Machine Learning

L715/B659
(with many thanks to Sandra Kübler!)

Dept. of Linguistics, Indiana University
Fall 2014
Some Available Packages

Packages of multiple algorithms:

- weka (http://www.cs.waikato.ac.nz/ml/weka/)
- mallet (http://mallet.cs.umass.edu): MAchine Learning for LanguagE Toolkit
- MLlib (https://spark.apache.org/mllib/): good for very large-scale learning
- MLPack (http://www.mlpack.org): “emphasis on scalability, speed, and ease-of-use”
- sofia-ml (https://code.google.com/p/sofia-ml/)

Some specific algorithms

- timbl (http://ilk.uvt.nl/timbl/): memory-based learning
- svmlight (http://svmlight.joachims.org)
Other packages I failed to list the other day

- Python Machine Learning kits:
  - Orange: http://orange.biolab.si/
  - mlpy: https://mlpy.fbk.eu/
  - PyML: http://pyml.sourceforge.net/
  - Anaconda Python has much pre-installed: https://store.continuum.io/cshop/anaconda/

- Various packages in R (http://cran.r-project.org/web/views/MachineLearning.html)


- ...
Example: sofia-ml

I encountered sofia-ml recently in a project, so we’ll look at it as one example package:

- https://code.google.com/p/sofia-ml/

It works well for “fast methods for classification and ranking on large, sparse data sets” & offers:

- Pegasos SVM
- Stochastic Gradient Descent (SGD) SVM
- Passive-Aggressive Perceptron
- Perceptron with Margins
- ROMMA
- Logistic Regression (with Pegasos Projection)
Example: sofia-ml

Obtaining & Compiling the code

From the webpage:

1. Check out the code (requires svn):
   
   > svn checkout http://sofia-ml.googlecode.com/svn/trunk/sofia-ml

2. Compile the code:

   > cd sofia-ml-read-only/src/
   
   > make

   > ls ../sofia-ml

   # Executable should be in main sofia-ml-read-only
sofia-ml
Demo: training

> ./sofia-ml --learner_type pegasos --loop_type stochastic --lambda 0.1 --iterations 100000 --dimensionality 150000 --training_file demo/demo.train --model_out demo/model
Reading training data from: demo/demo.train
Time to read training data: 0.023163
Time to complete training: 0.05237
Writing model to: demo/model
Done.
sofia-ml

Training options: --learner_type

- **pegasos**: Pegasos SVM learning (default)
  - --lambda: regularization parameter (values closer to zero giving less regularization)
  - --eta_type also needed
- **sgd-svm**: SGD-SVM learning algorithm
  - --lambda & --eta_type
- **passive-aggressive**: Passive Aggressive Perceptron learning
  - --passive-aggressive-c: set largest step size for update step (≈ 0 encourages simpler models)
  - --passive-aggressive-lambda: force model weight vector to lie within certain value
choosing a classifier

sofia-ml

Training options: --learner_type (cont.)

- margin-perceptron: Perceptron with Margins
  - --perceptron-margin-size: set update margin
    (0 = classic Perceptron; high values tolerate noise)
  - --eta_type

- romma ROMMA algorithm: No parameters to set.

- logreg-pegasos Logistic Regression with Pegasos updates
  - --lambda & --eta_type
Aside: SVMs

Support Vector Machines (SVMs) are quite popular

- Roughly speaking, an SVM creates a hyperplane to separate the two classes (positive & negative)
  - points (vectors) nearest the hyperplane are the support vectors
  - large-margin: create a wide separation, so that points of low-confidence are not decided upon

Pegasos (Primal Estimated sub-GrAdient SOlver) SVMs use one particular method for solving the SVM optimization problem in training

sofia-ml

Training options: --loop_type

Controls how examples are selected for training

- **stochastic** (default): normal stochastic sampling for stochastic gradient descent
  - On each iteration, pick a new example uniformly at random from the data set
- **balanced-stochastic**: balanced sampling from positives & negatives in data set
  - On each iteration, sample one positive & one negative example
  - Useful for training binary classifiers with a minority-class distribution
- **rank**: indexed sampling of candidate pairs for pairwise learning to rank
  - Useful when there are examples from several different qid groups
Training options: --loop_type (cont.)

- **roc** indexed sampling to optimize ROC Area
- **query-norm-rank**: sampling of candidate pairs, with equal weight to each qid group (regardless of size)
- **combined-ranking**: combined regression and ranking (CRR)
  - Alternates between pairwise rank-based steps & standard stochastic gradient steps on single examples
  - Relies on --rank_step_probability to balance between the two kinds of updates
- **combined-roc** CRR algorithm for combined regression & ROC area optimization.
  - Relies on --rank_step_probability
sofia-ml

Training options: --eta_type

Update for learning rate

- **basic**: on $i^{th}$ iteration, learning rate $\eta = \frac{1000}{(i+1000)}$
- **pegasos** *(default)*: on $i^{th}$ iteration, learning rate $\eta = \frac{1}{(i\cdot\lambda)}$
- **constant** learning rate $\eta$ always: $\eta = 0.02$
Training options: --prediction_type

- **linear** *(default)*: linear dot product $\langle w, x \rangle$
- **logistic**: prediction function $\frac{\exp(\langle w, x \rangle)}{1+\exp(\langle w, x \rangle)}$

There are other options available, but this is a good start
sofia-ml

Demo: testing

> ./sofia-ml --model_in demo/model \ 
   --test_file demo/demo.train \ 
   --results_file demo/results.txt
Reading model from: demo/model
  Done.
Reading test data from: demo/demo.train
Time to read test data: 0.023146
Time to make test prediction results: 0.000251
Writing test results to: demo/results.txt
  Done.
SVM-light sparse data format, examples:

# Class label is 1, feature 1 has value 1.2, 
# feature 2 (not listed) has value 0, 
# and feature 3 has value -0.5.  
1 1:1.2 3:-0.5

# Class label is -1, belongs to qid 3, and  
# all feature values are zero except  
# for feature 5011 with value 1.2.  
-1 qid:3 5011:1.2

# Class label is -1, feature 1 has value 7, comment  
# string is "This example is especially interesting."  
-1 1:7 3:-0.5 #This example is especially interesting.
sofia-ml

Notes on data format

Features:
- Feature IDs should be in ascending numerical order
- Lowest allowable feature ID is 1
- Any unspecified feature has value 0 (sparse representation)

Class labels:
- For binary-class classification: use 1 and -1
- For ranking problems: the labels may be any numeric value, with higher values judged as "more preferred"
- For testing where values are not known: okay to put in a dummy placeholder value such as 0

Query IDs (qid):
- A way to quasi-group data points (from IR uses)
Demo: evaluation

> perl eval.pl demo/results.txt

Results for demo/results.txt:

Accuracy 1.0000 (using threshold 0.00) (1000/1000)
Precision 1.0000 (using threshold 0.00) (322/322)
Recall 1.0000 (using threshold 0.00) (322/322)
ROC area: 1.000000

Total of 1000 trials.

Only works if you have gold standard categories in the file, of course
TiMBL is relatively popular for some language tasks:

- text-based features, including options for edit distance between features
- easy to use
- conceptually pretty clear (memory-based learning)
  - training: store all instances
  - testing: find the $k$ nearest vectors (distances) to new case

It can be rather slow, however
Memory-based learning (MBL) is a form of nearest neighbors classifying:

- all training instances are stored in a database
- new instances are compared to those in the database to see which are the closest
  - they are given the most likely label from the nearest neighbors stored in memory

There are a variety of ways to score neighbors.
Obtaining TiMBL

1. Go to: http://ilk.uvt.nl/timbl/
2. Click on the Download link
   ▶ A very useful document on this page is the Reference Guide
3. Unpack the file in a place you want to keep it
4. Go through the directions in the INSTALL file to install timbl
   ▶ Need various dependent packages: autoconf, autoconf-archive, pkg-config, libtool, libxml2-dev, & ticcutils
Running TiMBL

If I type `timbl` at the terminal, I get something like this:

TiMBL 6.0 (Release) (c) ILK 1998 - 2007.
Tilburg Memory Based Learner
Induction of Linguistic Knowledge Research Group
Tilburg University / University of Antwerp

usage: Timbl -f data-file {-t test-file}
or see: Timbl -h
for all possible options

i.e., to run `timbl`:
`timbl -f training-data -t testing-data`
Exploring TiMBL’s options

You can run timbl with a host of different options. See chapter 6 of the reference guide.

Main options:

- `-a` sets the classification algorithm
- `-m` sets the distance metric (and can also be used to ignore features or define features as numeric)
- `-w` sets the feature weighting possibilities
- `-k` sets the number of nearest neighbors

Many of the output options are useful at showing more of what happened inside timbl in order to classify instances.
Data formats

The main formats are:

- Column format: white space between features
- C4.5 format: commas between features
  - Look at demos/dimin.train and demos/dimin.test

If you need to refer to a space or a comma in your data, you should recode it as something else, e.g., `<COMMA>`

The last element you put on a line is the class, i.e., the value you’re trying to predict
Integrating TiMBL into programs

Information on the C++ API is at:


You could also try putting timbl into a python program:


This will allow you to classify individual instances without relearning every time.
Practice problem with sofia-ml

Set up a simple genre classifier using sofia-ml and/or timbl

1. Download the SUSANNE corpus (http://www.grsampson.net/Resources.html)
   ▶ This will unpack into an fc2/ directory

2. Use the first 8 files of each genre (A, G, J, N) as training, next 4 as development, final 4 as testing
   ▶ A = press reportage
   ▶ G = belles lettres, biography, memoirs
   ▶ J = learned (mainly scientific and technical) writing
   ▶ N = adventure and Western fiction

3. Extract what seem to be relevant features
   ▶ Columns: 3 = POS tag (class); 4 = word; 5 = lemma, 6 = syntactic functional information
   ▶ Hand-examine some files first ...

4. Classify, tweaking parameters & options
SUSANNE is a **corpus**: a collected body of text

- Each line corresponds to a word, with many other properties associated with it
- i.e., if you read it vertically, you can see what the text is (try `cut -f4 FILENAME` to get just the plain text)

This is a linguistically **annotated** corpus

- Someone has gone through and added part-of-speech & syntactic information (by hand)
- Most of our data will not be so nicely hand-annotated
  - But: we’ll have automatic tools to give us much of this functionality
Input: SUSANNE file

A01:0010.03 - YB <minbrk> - [Oh.Oh]
A01:0010.06 - AT The the [O[S[Nns:s.
A01:0010.09 - NPL1s Fulton Fulton [Nns.
A01:0010.12 - NNL1cb County county .Nns]
A01:0010.15 - JJ Grand grand 
A01:0010.18 - NN1c Jury jury .Nns:s]
A01:0010.21 - VVDv said say [Vd.Vd]
A01:0010.24 - NPD1 Friday Friday [Nns:t.Nns:t]
A01:0010.27 - AT1 an an [Fn:o[Nns:s.
A01:0010.30 - NN1n investigation investigation .
A01:0020.03 - IO of of [Po.
A01:0020.06 - NPl1t Atlanta Atlanta [Ns[G[Nns.Nns]
A01:0020.09 - GG +<apos>s - .G]
A01:0020.12 - JJ recent recent 
A01:0020.15 - JJ primary primary 
A01:0020.18 - NN1n election election .Ns]Po]
A01:0020.21 - VVDv produced produce [Vd.Vd]
A01:0020.24 - YIL <ldquo> - .
A01:0020.27 - ATn +no no [Ns:o.
A01:0020.30 - NN1u evidence evidence .
What is the task?

**Task:** determine which of 4 categories a new document falls into

- This means that analysis is on a *per document* level
- i.e., each feature **vector** refers to a whole document

Note that sofia-ml does binary classification, while timbl allows for n-ary classification

- This affects what the **class** is

We can set it up as follows:

- timbl: class $\in \{A, G, J, N\}$
- sofia-ml:
  - 4 one-vs-all classifiers? e.g., for one classifier: class $\in \{A, \text{not-A}\}$
  - 6 one-vs-one classifiers? e.g., A vs. G, A vs. J, etc.
  - Q: how would a class ultimately be determined?
Feature exploration?

What types of features could be relevant for genre classification?

- (normalized) counts of pronouns?
- average sentence length? word length?
- (normalized) punctuation counts?
- (normalized) counts of all content words?
- measure of lexical diversity (e.g., type-token ratio)?
- ...

...
Output

Let’s say we want to have these features:

1. count(*he*)
2. count(*she*)
3. count(*you*)
4. count(*I*)

Representations for file A01 would be (in A-vs-all setting):

- sofia-ml: 1 1:7 4:3
- timbl: 7, 0, 0, 3, A

Representations for file G01 would be:

- sofia-ml: -1 1:6 4:19
- timbl: 6, 0, 0, 19, G
Obtaining features

**Question:** How do we go from SUSANNE files to output representation?

- **Answer:** Use your favorite scripting/programming language!
  - Feel free to share corpus-reading code on the oncourse wiki (or the CL wiki)

Unix tricks can help

- **e.g.,** `cut -f4 G01 | grep -ci '^he$'` gives 6 as the count of *he* in file G01
- See Kenneth Church’s *Unix for Poets* ([http://www.lsi.upc.edu/~padro/Unixforpoets.pdf](http://www.lsi.upc.edu/~padro/Unixforpoets.pdf))
Obtaining features (2)

For sofia-ml (and similar representations), consider if you wanted to use the count of every known word as a feature

- Every feature is assigned a number, so in training you have to assign a number to each word
- In testing, then, you have to be able to read the same mapping, to assign features
More complicated features

How would we obtain/encode these features?

- Counts of bigrams of tags - i.e., two-tag sequences (e.g., AT NP1s, NP1s NNL1cb, etc.)
- The most frequent tag in the document (e.g., NN1n)
- The 10 most frequent words in the document
- Type-token ratio
  - type = abstract idea of a word
  - token = actual instance (e.g., 7 word tokens of the word type *he* in A01)
Feature selection

How do you know which features are helping or hurting?

- Some systems (e.g., timbl) will provide output indicating which features are treated as more important
  - e.g., using information gain to calculate
- You can try ablation experiments on development data
  - Remove a feature or set of features & observe new classification accuracy
  - And/or build feature sets from the ground up
Choosing a classifier

Some questions to ask from the practical perspective:
▶ Does it support the type of classification you want? (e.g., multi-class)
▶ Does it support the types of features you want? (e.g., categorical)
▶ Does it support trying different algorithms?
▶ Is it easy to install? Easy to use?
▶ Is it fast?
▶ Is it well-supported?
Choosing a classifier

Bias & Variance

- Bias: (non-)ability to approximate the data, degree to which model makes assumptions about data distribution
  - “High bias is related to under-fitting” (e.g., linear regression for a quadratic relationship)
- Variance: (non-)stability in the face of new training examples
  - “High variance is related to over-fitting” (e.g., $k$-NN changes a lot depending on training set)
- Regularization parameters help control bias-variance tradeoff
  - e.g., penalize complex models, set $k$ higher

(http://followthedata.wordpress.com/2012/06/02/practical-advice-for-machine-learning-bias-variance/)
Choosing a classifier
Accounting for bias & variance

What to do? (originally from Andrew Ng)

▶ High variance?
  ▶ get more training examples
  ▶ try smaller sets of features

▶ High bias?
  ▶ try new (more/different) features

To find whether it’s high variance or bias, compare training & testing (i.e., development) learning curves:

▶ cf. slides 7 & 8 of:

(http://followthedata.wordpress.com/2012/06/02/practical-advice-for-machine-learning-bias-variance/)
Choosing a classifier

Some advice from Manning, Raghavan, & Schütze (2008):

- Can use rules, if the task is relatively simple
  - Rules can also work well for post-processing ML output
- If you have little data, use a classifier with high bias (e.g., Naive Bayes)
  - Also can try semi-supervised training methods (e.g., bootstrapping, EM)
- If you have a huge amount of data, the classifier may have less of an impact on accuracy
  - Speed & ease of use become bigger questions

Other ways to boost performance

- Combine multiple classifiers
- Engage in feature engineering
  - Group together similar features (i.e., with similar votes)
  - Create new features for potentially informative concepts (e.g., subwords)
- Account for document structure (e.g., upweight items in a title, use separate feature spaces for different document zones)