Behavioral Data Mining

Lecture 3

Naïve Bayes Classifier
and Generalized Linear Models

Outline

- Naïve Bayes Classifier
- Regularization in Linear Regression
- Generalized Linear Models
- Assignment Tips:
  - Matrix Representation

Bayes' Theorem

$$P(e|D) = \frac{P(D|e)P(e)}{P(D)}$$

P(e) is called the prior probability of e. Its what we know (or think we know) about e with no other evidence.

P(D|e) is the conditional probability of D given that e happened, or just the likelihood of D. This can often be measured or computed precisely – it follows from your model assumptions.

P(e|D) is the posterior probability of e given D. It's the answer we want, or the way we choose a best answer.

You can see that the posterior is heavily colored by the prior, so Bayes' has a GIGO liability. e.g. its not used to test hypotheses

Naïve Bayes Classifier

We assume we have a set of documents, and a set of classification classes. Then the probability that a document d lies in class c is

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

Where $P(t_k|c)$ is the probability that the kth term (word) appears in a document in class c. $P(c)$ is the prior for class c. n_d is the number of terms in document d.

This simple form follows from the assumption that the $P(t_k|c)$ are independent.
Since that's almost never true, this is a "naïve" assumption.

Classification

For classification, we want the most likely class given the data.
This is the Maximum A-Posteriori estimate, or MAP estimate.

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k|c).$$

Where the $\hat{P}$ denotes an estimate of P.
Things will start to simplify when we take logs:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)].$$

And then notice that the second term can be computed with a vector dot product.

Classification

i.e. if we compute log P(t|c) for all terms, and then take an inner product with a (0,1) sparse vector for terms occurring in the document, we get the RHS sum.

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)].$$

Note: for fast classification, you can insert the log P(t|c) vectors as "documents" in a search engine that supports term weights. Then using a new document as a query, the engine will retrieve a likelihood-sorted list of classes. The first one is the best match.

Best and Worst of NB Classifiers

- Simple and fast. Depend only on term frequency data for the classes. One shot, no iteration.
- Very well-behaved numerically. Term weight depends only on frequency of that term. Decoupled from other terms.
- Can work very well with sparse data, where combinations of dependent terms are rare.
- Subject to error and bias when term probabilities are not independent (e.g. URL prefixes).
- Can't model patterns in the data.
- Typically not as accurate as other methods*.

* This may not be true when data is very sparse.

Parameter Estimation

For the priors, we can use a maximum likelihood estimate (MLE) which is just:

$$\hat{P}(c) = \frac{N_c}{N},$$

where N_c is the number of training docs in class c, N' total number of training docs.

For the $\hat{P}(t|c)$ we have:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}},$$

where T_{ct} is the number of occurrences of term t in training documents from class c, V is the term vocabulary.

Bernoulli vs Multinomial models

There are two ways to deal with repetition of terms:

- Multinomial models: distinct placements of terms are treated as independent terms with the same probability
- Bernoulli: Placements are not counted. Term presence is a binary variable

$$\begin{aligned} \textbf{Multinomial} \quad & P(d|c) = P(\langle t_1, \ldots, t_k, \ldots, t_{n_d} \rangle | c) \\ \textbf{Bernoulli} \quad & P(d|c) = P(\langle e_1, \ldots, e_i, \ldots, e_M \rangle | c) \end{aligned}$$

- The multinomial formula nd terms where nd is the length of the doc. Bernoulli has M terms where M is the number of distinct terms in the doc.

Bernoulli vs Multinomial models

- Multinomial: regarded as better for long docs

- Binomial: better for short docs (Tweets? SMS?). Also often better for web browsing data: counts are less meaningful because of Browser refresh, back button etc.

Dealing with Zero Counts

Its quite common (because of power law behavior) to get T_{ct} counts which are zero in the training set, even though those terms may occur in class documents outside the training set.

Just one of these drives the posterior estimate for that document/class to zero, when in fact it may be a good choice.

You can avoid zero probability estimates by smoothing.

Laplace Smoothing

Or "add-one" smoothing:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B'}$$

Where B is |V| the vocabulary size.

Rationale: Analyze the counts as an explicit probabilistic process (Dirichlet/multinomial), and find the expected value of the model parameter, instead of the MLE estimate.

Additive Smoothing

In practice, its worth trying smaller values of the additive smoothing parameter. Typically a positive value $\alpha < 1$ is used

$$\hat{P}(t|c) = \frac{T_{ct} + \alpha}{\sum_{t' \in V}(T_{ct'} + \alpha)} = \frac{T_{ct} + \alpha}{(\sum_{t' \in V} T_{ct'}) + B\ \alpha}$$

This is a compromise between the expected and "most likely" parameter for the count model.

Results

UK

london	0.1925
uk	0.0755
british	0.0596
stg	0.0555
britain	0.0469
plc	0.0357
england	0.0238
pence	0.0212
pounds	0.0149
english	0.0126

China

china	0.0997
chinese	0.0523
beijing	0.0444
yuan	0.0344
shanghai	0.0292
hong	0.0198
kong	0.0195
xinhua	0.0155
province	0.0117
taiwan	0.0108

poultry

poultry	0.0013
meat	0.0008
chicken	0.0006
agriculture	0.0005
avian	0.0004
broiler	0.0003
veterinary	0.0003
birds	0.0003
inspection	0.0003
pathogenic	0.0003

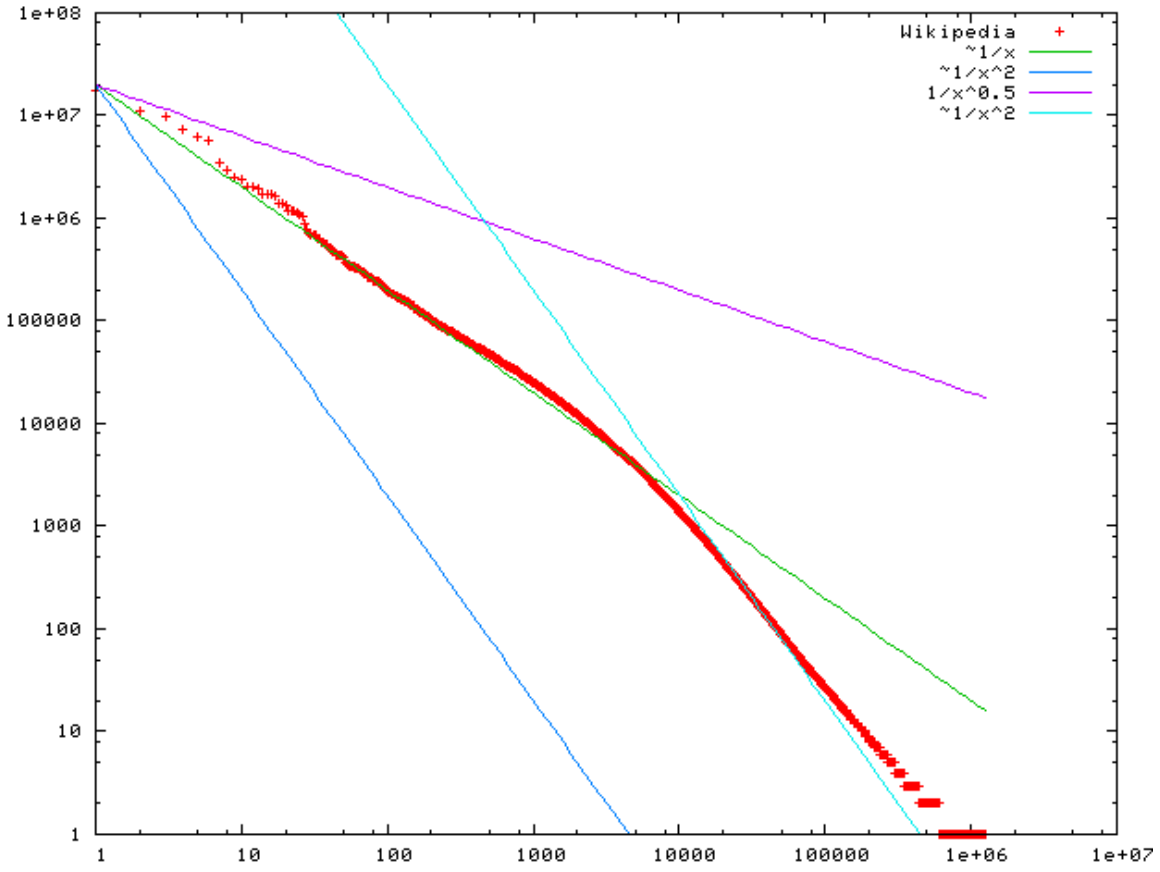
First Assignment

- Train and evaluate a naïve Bayes classifier on sentiment dataset: a collection of movie reviews with positive/negative tags.
- Run it and measure accuracy on a disjoint sample of the same dataset (cross-validation).

- The result will be usable as a sentiment weight vector for marking up social media posts.

Feature Selection

- Word frequencies follow power laws:



- Wikipedia: word rank on x, frequency on y

Feature Selection

- One consequence of this is that vocabulary size grows almost linearly with the size of the corpus.
- A second consequence is that about half the words occur only once.
- Rare words are less helpful for classification, most are not useful at all.
- Feature selection is the process of trimming a feature set (in this case the set of words) to a more practical size.

Mutual Information

Mutual information measures the extent to which knowledge of one variable influences the distribution of another.

$$I(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U = e_t, C = e_c) \log_2 \frac{P(U = e_t, C = e_c)}{P(U = e_t)P(C = e_c)}$$

Where U is a random variable which is 1 if term e_t is in a given document, 0 otherwise. C is 1 if the document is in the class c, 0 otherwise. These are called indicator random variables.

Mutual information can be used to rank terms, the highest will be kept for the classifier and the rest ignored.

CHI-Squared

CHI-squared is an important statistic to know for comparing count data.

Here it is used to measure dependence between word counts in documents and in classes. Similar to mutual information, terms that show dependence are good candidates for feature selection.

CHI-squared can be visualized as a test on contingency tables like this one:

	Right-Handed	Left-Handed	Total
Males	43	9	52
Females	44	4	48
Total	87	13	100

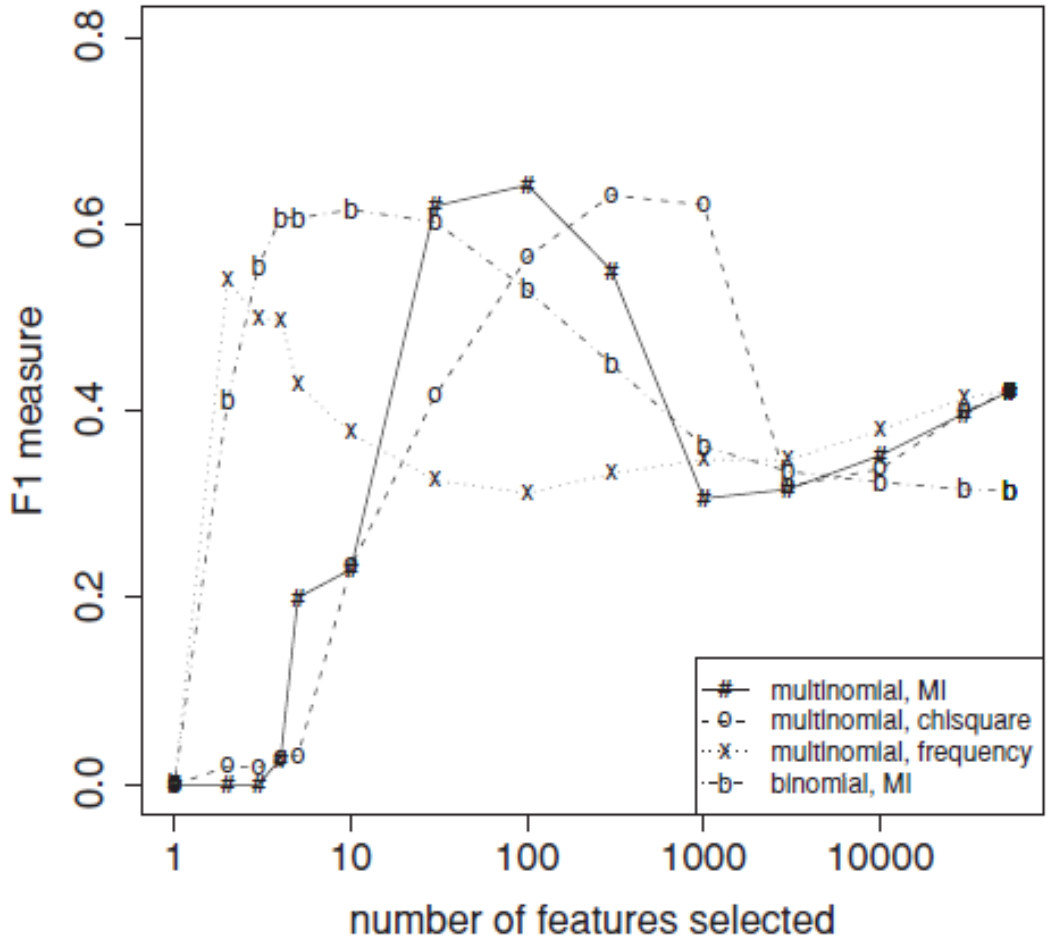
CHI-Squared

The CHI-squared statistic is:

$$X^2(\mathbb{D}, t, c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}}$$

It measures the extent to which the actual counts in the table are different from the "expected" counts which assume independence.

Feature Set Size and Accuracy



Bayes Classifier Summary

- Bayes theorem allows us to adjust beliefs about hidden events from data.
- Naïve Bayes classifiers use simple generative models (multinomial or bernoulli) to infer posterior probabilities of class membership.
- Naïve Bayes assumes independence between term probabilities.
- NB is fast, simple and very robust.
- Feature selection can drastically reduce feature set size, and improve accuracy.

Outline

- Naïve Bayes Classifier
- Regularization in Linear Regression
- Generalized Linear Models
- Assignment Tips:
  - Matrix Representation

Regularizers – Ridge and Lasso

When solving a linear regression problem $\hat{Y} = \hat{\beta}\mathbf{X}$, the solution $\hat{\beta} = \mathbf{y}\,\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$ is undefined if $(\mathbf{X}\mathbf{X}^T)^{-1}$ is singular.

This is likely for example, with dependent features (trigrams and bigrams).

There still is a solution for $\hat{\beta}$, in fact a linear space of solutions.

We can make the solution unique by imposing additional conditions, e.g.

Ridge regression: minimize L_2–norm of β. The solution is

$$\hat{\beta} = \mathbf{y}\,\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \boldsymbol{\lambda}\boldsymbol{I})^{-1}$$

The matrix in parentheses is generically non-singular – non-singular for almost any value of λ.

Regularizers

Regularization can be viewed as adding a probability distribution on X (independent of Y).

Minimizing the L_2–norm of X is equivalent to using a mean-zero Normal distribution for $\Pr(X)$.

The regression problem can be formulated as a Bayesian estimation problem with $\Pr(X)$ as a prior.

The effect is to make the estimate conservative – smaller values for β are favored, and more "evidence" is needed to produce large coefficients for β.

Can choose the regularizer weight (λ) by observing the empirical distribution of β after solving many regression problems, or simply by optimizing it during cross-validation.

Lasso

Lasso regression: minimize L_1–norm of β.

Typically generates a sparse model since the L_1 norm drives many coefficients to zero.

The function to minimize is

$$(\mathbf{y} - \beta\mathbf{X})^T(\mathbf{y} - \beta\mathbf{X}) + \lambda|\beta|$$

and its derivative is

$$(\mathbf{y} - \beta\mathbf{X})\mathbf{X}^T + \lambda\text{sign}(\beta)$$

An widely-used solution is Least-Angle Regression (LARS) due to Hastie. Another option is stochastic gradient.

Coefficient tests

Regularization helps avoid over-fitting, but many coefficients will still carry "noise" values that will be different on different samples from the same population.

To remove coefficients that are not "almost surely" non-zero, we can add a significance test. The statistic is:

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{\upsilon_j}}$$

where υ_j is the j^{th} diagonal element of $(\mathbf{X}\mathbf{X}^T)^{-1}$, and σ^2 is an estimate of the error variance (mean squared difference between Y and βX).

Significant Coefficients

The z value has a t-distribution under the null hypothesis ($\beta_j = 0$) but in most cases (due to many degrees-of-freedom) it is well-approximated as a normal distribution.

Using the normal CDF, we can test whether a given coefficient is significantly different from zero, at say a significance of 0.05

The weakness of this approach is that regression coefficients depend on each other – dependence may change as other features are added or removed. E.g. highly dependent features "share" their influence.

Stepwise Selection

We can select features in greedy fashion, at each step adding the next feature that most improves the error.

By maintaining a QR decomposition of X^TX and "pivoting" on the next best feature, stepwise selection is asymptotically as efficient as the standard solution method.

Regression and n-grams

Unlike naïve Bayes, regression does not suffer from bias when features are dependent.

On the other hand, regression has insufficient degrees of freedom to model dependencies. e.g.,

"page" and "turn(er)" vs. "page turner"

A unigram model distributes this influence between the two words, increasing the error when each occurs alone.

The strongest dependencies between words occur over short distances. Adding n-grams allows the strongest dependent combinations to receive an independent weight.

An example

Dataset: Reviews from amazon.com, about 1M book reviews. Includes text and numerical score from 1 to 5.

Each review (X) is represented as a bag-of-words (or n-grams) with counts.

The responses Y are the numerical scores.

Each X is normalized by dividing by its sum.

10-fold cross-validation is used for training/testing.

Final cross-validation RMSE = 0.84

Strongest negatives

'productive'
'covering'
'two stars'
'not recommend'
'calling'
'online'
'at best'
'track'
'travels'
'comprehend'
'ashamed'
'worthwhile'
'stern'
'parody'
'evaluate'
'web'
'exposes'
'covers'
'secret'
'foster'
'not buy'
'frederick'

Strongest positives

'not disappointed'
'be disappointed'
'five stars'
'no nonsense'
'5 stars'
'points out'
'no better'
'only problem'
'negative reviews'
'not only'
'the negative'
'a try'
'not agree'
'of print'
'adulthood'
'even better'
'literary'
'instead of'
'not put'
'be it'
'i wasnt'
'to avoid'

Outline

- Naïve Bayes Classifier
- Regularization in Linear Regression
- Generalized Linear Models
- Assignment Tips:
  - Matrix Representation

Logistic Regression

Goal: use linear classifier functions and derive smooth [0,1] probabilities from them,

Or: extend naïve Bayes to model dependence between features as in linear regression.

The following probability derives from X

$$p = \frac{1}{1 + \exp(-\sum_{j=1}^{n} X_j \beta_j)}$$

And satisfies

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) = \sum_{j=1}^{n} X_j \beta_j$$

Logistic Regression

Logistic regression generalizes naïve Bayes in the following sense:

For independent features X, the maximum-likelihood logistic coefficients β are the naïve Bayes classifier coefficients.

And the probability computed using the logistic function is the same as the Bayes posterior probability.

Logistic Regression

The log likelihood for a model β on N data vectors is:

$$l(\beta) = \sum_{i=1}^{N} y_i \beta\, x_i - \log(1 + \exp(\beta x_i))$$

Where we assume every *x* has a zeroth coefficient = 1.

The derivative *dl / d*β is

$$\frac{dl}{d\beta} = \sum_{i=1}^{N} y_i x_i - \frac{x_i}{1 + \exp(-\beta^T x_i)} = \sum_{i=1}^{N} x_i\,(y_i - p_i)$$

This is a non-linear optimization, but the function is convex. It can be solved with Newton's method if the model dimension is small. Otherwise, we can use stochastic gradient.

Regularizers for Logistic Regression

Just as with linear regression, Logistic regression can easily overfit data unless the model is regularized.

Once again we can do this either with L_1 or L_2 measures on the coefficients. The gradients are then:

L_2: $$\frac{dl}{d\beta} = \sum_{i=1} x_i \, (y_i \, - p_i) + \; 2\lambda\beta$$

L_1 : $$\frac{dl}{d\beta} = \sum_{i=1} x_i \, (y_i \, - p_i) + \; \lambda \, \mathrm{sign}(\beta)$$

As with linear regression, the L_1–norm tends to drive weak coefficients to zero. This helps both to reduce model variance, and also to limit model size (and improve evaluation speed).

Generative Linear Regression

Yet another formulation for Linear Regression (minimizing squared error) uses a probabilistic model:

$$Y = \beta X + \varepsilon$$

Where ε is a random error term that is normally distributed.

The likelihood is given by $\prod \exp\left(\frac{-{\varepsilon_i}^2}{2\sigma^2}\right)$ and taking logs gives

$$\frac{-1}{2\sigma^2}\sum_{i=1}^{n}{\varepsilon_i}^2 = \frac{-1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta x_i)^2$$

So choosing the β that **maximizes the likelihood of this generative model** also **minimizes the total squared error**.

Generative Logistic Regression

Logistic models also have a generative interpretation where the output of the model is the probability of a Bernoulli trial (i.e. a coin toss with probability p).

- If the output *Y* is *1*, the likelihood is *p*.
- If the output *Y* is *0*, the likelihood is *(1-p).*

substituting and maximizing the likelihood gives the same logistic model as before.

Refactoring Regression Models

We now have two kinds of "output" model

- Linear regression has a normally-distributed error process.
- Logistic regression models a Bernoulli process whose probability is a logistic function of the linear predictor.

For these and a number of other models, we use a basic linear predictor and an auxiliary function to specify the ***mean*** of an output distribution.

By selecting this function and the distribution we enumerate a family of regression models called ***Generalized Linear Models***.

Generalized Linear Models

GLMs have three components:

- A **probability distribution** from the exponential family (e.g. Normal for linear regression, Bernoulli for logistic).
- A **linear regression function** $\eta = \beta X$.
- A **link function** g such that $\mathrm{E}(Y) = \mu = g^{-1}(\eta)$

The mean $\mu = g^{-1}(\eta)$ is the key parameter of the distribution, and adequately specifies the distribution for likelihood maximization. The function g^{-1} is called the **mean function**.

Examples are here:

http://en.wikipedia.org/wiki/Generalized_linear_model

GLM Regularization

GLM packages (Matlab, R) support one or more forms of regularization

- Lasso (L_1)
- Ridge (L_2)
- Elastic-net which is a combination of Lasso and Ridge.

Together with the distribution and link function options, GLMs represent a rich family of global models.

Outline

- Naïve Bayes Classifier
- Regularization in Linear Regression
- Generalized Linear Models
- Assignment Tips:
  - Matrix Representation

Sparse Matrices

The term counts X can be stored as an NxM sparse matrix, where N is the number of terms, M the number of docs.

Sparse matrices contain only the non-zero values. Formats are:

Coordinate: store all tuples of (row, col, value), usually sorted in lexicographic (row, col) or (col, row) order.

CSC, Compressed Sparse Column: rows, values, compress(cols)

CSR, Compressed Sparse Row: cols, values, compress(rows)

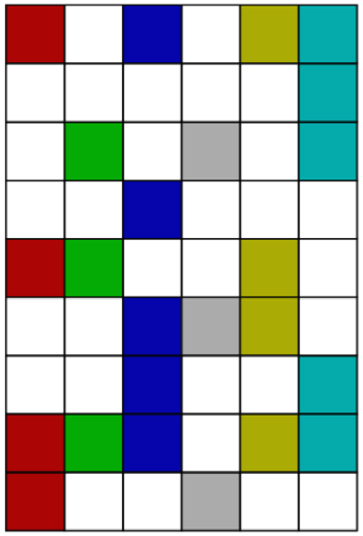
CSC especially is a widely-used format in scientific computing, for sparse BLAS and Matlab.

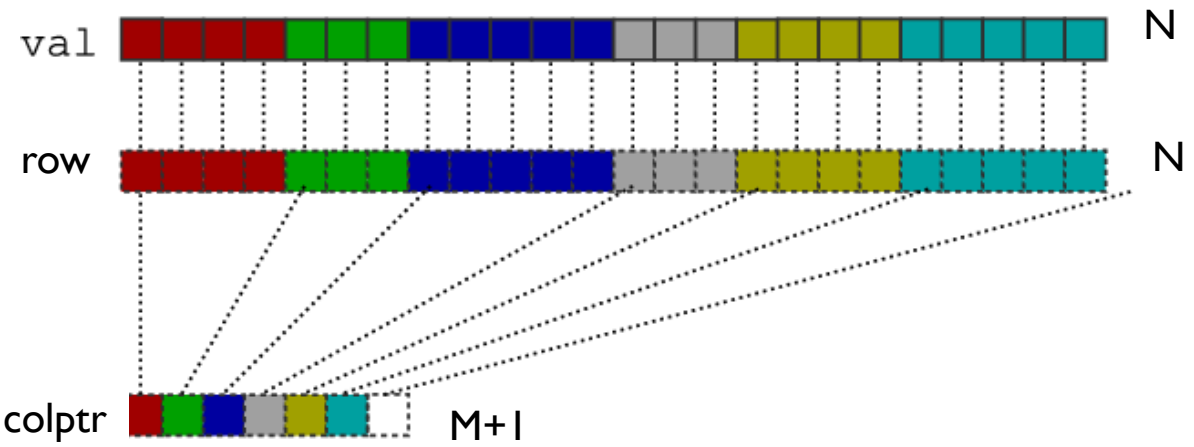
CSC supports efficient slicing by columns – this is useful for permutations, blocking and cross-validation by docids.

CSC format

Elements are sorted in (col, row) order, i.e. column-major order.

Value and row are the same as for coordinate format. Instead of storing every column value, CSC stores a matrix of M+1 column pointers (colptr) to the start and end+1 of the elements in each column.

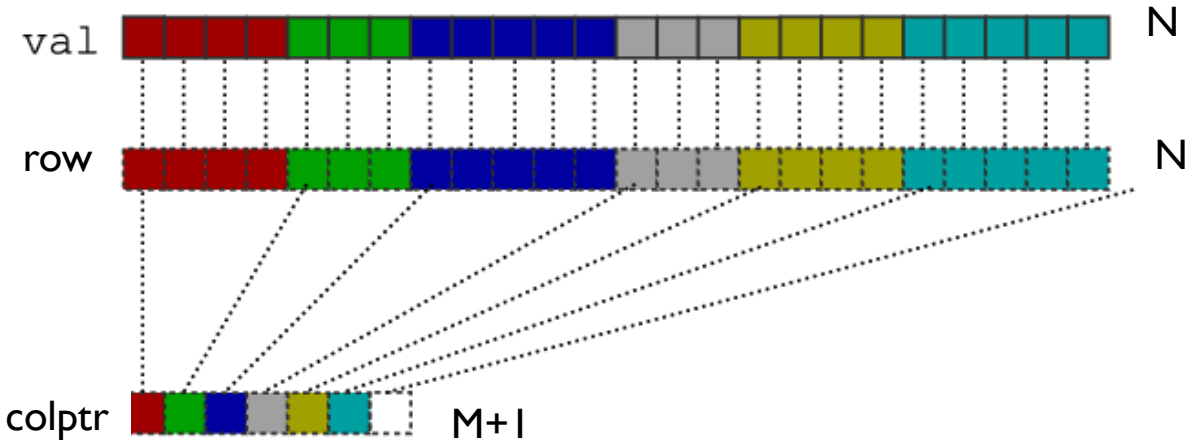




CSC format

Because of memory grain and caching (next time), its much faster to traverse the columns of a CSC sparse matrix than rows for matrix operations.

Additional techniques (autotuned blocking) may be used inside sparse matrix-vector multiply operators to improve performance – you don't need to implement this, but you should use it when available.



Other tips

- Use binary files + compression speeds up reads. Should be less than 10 secs for 0.5 GB.
- Use blocking for datasets that wont fit in main memory.
- Seek to start address of a block of data in the binary file – no need to explicitly split the file.
- Data is available in binary form already tokenized, with dictionary, if you need it.