COMP5046: Classification with Features

Joel Nothman

joel.nothman@sydney.edu.au

School of Information Technologies
University of Sydney

2018-04-10

Part I

Naïve Bayes

What are we trying to do?

- some examples:
  - try to predict if it will rain tomorrow
  - try to predict what someone will say next
  - classify texts into categories, e.g. law, health, etc.

- we have a lot of information to exploit in language
- **build a model from training data that can predict future data**

Outline

- Bayes' Rule for classification
- Naïve Bayes
- **Feature representation**
- Perceptron models
- Maximum Entropy models

What are we trying to calculate?

- x is the information we have (the observed context):
  - who is talking and what they've already said
  - the entity and their Wikipedia article
- y is what we're trying to predict:
  - what word they'll say next
  - the category the entity fits into
- now calculate $p(y|x)\ \forall\ y$ (all classes)
- and select the y that maximises $p(y|x)$, formally:

$$\underset{y}{\operatorname{argmax}}\, p(y|x) \tag{1}$$

Bayes' Rule derives from conditional probability

- conditional probability leads to the chain rule:

$$p(y|x) \equiv \frac{p(y \cap x)}{p(x)} \tag{2}$$

$$p(y \cap x) = p(x)p(y|x) = p(y)p(x|y) \tag{3}$$

- rewriting Equation 3 we get **Bayes' rule** (or theorem):

$$p(y|x) \equiv \frac{p(y)p(x|y)}{p(x)} \tag{4}$$

Bayes' Rule splits $p(y|x)$ into pieces

- combine the last two slides together
- we can calculate the most likely y given x:

p(x) is constant for every instant

$$\operatorname*{argmax}_y p(y|x) = \operatorname*{argmax}_y \frac{p(y)p(x|y)}{p(x)} \tag{5}$$

$$= \operatorname*{argmax}_y p(y)p(x|y) \tag{6}$$

- we can drop out the denominator for fixed x when we only want y
- **often useful when $p(x|y)$ and $p(y)$ easier to calculate**
- **or they encode the structure of the problem**
- NB: $p(x|y)$ can cause confusion, since it seems back to front

Calculating the prior probability, $p(y)$, is usually easy

- how many times the class y occurs divided by the total number of training instances

$$p(y) = \frac{\text{count}(y)}{\sum_{y_i \epsilon Y} \text{count}(y_i)} \tag{7}$$

- **relative frequency calculated over our training data**
- smoothing is still important for rare classes

Calculating $p(x|y)$ is often difficult

- conditioning on y is easy in classification
- but modelling the probability of x ...

- $p(x|y)$ can also be calculated using relative frequency if you assume x can be treated as an atom (a unit)
- although smoothing is still critical

- **difficult to calculate if x is a complex, structured object** e.g. a Wikipedia article
- probabilities for $p(x|\text{anything})$ become vanishingly small
- could use a language model for text, but we may want to encode diverse information

Features identify aspects of a context

- features describe a particular state of the decision context
- **and** the hypothesised/actual class for classification tasks

(`actress` in Wikipedia text, `Film/TV`)	Jessica Alba
(`murderer` in Wikipedia text, `Crime`)	Mark "Chopper" Reid
(`song` in Wikipedia text, `Music`)	Lady Gaga

- features don't need to be absolute predictors of a class
- **i.e. features can be ambiguous**
- they can also be *negative evidence* for a class

A feature is...

- often a boolean *indicator* function (i.e. returning 0 or 1)

$$f_i(x, y) = \text{attribute}(x) \cap (\text{class} = y) \tag{8}$$

- where attribute(x) checks evidence from the context
  - might identify a boolean property (e.g. `islower(word)`)
  - might pick out elements of the context (e.g. `prev_word == Mr`)
- `islower` and `prev_word` are *contextual predicates*
- `Mr` is the *value* of the `prev_word` contextual predicate

- more generally, f_i is a (bounded) real-valued function
- and it can pick out aspects of the class y, as well as the context x

A feature is often defined in terms of x only

- making feature vector a function of x **and** y allows y to be structured
  - there may be too many classes to learn separate parameters for each
  - similar y can share features and hence learnt parameters

- for simple classification, we ignore structure of y
  - use same attributes of x for features with each class y

	PER				LOC				ORG			
$\mathbf{f}(John, PER)$	1	0	1	1	0	0	0	0	0	0	0	0
$\mathbf{f}(John, LOC)$	0	0	0	0	1	0	1	1	0	0	0	0
$\mathbf{f}(John, ORG)$	0	0	0	0	0	0	0	0	1	0	1	1

- note y here does not need to be the *gold standard* class for x
  - we can calculate feature values for each y being considered for some x

Features in Naïve Bayes

- split the context (training instance) into individual features:
- replace x with $x = \{x_1, x_2, \ldots x_n\}$

$$p(x|y) = p(x_1, \ldots, x_n|y) \tag{9}$$
$$= p(x_1|y)p(x_2|x_1, y) \ldots p(x_n|x_{n-1} \ldots x_1, y) \tag{10}$$

- this is no easier to calculate than before!
- **assume conditional independence between features to simplify:**

$$p(x|y) = p(x_1|y)p(x_2|y) \ldots p(x_n|y) \tag{11}$$
$$= \prod_i p(x_i|y) \tag{12}$$

Reminder: Conditional Independence

- two events A and B are **conditionally independent** iff:

$$p(A \cap B|C) = p(A|C)p(B|C) \tag{13}$$

- i.e. $p(A|B \cap C) = p(A|C)$ and $p(B|A \cap C) = p(B|C)$
- knowing C gives no knowledge about $A \cap B$ beyond what C indicates individually about A or B
- i.e. knowing C may give some knowledge about A or B

Complex Features

- features can be arbitrarily complex
  - e.g. document level features in named entity recognition (document = `cricket` & current word = `Lancashire`, `ORG`)
    $\Longrightarrow$ hopefully tag `Lancashire` as `ORG` not `LOC`
- features can be combinations of atomic features
  - (current word = `David` & next word = `Jones`, `ORG`)
    $\Longrightarrow$ hopefully tag `David` as `ORG` not `PER`

- **feature engineering** during model development to define useful features
  - designed to target frequent errors

Conditional independence is a lie!

- conditional independence is an assumption of Naïve Bayes
- **unfortunately the assumption is usually totally wrong**
- for example:
  `unigram=actor` in text, `Film/TV`
  is not independent of
  `bigram=American actor` in text, `Film/TV`
- **often we get away with it anyway!**

Estimation of Naïve Bayes parameters

- each $p(x_i|y)$ is calculated using relative frequencies (*see lab!*):

# of feature i in class/(sum of any feature in class)

$$p_{iy} = \frac{\text{count(feature } i \text{ in class y)}}{\sum_{i'} \text{count(feature } i' \text{ in class y)}} \tag{14}$$

- *parameters*: values calculated in training, used to predict
- **this relative frequency is the Maximum Likelihood Estimate**
- MLE is one approach to *faithful representation*
- **what about previously unseen features?**
- smoothing: redistribute mass to unseen features for more *sensible generalisation*

Part II

Perceptron models

Generative and Discriminative Models

- Using Bayes' Rule requires a model of an observation x
  - a **generative model** calculates $P(x, y)$
  - Can be hard to design a principled model
    i.e. without naïve independence assumptions

- To predict the best y, we can learn $P(y|x)$ directly
  - or even just $\text{score}(y|x)$ (not a valid probability)
  - a **discriminative model**
  - State of the art for many tasks
  - The system can efficiently learn how to trust features

Looking at features differently

- instead of thinking of features as events in a joint probability:

$$p(x|y) = p(x_1, \ldots, x_n|y) \tag{15}$$

- think of them as being weighted by how likely they are
- e.g. for named entity recognition:
    - (word $=$ `John`, PER) $\Longrightarrow$ high weight
    - (word $=$ `John`, LOC) $\Longrightarrow$ negative weight
    - (word has an even number of letters, PER) $\Longrightarrow$ near-zero weight

Feature vectors

classifying a single word

- for all **active** features, add up all the associated weights

	PER				LOC				ORG			
$\mathbf{f}(John, PER)$	1	0	1	0	0	0	0	0	0	0	0	0
$\mathbf{f}(John, LOC)$	0	0	0	0	1	0	1	0	0	0	0	0
$\mathbf{w}$	3	1	1	0	-2	2	-1	0	-1	1	1	0

$$\text{score}(x, y, \mathbf{w}) = \mathbf{w} \cdot \mathbf{f}(x, y) \quad (16)$$

$$= \sum_i w_i f_i(x, y) \quad (17)$$

- dot product between the weight vector and feature vector

$$\text{score}(John, PER, \mathbf{w}) = (1 \times 3) + (1 \times 1) = 4 \quad (18)$$

$$\text{score}(John, LOC, \mathbf{w}) = (1 \times -2) + (1 \times -1) = -3 \quad (19)$$

Weights can be hard to interpret

- can incorporate many complex overlapping features
  - no independence assumptions
- allows linguistically motivated features
- but can be hard to interpret:
  - $x =$ the cats sat on the mat
  - $f_{(\text{has word cats})}(x) = f_{(\text{has stem cat})}(x) = 1$
  - $w_{(\text{has word cat})} = 1$
  - $w_{(\text{has word cats})} = 1$
  - $w_{(\text{has stem cat})} = 1$

  Imagine we want to know which features are most important...

Weights can be hard to interpret

- can incorporate many complex overlapping features
  - no independence assumptions
- allows linguistically motivated features
- but can be hard to interpret:
  - $x = \text{the cats sat on the mat}$
  - $f_{(\text{has word cats})}(x) = f_{(\text{has stem cat})}(x) = 1$
  - $w_{(\text{has word cat})} = 1$
  - $w_{(\text{has word cats})} = 1$
  - $w_{(\text{has stem cat})} = 1$

    Imagine we want to know which features are most important…

    Weight is effectively double how it appears

Weights can be hard to interpret

- can incorporate many complex overlapping features
  - no independence assumptions
- allows linguistically motivated features
- but can be hard to interpret:
  - $x =$ the cats sat on the mat
  - $f_{(\text{has word cats})}(x) = f_{(\text{has stem cat})}(x) = 1$
  - $w_{(\text{has word cat})} = 1$
  - $w_{(\text{has word cats})} = 1$
  - $w_{(\text{has stem cat})} = -1$
    Weight is effectively much smaller than it appears

discriminative models are hard to interpret since the weights do not provide any information

Optimisation

- predicted class for x is: $\underset{y}{\operatorname{argmax}}\ \mathbf{w} \cdot \mathbf{f}(x, y)$
  - prediction derives from a **linear** function of features
  - (other discriminative models may be non-linear)
- weights $\mathbf{w}$ are parameters of the model
- need to find $\mathbf{w}$ that gives good predictive performance
  - faithful representation of training data
  - sensible generalisation to new observations
- many approaches to finding $\mathbf{w}$
  - perceptron: simple; online
  - maximum entropy: principled; probabilistic interpretation
  - support vector machines
  - ...

Perceptron

- the goal is to minimise training errors
- **online** – uses decoding/prediction as part of the training process

- perceptron is not often used in state of the art systems
- but it gives you an intuition around optimising a linear model
- and will be used for Assignment 3 where an online learner is needed

The algorithm

- set weights $\mathbf{w}^{(0)}$ to 0
- for t iterations:
  - for each training instance x, with correct class y:
    - classify x with the current weights $\mathbf{w}^{(t)}$ to find y^*
    - if $y = y^*$, do nothing
    - otherwise, update weights: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{f}(x, y) - \mathbf{f}(x, y^*)$

(y* is the best guess, y* will need to be chosen randomly as w=0)

update weights, former weight + features of true class - features of current predicted class

- y is the correct, gold-standard class from the training data
- y^* is the current model's best guess at the class
- If these are not the same, we try to push the weights towards a better prediction given the same features:
  - inhibit the model choosing y^*
  - encourage the model to choose y

Updating weights

- Three cases for each feature:
  - it's active in correct class but not in the model's class
  - it's active in model's class but not in the correct class
  - it's the same in both classes

$\mathbf{w}^{(t)}$	0	-1	...	2
$\mathbf{f}(x, y)$	1	0	...	0
$\mathbf{f}(x, y^*)$	0	1	...	0
$\mathbf{w}^{(t+1)}$	1	-2	...	2

Perceptron

- decoding:

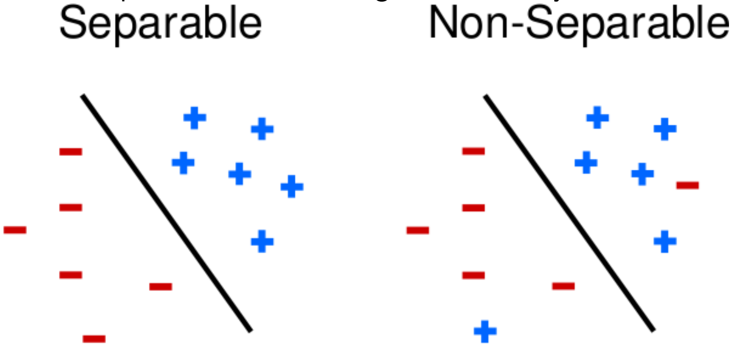
$$y^* = \operatorname*{argmax}_y \mathbf{w} \cdot \mathbf{f}(x, y) \tag{20}$$

- simple and fast to train
- problems with overtraining

Separability

learning a weight vector = learning a line between the classes

for non-separable data, the weights continually move around



http://www.cs.berkeley.edu/~klein/papers/classification-tutorial-naacl2007.pdf

Averaged perceptron

- set weights $\mathbf{w}^{(0)}$ to 0
- for t iterations:
  - for each training instance x, with correct class y:
    - classify x with the current weights $\mathbf{w}^{(t)}$ to find y^*
    - if $y = y^*$, do nothing
    - otherwise, update weights: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{f}(x, y) - \mathbf{f}(x, y^*)$
    - save $\mathbf{w}^{(t)}$
- **calculate the average weights over all $\mathbf{w}^{(t)}$**
- prediction on new data:

$$\underset{y}{\operatorname{argmax}} \sum_t \mathbf{w}^{(t)} \cdot \mathbf{f}(x, y) \tag{21}$$

Part III

Maximum Entropy Models

Maximum Entropy principle

(~ stats: Logistic Regression or
NN: soft max activation with soft entropy loss)

- *Maximum Entropy* modelling:
  - predicts observations from training data (faithful representation)
  - this **does not uniquely identify the model**
- chooses the model which has the most uniform distribution:
  - i.e. the **model with the maximum entropy** (sensible generalisation)

The maximum entropy formulation: constrained optimisation

- Find a model $p(y_i|x_i)$ which
- Is constrained for each feature j:
  the expectation under the model equals the empirical expectation

$$E_p f_j = E_{\tilde{p}} f_j \quad \forall \text{ features } f_j \tag{22}$$

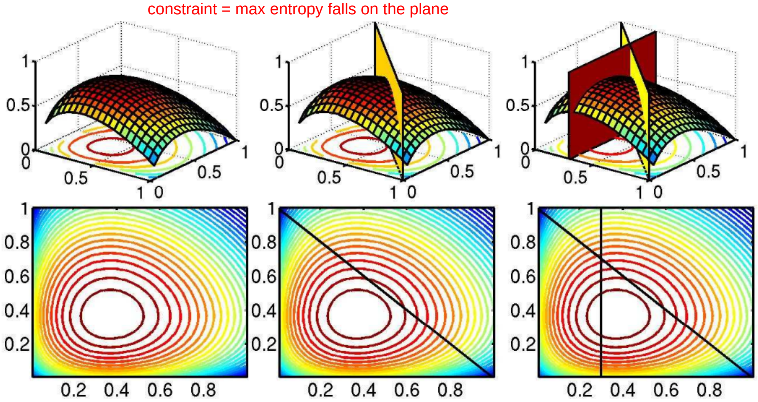
$$\ldots \tag{23}$$

$$\sum_i p(y_i|x_i) f_j(x_i, y_i) = \sum_i f_j(x_i, y_i) \tag{24}$$

all we need to know is the constraints allows us to faithfully represent the data

- Maximises entropy $H(p) = -\sum_x p(x) \log_2 p(x)$
  - a measure of uncertainty ($\therefore$ uniformity) in a distribution

Maximising entropy under constraints



http://www.cs.berkeley.edu/~klein/papers/maxent-tutorial-slides.pdf

Logistic regresssion finds the same weights

- MaxEnt is hard to optimise directly
- But results in the same model as **logistic regression**

Deriving logistic regression

- we want the linear scoring function we saw before

$$y^*(x) = \operatorname*{argmax}_y \sum_t \mathbf{w}^{(t)} \cdot \mathbf{f}(x, y) \tag{25}$$

- we want the model to be probabilistic
- how can we calculate the weights?
- maximise the likelihood of the training data (MLE)

$$L(\mathbf{w}) = \prod_i p(y_i | x_i, \mathbf{w}) \tag{26}$$

we want to find the weight vector W which gives the max probability

Weights

- turn the score function we had previously into a probability distribution

$$p(y|x,\mathbf{w}) = \mathbf{w}\cdot\mathbf{f}(x,y) = \sum_i w_i f_i(x,y) \tag{27}$$

but there are negative weights and we can't have negative probabilities

$$p(y|x,\mathbf{w}) = exp(\mathbf{w}\cdot\mathbf{f}(x,y)) \tag{28}$$

but a probability distribution must sum to 1

$$p(y|x,\mathbf{w}) = \frac{\exp(\mathbf{w}\cdot\mathbf{f}(x,y))}{\sum_{y'}\exp(\mathbf{w}\cdot\mathbf{f}(x,y'))} \tag{29}$$

Maximising the likelihood

- easier to calculate using the log-likelihood:

$$
\begin{aligned}
L(\mathbf{w}) &= \log \prod_i p(y_i | x_i, \mathbf{w}) && (30) \\
&= \log \prod_i \frac{\exp(\mathbf{w} \cdot \mathbf{f}(x_i, y_i))}{\sum_{y'} \exp(\mathbf{w} \cdot \mathbf{f}(x_i, y'))} && (31) \\
&= \sum_i \log \left(\frac{\exp(\mathbf{w} \cdot \mathbf{f}(x_i, y_i))}{\sum_{y'} \exp(\mathbf{w} \cdot \mathbf{f}(x_i, y'))} \right) && (32) \\
&= \sum_i \left(\mathbf{w} \cdot \mathbf{f}(x_i, y_i) - \log \sum_{y'} \exp(\mathbf{w} \cdot \mathbf{f}(x_i, y')) \right) && (33)
\end{aligned}
$$

- **this is an unconstrained optimisation problem**

Convexity

- the surface is convex, because the functions in the sum are convex

$$H(p) = -\sum_{x,y} p(y|x) \log_2 p(y|x) \tag{34}$$

$$L(\mathbf{w}) = \sum_i \left(\mathbf{w} \cdot \mathbf{f}(x_i, y_i) - \log \sum_{y'} \exp(\mathbf{w} \cdot \mathbf{f}(x_i, y')) \right) \tag{35}$$

- **no local maxima**
- this makes the search problem simpler
- e.g. can use gradient descent

Regularization (or smoothing)

- many of the model's weights are optimised based on very little data
  - there are a large number of weights to set
  - the training data can be very sparse
- this leads to a model that is **overfitted** to the training data
- it will perform poorly on unseen data
- Solution: assume a **Gaussian prior** distribution over weights
  - equivalently: add L_2 norm of weight matrix to objective function
  - **L_2 regularisation**
  - penalises weights with large magnitude
  - downplays highly discriminative but unreliable (low freq.) features

Generative and Discriminative Models

- **joint** or **generative** models
  - model **both** observed instance (i.e. outcome) and classification
  - as if class **generated** instance
  - requires modelling probability of an observation
  - language models, Naïve Bayes, hidden markov models, PCFGs
  - trained using (smoothed) Maximum Likelihood Estimate
- **conditional** or **discriminative** models
  - model classification **given** observed instance
  - Maximum Entropy, SVMs, perceptrons
  - trained using (smoothed) conditional Maximum Likelihood Estimate

Other Discriminative Models

- Maximum Entropy models weight individual evidence (features)
- other approaches, e.g. SVMs or perceptrons weight evidence
- although all perform classification, other approaches are harder to interpret as probability distributions over classes
- important when uncertainty is important downstream

Things to do with a model (h/t UMass)

Task	Given	Find
Engineer features/model	$\langle x, y \rangle_i$	$\mathbf{f}(x, y, \mathbf{w})$
Estimate parameters	$\mathbf{f}(x, y, \mathbf{w}), \langle x, y \rangle_i$	$\mathbf{w}$
Decode/predict	$\mathbf{f}(x, y, \mathbf{w}),\ x,\ \mathbf{w}$	y^* for x

Take away

- Feature representation of text
- Feature engineering
- Generative vs discriminative models
- Linear weighting of features
- Perceptron algorithm and averaged/voted variants
- Terms: parameters, estimation, decoding
- Again: faithfulness vs generalisation