Behavioral Data Mining

Lecture 2

Outline

Review of Statistical Learning

- Linear Regression, Nearest Neighbor Estimator, Loss
- Bias-Variance Tradeoff
- Naïve Bayes Classifier

Statistical Learning

We will follow the exposition in Hastie et al., "Statistical Learning"

Variables:

Qualitative (discrete): {*True, False*}, {*Red, Green, Blue*}, {*1, 2, 3, 4, 5*}, *or latent (anonymous) factors like* $\{F_1, F_2, F_3, F_4, F_5\}$

Quantitative (continuous): Real values $\in \mathbb{R}$, or values in an interval. e.g. height, weight, frequency,…

Statistical Learning

Variables will often be vectors (uppercase italic) $X = (X_1, \ldots, X_n)$

or m x n matrices (uppercase bold) $\mathbf{X} = \begin{pmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \cdots & X_{mn} \end{pmatrix}$

Observations will be written as matching lowercase, e.g. a series of observations of X would be $(x_1, \ldots, x_p)$

Estimates will be written as symbols with a hat: $\hat{Y}$ for a variable Y or $\hat{\beta}$ for a parameter β.

Linear Regression

The predicted value of Y is given by:

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

and the vector of coefficients $\hat{\beta}$ comprise the regression model.

In the general case, we can have a vector output and a matrix of coefficients and write $\hat{Y} = \hat{\beta}\mathbf{X}$ (assume $X_0 = 1$).

(thinking of observations as columns rather than rows will work better for matrix algorithms, i.e. $\mathbf{X}$ is transposed rel. to book).

Statistical Learning

i.e. **columns of** $\mathbf{X} = \begin{pmatrix} X_{11} & \cdots & X_{1n} \\ \vdots & \ddots & \vdots \\ X_{m1} & \cdots & X_{mn} \end{pmatrix}$

are distinct observations, rows of X are input features.

Same for output features.

Residual Sum-of-Squares

To determine the model parameters $\hat{\beta}$ from some data, we need to define an error function, like Residual Sum of Squares:

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta x_i)^2$$

or symbolically $\mathrm{RSS}(\beta) = (\mathbf{y} - \beta\mathbf{X})^T(\mathbf{y} - \beta\mathbf{X})$. To minimize it, take the derivative wrt β which gives:

$$(\mathbf{y} - \beta\mathbf{X})\mathbf{X}^T = 0$$

And if $\mathbf{X}\mathbf{X}^T$ is non-singular, the unique solution is:

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

Computing Regression Solutions

We get an exact solution if we can compute $(\mathbf{X}\mathbf{X}^T)^{-1}$. This matrix is m*m where m = number of input variables (features).

If m is not too large (say 10^4 or less), we can store and accumulate $\mathbf{X}\mathbf{X}^T$ in memory. We don't need all of $\mathbf{X}$ which can remain on disk, only blocks of it.

Let $\mathbf{X}_i$ denote the ith *block* of input samples, i.e. a block of $\mathbf{X}$ whose column indices range from $(i\text{-}1)*b+1,\ldots,i*b$. Then if there are n/b blocks

$$\mathbf{X}\mathbf{X}^T = \sum_{i=1}^{n/b} \mathbf{X}_i \mathbf{X}_i^T$$

And we can compute a similar sum for $\mathbf{y}\,\mathbf{X}^T$

Computing Regression Solutions

If there are n samples in total, the complexity of this approach is $O(m^2 n)$. This may be too large, but remember *constants matter*.

For dense data, matrix-matrix multiply is extremely fast (1 TF per GPU, or several hundred GF on a CPU), and this method requires only a single pass over the dataset.

An Iterative Regression Solution

We are trying to solve a linear system $\hat{\beta} = \mathbf{y}\,\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$

Rewrite it as $\hat{\beta}\mathbf{A} = \mathbf{r}$ where $\mathbf{A} = \mathbf{X}\mathbf{X}^T$ and $\mathbf{r} = \mathbf{y}\,\mathbf{X}^T$

Then we can use any of a variety of iterative solvers (e.g. conjugate gradient) to solve for $\hat{\beta}$. These solvers only require evaluations of $\tilde{\beta}\ \mathbf{A}$ for various intermediate vectors $\tilde{\beta}$.

Each evaluation of $\tilde{\beta}\ \mathbf{A} = \tilde{\beta}\ \mathbf{X}\mathbf{X}^T$ can be done in blocks in one pass over the dataset and requires **O(m) space and O(m n) time**.

The total complexity of an iterative solution is O(m n k) for k iterations, which is faster when k << m.

Going Faster

The disadvantage of this approach is that we have to make k passes over the dataset since $\mathbf{X}\mathbf{X}^T$ wont fit in memory.

There is a class of **Communication-Minimizing Algorithms** that can solve iterative systems in (typically) fewer iterations. These include block conjugate-gradient, block GMRES, etc.

They compute products $\tilde{\beta}\,\mathbf{X}\mathbf{X}^T$ where $\tilde{\beta}$ is a (row) *multivector* , i.e. a matrix of vectors, instead of a simple vector.

Intuitively, this method explores the solution space faster.

In practice $\tilde{\beta}\,\mathbf{X}\mathbf{X}^T$ for a multivector $\tilde{\beta}$ may be only slightly slower than for a simple vector.

Nearest Neighbors

The estimator for *x* using k nearest-neighbors is

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

Where y_i is the response for input x_i.

- Apparently requires no assumptions about the form of the function Y(X).
- The "model" however is potentially huge, and inference is very expensive O(n m) per sample.

Nearest Neighbors

In practice there are many hybrids that mix regression and neighbor methods, e.g. the Netflix prize winner.

Kernel methods express regression as a weighted combination of outputs, using only inner products of inputs. In general, the combination weights can be anywhere in the interval [0,1].

Local regression fits linear models (typically with reduced complexity compared to a global model) locally.

Loss Functions

To generalize the simple regression model above, we assume that outputs *Y* can be computed using a function *f(X)* of inputs *X*.

We further assume that *X* and *Y* have a joint probability distribution *Pr(X,Y)*.

Then we define a **Loss Function L(Y, f(X))** to model the error we want to minimize. The **squared error loss** is

$$L\big(Y, f(X)\big) = \big(Y - f(X)\big)^2$$

Regression Function

To integrate out the effects of the probability distribution, we define the Expected Prediction Error as:

$$\mathrm{EPE}(f) = \mathrm{E}\big((Y - f(X))^2\big)$$

After conditioning on *X*, we find that the solution for *f* is:

$$f(x) = E(Y|X = x)$$

Which is known as the **regression function**.

The K-nearest neighbor predictor approximates the regression function with a local average.

Probabilistic Least Squares

If we assume that

$$f(x) = \beta x$$

After substituting into the regression function formula and solving, we find that:

$$\beta = \mathrm{E}(YX^T)[\mathrm{E}(XX^T)]^{-1}$$

And if we approximate the expected values by averages from the available data, we get the formula for $\hat{\beta}$ as before.

Minimum Absolute Error

If instead of minimizing squared error, we may wish to minimize the sum of absolute errors

$$L\big(x, f(x)\big) = |y - f(x)|$$

Then the regression function in this case is

$$f(x) = \text{median}(Y \mid X = x)$$

The conditional median of *Y* given *X* = *x*.

This estimator is more robust (less sensitive) to outliers, but not as smooth.

Discrete Loss

Finally if *f(x)* is a classifier (zero or one output) we can define a zero-one loss function, and solve the regression function. The result is the Naïve Bayes classifier.

Outline

Review of Statistical Learning

- Linear Regression, Nearest Neighbor Estimator, Loss
- Bias-Variance Tradeoff
- Naïve Bayes Classifier

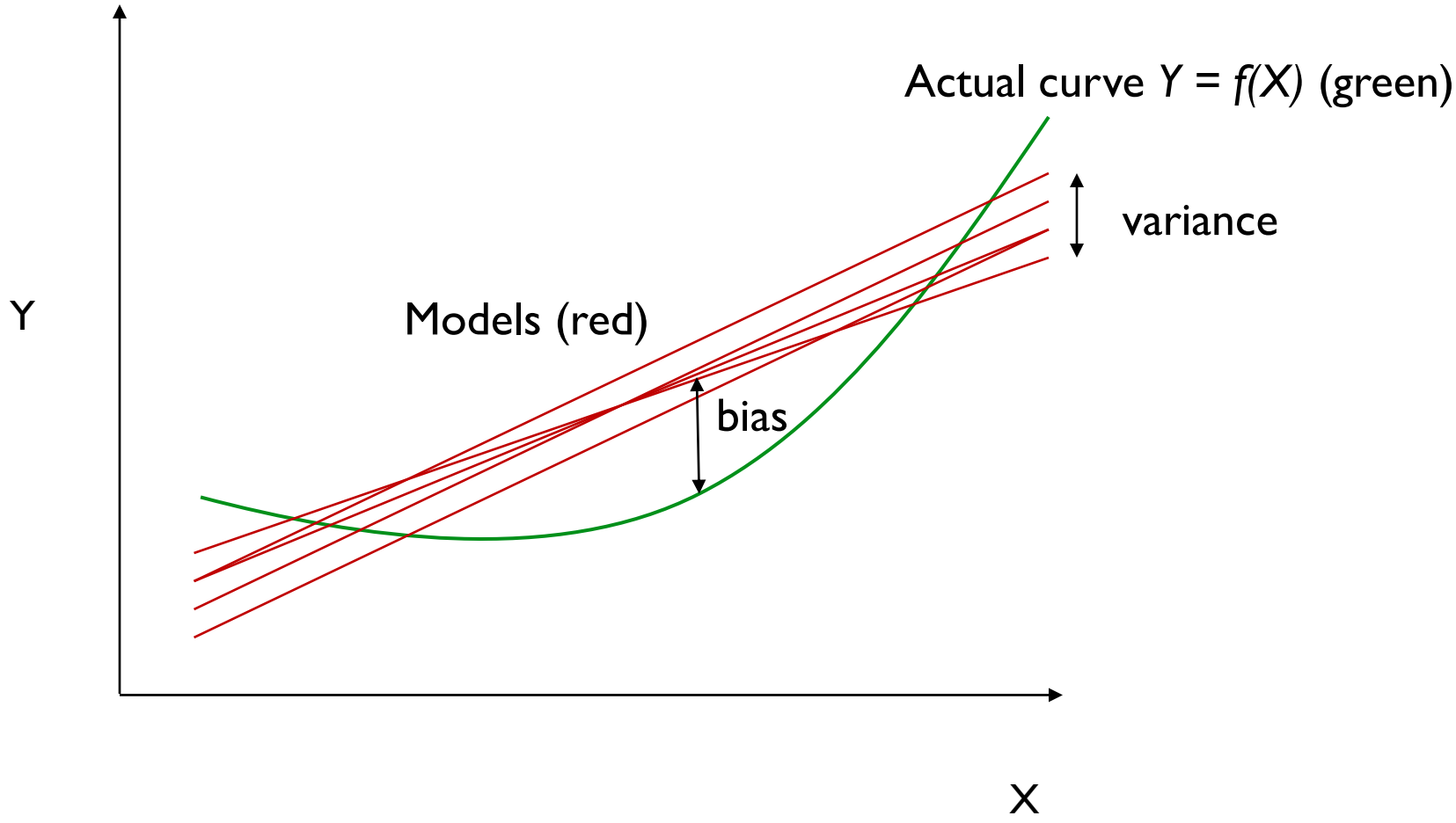
Bias and Variance

Informally, the error of an estimator at a point can be decomposed into a **variance** component and a **bias** component.

Variance: the variability of the estimate due to differences in the training data – which is a sample from some ideal distribution.

Bias: the systematic deviation of the estimate from the actual output due to lack of fit by the model.

Bias and Variance



Bias/Variance Tradeoff

There is a tradeoff between bias and variance.

Simple models have few parameters → low variance in estimates
But simple models have large systematic errors → high bias

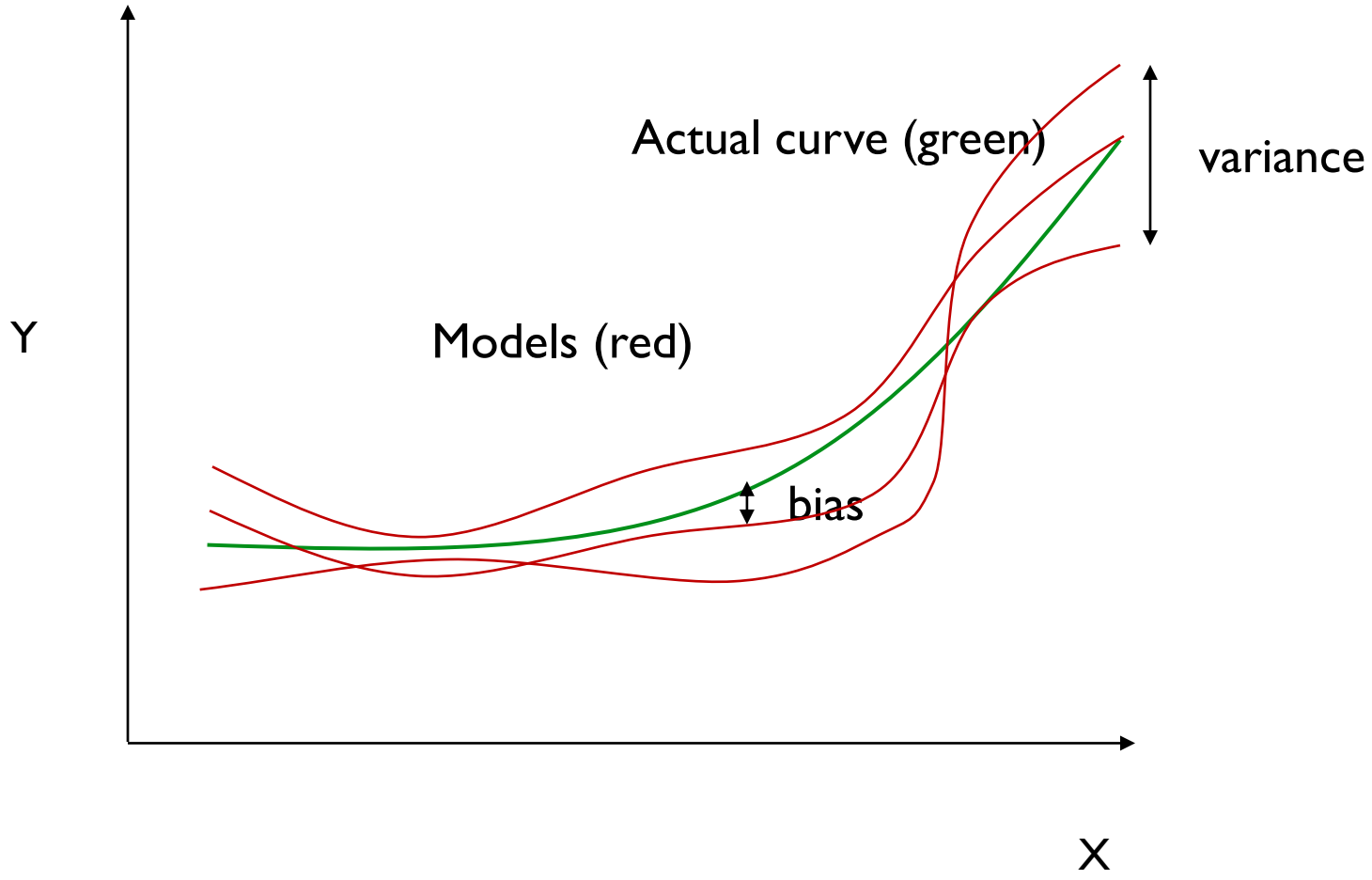
Complex models have many parameters → high variance
But they can generate better fit to data → low bias

Bias/Variance Tradeoff

Linear regression is about the simplest possible model, and has high bias but low variance.

K-nearest neighbors has very low bias (especially for small k), but potentially huge variance.

Low Bias, High Variance



Bias/Variance in Distributed Inference

Variance decreases with sample set size.

When datasets are partitioned across machines, an ideal model for each sample will have higher variance than a model for the entire dataset.

We can try reducing the variance by averaging the models, but what happens to the bias? (Zhang et al. paper).

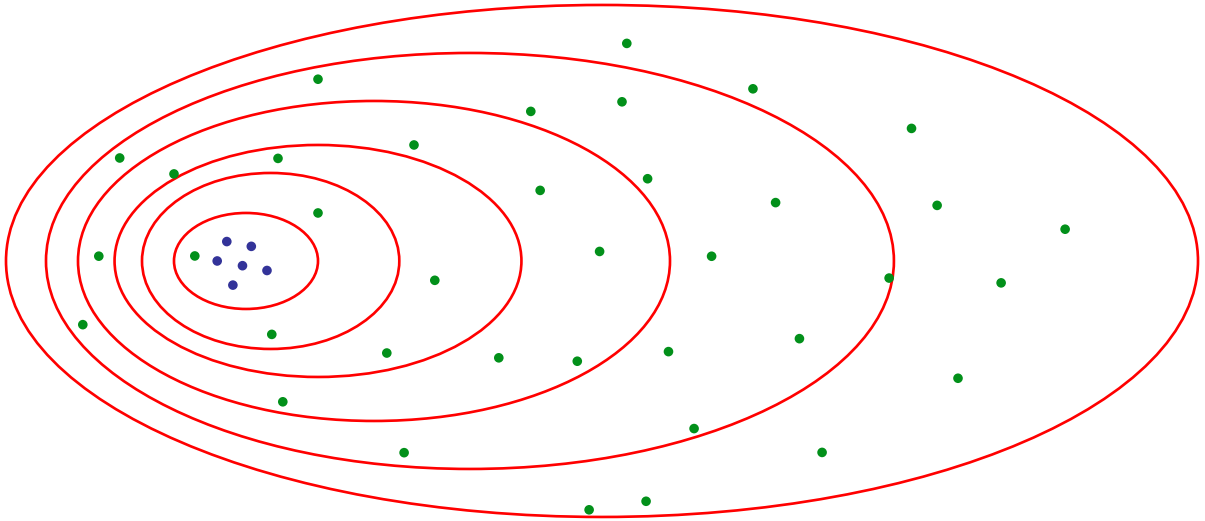
Bias/Variance in Distributed Inference

Intuition: models are optima in non-linear function spaces:

The large-sample models (blue) have low variance and low bias.

The small-sample models (green) have higher variance, and higher bias because of it.

Simple averaging doesn't reduce this bias (but weighting does).



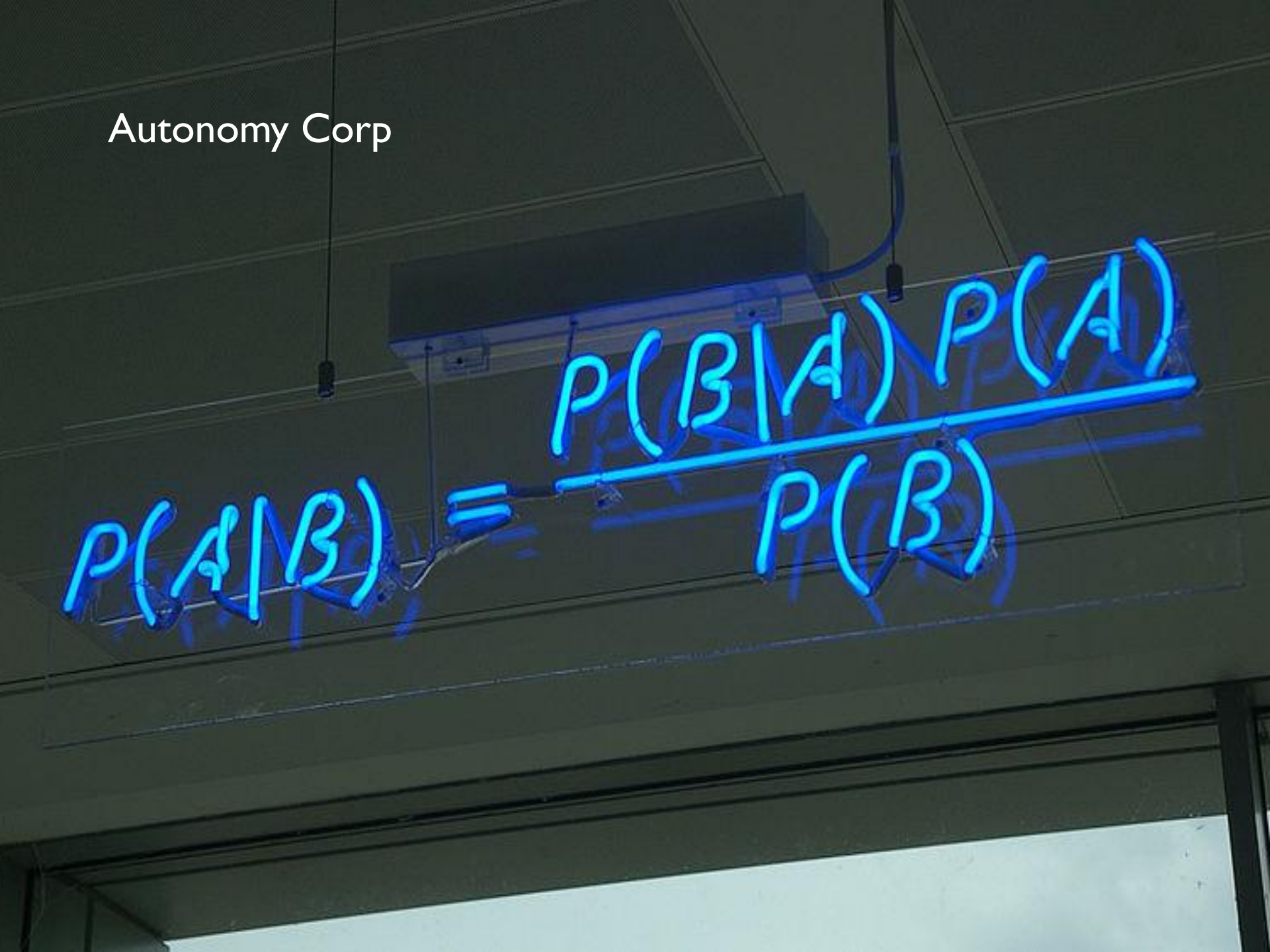
Contour plot of model loss (lowest in the center)

Outline

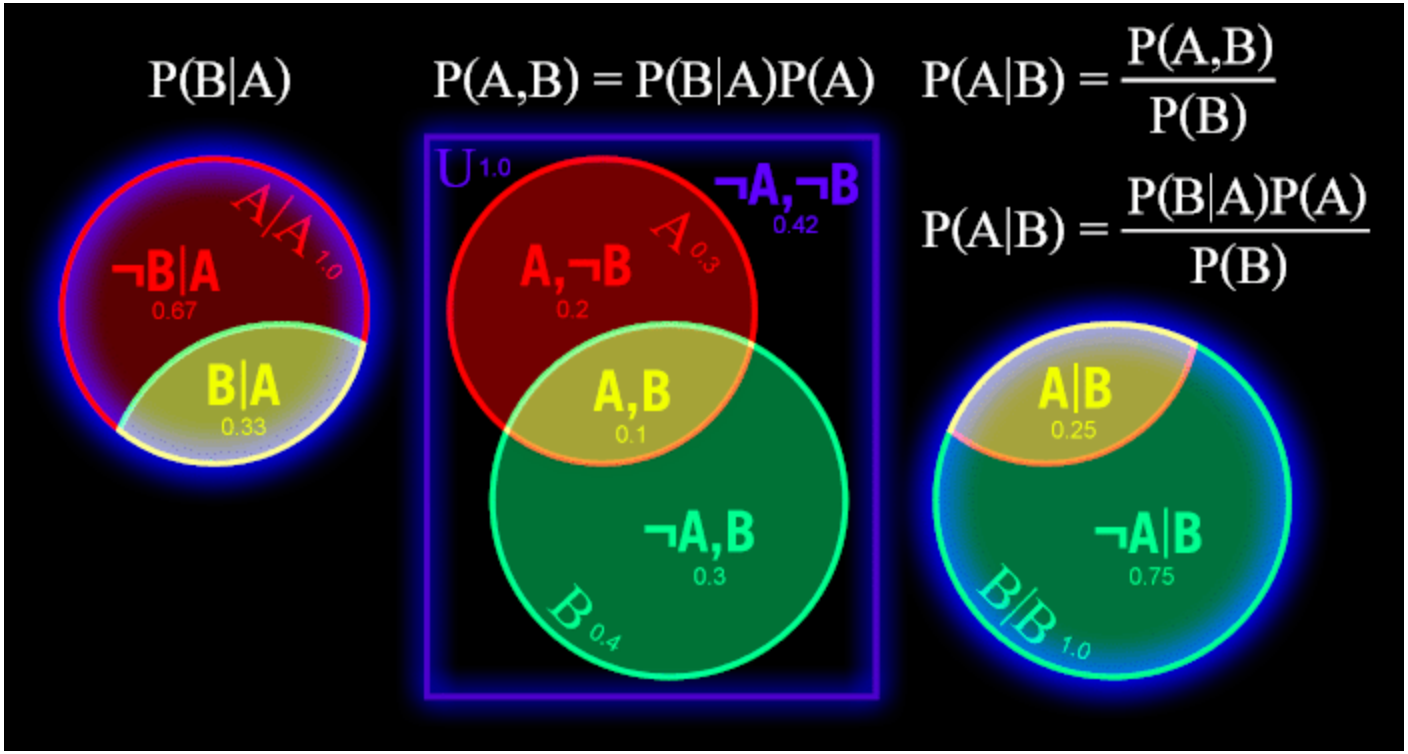
Review of Statistical Learning

- Linear Regression, Nearest Neighbor Estimator, Loss
- Bias-Variance Tradeoff
- Naïve Bayes Classifier

Autonomy Corp



Bayes' Theorem



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

Bayesian vs Frequentist Perspectives

- Bayesian inference is used most often used to represent probabilities which are beliefs about the world. It represents the Bayesian perspective.

- The Bayesian perspective was not widely accepted until the mid 20th century, since most statisticians held a frequentist perspective. The frequentist perspective interprets probability narrowly as relative frequencies of events.

Bayesian User Conference 2012: "Tentative program follows. Based on attendance, each day's final program will be announced the following day"

Alternatives

We know from the axioms of probability that $P(e) + P(\neg e) = 1$

Or if we have multiple, mutually exclusive and exhaustive hypotheses (e.g. classifying) $e_1 \ldots e_n$, then $P(e_1) + \ldots + P(e_n) = 1$

Its also true that $P(e|D) + P(\neg e|D) = 1$

And $P(e_1|D) + \ldots + P(e_n|D) = 1$

The normalizing probability P(D) in Bayes' theorem may not be known directly in which case we can compute it as

$$P(D) = P(D|e_1)\,P(e_1) + \ldots + P(D|e_n)\,P(e_n)$$

Or not. We don't need it to rank hypotheses.

Ranking

The un-normalized form of Bayes is

$$P(e|D) \propto P(D|e)P(e)$$

Which is good enough to find the best hypothesis.

Chains of evidence

Bayes' theorem has a recursive form for updates given new information.

$$P(e|D_2, D_1) = \frac{P(D_2|e, D_1)\, P(e|D_1)}{P(D_2|D_1)}$$

Where the old posterior $P(e|D_1)$ becomes the new prior.

Using Logs

For any positive p and q, $p > q \Leftrightarrow \log(p) > \log(q)$

It follows that we can work with logs of probabilities and still find the same best or most likely hypothesis.

Using logs:

- Avoids numerical problems (overflow and underflow) that happen with long chains of multiplications.
- Often leads to linear operations between model and data values.

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

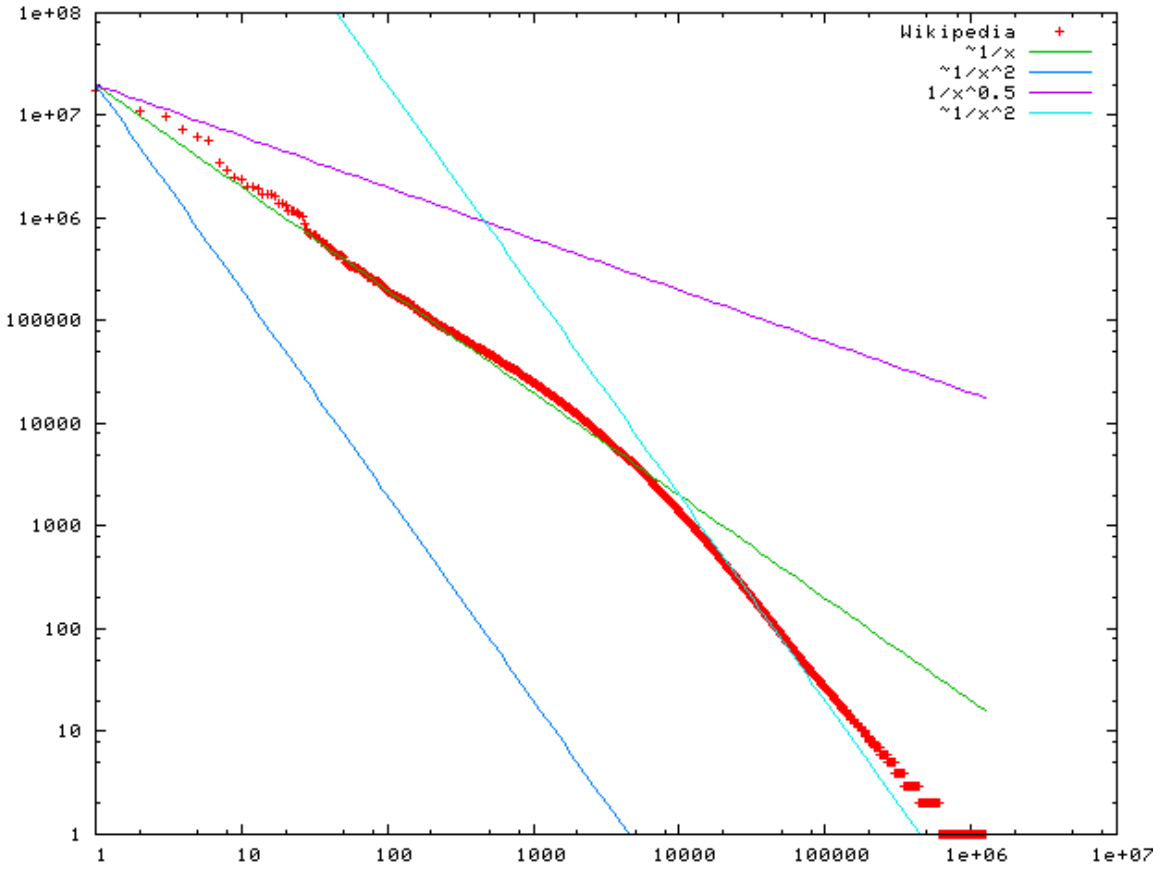
First Assignment

- 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

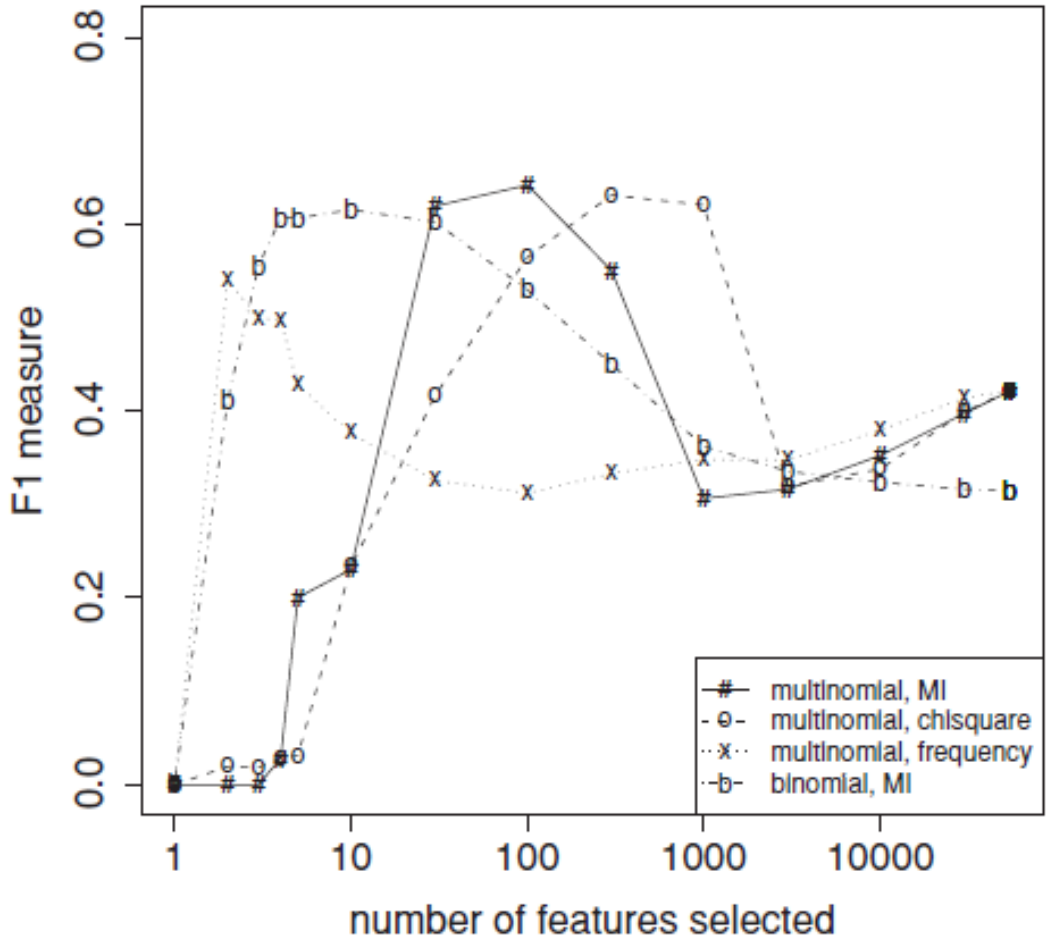
CHI-Squared

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.

Summary

Review of Statistical Learning

- Linear Regression, Nearest Neighbor Estimator, Loss
- Bias-Variance Tradeoff
- Naïve Bayes Classifier