of dual variables
- handles many thousands of support vectors
- handles several hundred thousands of training examples
- supports standard kernel functions and lets you define your own
- uses sparse vector representation

./svm_learn –c XX –t XX [–g XX] –j XX training.dat model.mol

./svm_classify testing.dat model.mol prediction.prdt

Options:

-c : C: trade-off between training error and margin

-t: type of kernel function:

0: linear (default)

*1: polynomial (s a*b+c)^d*

2: radial basis function exp(-gamma ||a-b||^2)

*3: sigmoid tanh(s a*b + c)*

4: user defined kernel from kernel.h

-g: parameter gamma in rbf kernel

-j: Cost: cost-factor, by which training errors on positive examples outweight errors on negative examples

b) Mulan - Weka JavaKit:

Weka[5] is a collection of machine learning algorithms for data mining tasks. It includes several tools such for data mining purpose, such as pre-processing, classification, regression… etc. Several algorithms are developed inside Weka such as K-means, Association Rules, Decision Tree, Random Forest, and various Bayes methods…etc.

Mulan[6] is a Java multi-label learning algorithms toolkit based on Weka's basis. It basically transforms the multi-label problem into a corresponding single-label problem and applies various Weka's well-defined single-label classifier to solve the problem.

There're several methods that can transform a multi-label problem into a single-label problem. In our RF experiments we basically use Binary Relevance (BR) transformation only. Several transformations remain to be tried and report their performance over the given dataset. For further implementation and usage, please refer to the example shown in Mulan's online documentation.

5. Experiments

Our whole experiment can be divided into three sections:

First, since the given dataset contains unknown feature information, we use several methods to solve this problem.

1. Intuitively, we replace these unknown features with uniform random number between [0,1].
2. Furthermore, we also use Gaussian random number to replace the missing feature by calculating the same feature dimension of other training samples to find the mean(μ) and variance(σ) as our Gaussian variables, that is, Gauss(μ, σ)
3. Last but not least, we think that the unknown features may lead to significant

problem to the prediction. Therefore we decide **not to** take missing feature into account when forming our learning model. This mean is used in training random forest models.

Secondly, we implement several simple learning algorithms on the per-processed dataset. Here we focus on two popular models, SVM and Random Forest.

1. SVM

   There are lots of SVM tool in the Internet, here we choose SVM-light. First we implement linear kernel SVM, and the result is not as good as we expected. Then we further use Gaussian RBF kernel version, based on dataset with uniform random number replacing missing features. The result is better than previous one, by using γ = 50 C=1, it decreases the hamming loss from 18.44 to about 4.

   After our discussion, we decide to implement Gaussian RBF kernel SVM on second dataset, that is, the one with Gaussian random number replacing missing features. We found that the features in whole dataset scales between [0,1], so we set the variables γ= 2 (σ=0.5), and C =1. The result is slightly better. After this we try to tune the parameters γ and C to get the best one. Finally we get best hamming loss 3.2986 and F1 score 0.5746. But there is one main problem, the biggest disadvantage we encountered is that the training time complexity is pretty high due to large training dataset.

2. Random Forest

   Random forest algorithm works very different from SVM, we use java package Mulan and Weka to implement RF. Since Mulan is a multi-label learning algorithms toolkit, after changing training data into its readable input, we can easily set how many trees to divide and how many features to form a bagging feature set.

   To begin with, we set 10 trees and 10 features on different replaced-feature datasets. Surprisingly the training time takes less than an hour, and the resulting hamming loss decreases about 0.1 compared to SVM's best result. The thought is that because it randomly chooses fixed number of features, it may somehow avoid choosing too many "artificial" features created by random generator. We then try to "plant" more trees and select more features. Finally we find 65 trees and 35 selected features for bagging got best prediction result.

   On the other hand, we find that Weka has a very powerful feature-selection mechanism, it can avoid choosing missing feature. Therefore we don't need to handle the missing data problem, the pre-process for the training dataset is no need anymore. Simply set missing features as "?", Weka package can ignore the missing data, rather choosing other features. On the basis of this, we re-run many random forest variables, getting the best result.

Thirdly, after training lots of hypotheses above, we try to combine them by voting

linearly and non-linearly. We combine the five best SVM hypotheses, getting a slight better prediction than these five. Combining five best RF results also get better result.

Finally we try to combine two different algorithms' hypotheses. Since the prediction result of Random Forest's outperform SVM's, we couldn't get a better combined prediction. But the individual combination of each algorithms shows that aggregation is indeed a way to increase the performance.

Best Hamming Loss

Algorithm	Hamming Loss
SVM Linear kernel	18.4436
SVM Gauss.-RBF kernel (γ=0.5,C=10000)	3.2986
Random Forest (65 Trees, 35 Features)	3.0794
Combine SVM and RF	3.1934

Best F1 Score

Algorithm	F1 Score
SVM Linear kernel	0.2590670245
SVM Gauss.-RBF kernel (γ=0.5,C=10000)	0.5746316012
Random Forest (65 Trees, 35 Features)	0.591861412
Combine SVM and RF	0.5479715766

6. Conclusion & Analysis

To sum up, handling missing data is a critical point in the given dataset. Try to recover missing data from the implicit distribution in the backend is not practical and experiments suggest that the recovery mechanism, if not handled well, is worse than without any recovery. As a result, SVM classifier in this case, performs generally not as amazing as other tasks we have encountered before. However, other algorithms that can explicitly handle missing features such as Random Forest have a very impressive performance on the given dataset. The reason should be that with its missing feature handling capabilities, one doesn't need to make any assumptions on the missing data and thus won't suffer the distortion caused by wrongly-recovered missing features.
The parameters in Random Forest should be carefully selected. In our investigation, the number of trees should grow with the growth of features, otherwise the performance will degrade. General speaking, the growth number of trees with fixed number of features will gradually increase the performance at early phase and then saturate at some optimal point afterward. However, the monotonous increase of number of features doesn't guarantee to converge to an optimal point. The selection of the two parameters should be carefully tuned to get a good prediction result.

The following table is the brief summarization we make:

Characteristic	PLA	SVM	RF
Efficiency	A	C	A
Scalability	C	C	A
Popularity	C	A	A
Interpretability	B	C	A
Separability	C	A	A
Generalizability	C	B	A
Pros	Easy to implement	Sound mathematical foundation Easy to implement Handle linearly inseparable data	Handle redundant features **Handle missing data** Less preprocessing Great efficiency
Cons	Long training time Bad on noised dataset	Long computation Over-fitting Trial and error on parameters	Over-fitting given noisy data

7. Collaboration

The division of our work is equally distributed, the following is the work we done:

蘇培豪:

Training data preprocessing, PLA, SVM implementations, SVM and Random Forest variable tuning, report writing 3,5,6.

溫宗憲:

Training data preprocessing, SVM, Random Forest implementations, SVM and Random Forest variable tuning, report writing 1,2,4,6.

8. Reference

[1] Winner's notes. Eleftherios Spyromitros – Xioufis on Music Instruments Recognition. http://blog.tunedit.org/

[2] Winners' notes. CNSlab team on music instruments recognition. http://blog.tunedit.org/2011/04/22/winners-notes-cnslab-instruments-recognition/

[3] Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. In: Proceedings of ECML PKDD 2009, Bled, Slovenia, pp. 254–269

[4] Tsoumakas, G., Katakis, I., Vlahavas, I.: Random k-labelsets for multi-label classification. IEEE Transactions on Knowledge and Data Engineering (2011)

[5] Weka. http://www.cs.waikato.ac.nz/ml/weka/

[6] Mulan. http://mulan.sourceforge.net/index.html