COMPUTER SCIENCE
INDIANA UNIVERSITY
School of Informatics and Computing
Bloomington

Naive Bayes

Reminders/Comments

- Generally good job on Assignment 1
  - if you did not do as well as you hoped, do not worry too much and focus on doing well in Assignment 2
- For assignment 2, must submit a typed up document (with your name on that document)
- Thought questions were better this time

Thought question

- (Overfitting) The note in chapter 4.4.1 says: "overfitting is indicated by a significant difference in fit between the data set on which the model was trained and the outside data set on which the model is expected to be applied". For the given example, overfitting is caused by "complexity of the model was increased considerably, whereas the size of the data set remained small". According to my understanding, any model trained perform well in training set but perform not well in testing set can be considered as overfitting. Usually, the training set and the testing set both belong to the same distribution in theory. However, assume the condition we have a very big training set and a very small testing set, maybe the model can explain the training set very well, but very bad for the testing set. Is this still called overfitting?
  - The underlying assumption is the train and test are distributed the same
  - If training on a huge training set, close to true underlying parameters and so will perform reasonably on the test set
  - This will not be true if (a) the chosen distribution is not appropriate, so perform poorly on both training and test set or (b) if the training and test are not identically distributed (see the field of sample bias correction)

Thought question

- What allows gradient descent be applied to non-convex functions? Isn't this contradictory to the nature of gradient descent since we're trying to find a global minima? In class we've always assumed that the function we're applying gradient descent to is convex. However, in this presentation, (www.cs.nyu.edu/~yann/talks/lecun-20071207-nonconvex.pdf) the author argues that convexity is overrated and sometimes we want to deal with a non-convex loss functions as it improves accuracy and speed in certain situations. He then goes on to imply that we can use stochastic gradient descent to help solve these problems, which contradicts our assumption that our function must be convex.
  - Gradient descent can be applied to any function, convex or not convex
  - Some of the stochastic optimization theory only applies to convex functions (especially for stochastic gradient descent)

Convex versus non-convex

- Both types of objectives can be useful
  - I am not advocating restricting only to convex objectives
- In some cases, want a model that corresponds to a non-convex objective (example: multi-layer neural network)
- In other cases, however, simply being more careful in specification gives an equivalent convex optimization
  - So why not use an equally “powerful”, but easier to optimize loss?
  - In some cases, we can also work really hard and represent a hard non-convex optimization as a similar convex optimization
- Important to recognize when its worthwhile moving to a non-convex objective and what the trade-offs are

Global versus local

- I have somewhat interchangeably used “convex versus non-convex” and “global versus local”

- What we really care about is the ability to get global solutions, rather than local solutions

- Convex functions make this easy, so optimizing convex functions is one way to ensure we get global solutions

- In other cases, however, we can guarantee global optimality even with non-convex functions —> in this case, its okay that its non-convex, because we still get the optimal solution

Example: non-convexity using Euclidean loss with sigmoid transfer

$$\mathrm{Err}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - p_i)^2 \qquad \triangleright\, p_i = \sigma(\mathbf{x}_i^\top \mathbf{w}), \quad e_i = y_i - p_i$$

$$\nabla \mathrm{Err}(\mathbf{w}) = -2\mathbf{X}^\top \mathbf{P}\,(\mathbf{I} - \mathbf{P})\,(\mathbf{y} - \mathbf{p})\,.$$

$$H_{\mathrm{Err}(\mathbf{w})} \quad = \quad 2\mathbf{J}^\top \mathbf{J} + 2\mathbf{J}^\top \mathbf{E}(2\mathbf{P} - \mathbf{I})\mathbf{X},$$

- This Hessian **not** guaranteed to be positive definite
- Since we have a convex loss (cross-entropy), derived principally from maximum likelihood, should use that instead

Thought question

- While in linear regression target value Y = R, and in the classification problems Y is like {0, 1} or {-1, 1}, can we say that classification is a special case of regression? So that we can use linear regression to solve classification problems, since the matrix computing is much easier.
  - One can specify any distribution over Y, compute E[Y lx] and then use that prediction in some way to predict y
  - The question is more about performance
  - In many cases, training a linear versus a logistic regression model will give similar performance
  - In other cases, however, the logistic better focuses prediction into separating the two classes {0,1}, whereas linear is trying to minimize squared distances to a line and so is more sensitive to outliers
  - **Matlab Demo!**

Discriminative versus generative

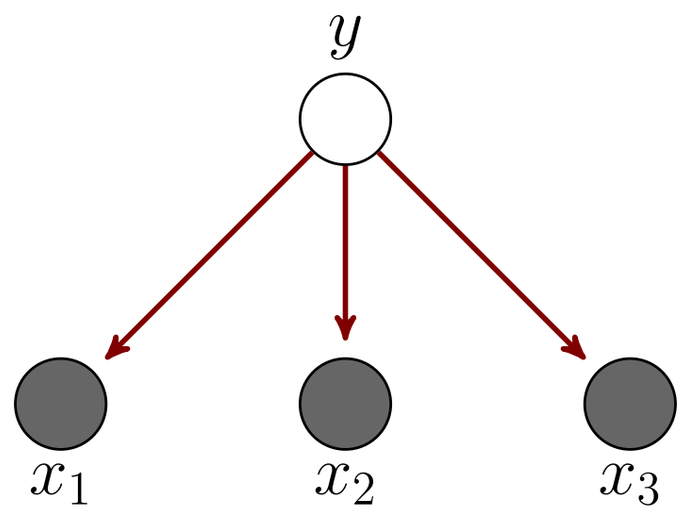
- So far, have learned p(y l x) directly
- Could learn other probability models, and using probabilty rules to obtain p(y l x)
- In generative learning,
  - learn p(x l y) p(y) (which gives the joint p(x, y) = p(xly) p(y))
  - compute p(x l y) p(y), which is proportional to p(y l x)

Pros and Cons

- Discriminative
  - focus learning on the value we care about: p(y l x)
  - can be much simpler, particularly if features x complex, making p(x l y) difficult to model without strong assumptions
- Generative
  - can be easier for practitioner/expert to encode prior beliefs
  - e.g., for classifying faces in male or female, the structure/distribution over the features (heavier eyebrows, squarer jaw, etc.) can be more clearly specified by p(x l y), whereas p(y l x) does not allow this information to be encoded

Naive Bayes

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y).$$



- How do we learn p(x | y) realistically?

- One option is to make a (strong) conditional independence assumption: the features are independent given the label
  - Surprisingly, despite the fact that this seems unrealistic, in practice this work well —> one hypothesis is that we are mostly running these algorithms on “easy” data, another is that this assumptions matches settings were dependencies cancel in some way

- Example: given a patient does not have the disease (y=0), attributes about patient are uncorrelated (e.g., age & smokes)
  - even within a class, age and smokes could be correlated

How to use Naive Bayes?

- Our decision rule for using these probabilities is the same as with (multinomial) logistic regression: pick the class with the highest probability

- Assume you've learned p(x l y) and p(y) from a dataset

- Compute

$$f(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} p(y|\mathbf{x})$$
$$= \arg\max_{y \in \mathcal{Y}} p(\mathbf{x}|y)p(y)/p(\mathbf{x})$$
$$= \arg\max_{y \in \mathcal{Y}} p(\mathbf{x}|y)p(y)$$

Types of features

- Before, we had to choose the distribution over y given x
  - e.g. p(y | x) is Gaussian for continuous y
  - e.g. p(y | x) is Poisson for y in {1, 2, 3, …}
  - e.g. p(y | x) is Bernoulli for y in {0,1}
- Typically parameters to p(y | x) had E[y | x] = xw
- Now we need to choose the distribution over x given y; how do we pick p(x | y)? What are the parameters?

Exercise: binary features

- For binary features x in {0,1}, binary y in {0,1}, what is p(x l y)?

- How do we learn p(x l y)?

- How do we learn p(y)?

Exercise: continuous features

- For continuous features x, binary y in {0,1}, what could we choose for p(x I y)?

- What are the parameters and how do we learn them?

Whiteboard

- Naive Bayes

- Multiclass with multinomial logistic regression