Multiclass classification

Kairit Sirts

04.04.2014

Multiclass classification

We have already seen artificial neural networks, but there are many more methods:

- One versus all
- All versus all
- Classification tree
- Naïve Bayes
- Maximum entropy model (multiclass logistic regression)

One versus all (OVA) multiclass classification

- Assume we have K classes and a binary classifier (eg logistic regression)
- Train K binary classifiers, such that for each classifier:
  - Data labelled with C_k is treated as positive examples
  - Data with all other labels is treated as negative examples
- For predicting the class of a new example:
  - Predict the label with each classifier.
  - Add the result ($+1$ or -1) to the score vector respective component.
  - Final prediction is the class with largest score with ties broken randomly.

All versus all (AVA) multiclass classification

- Assume we have K classes and a binary classifier (eg logistic regression)
- Train $K(K-1)/2$ binary classifiers, such that for each i-th and j-th class pair:
  - Treat the data with ith class label as positive examples
  - Treat the data with jth class label as negative examples
- For predicting the class of a new example:
  - Predict the label with each classifier.
  - For positive prediction the ith class gets a point, for negative prediction the point goes to the jth class
  - Final prediction is the class with the largest score.

Classification tree

- Build binary tree of binary classifiers
- With K classes $K - 1$ classifiers are necessary
- At the root, half of the classes are considered positive and the other half negative
- Need to know the data for deciding how to organize the classes in the tree.

Bayes theorem

- Let's assume we have k classes.
- The **posterior probability** of a class C_k for an input $\mathbf{x}$ is:

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})},$$

- where $p(\mathbf{x}|C_k)$ is the likelihood, $p(C_k)$ is the **prior probability** and $p(\mathbf{x})$ is the **marginal data likelihood**.
- $p(C_k)$ is the probability of a class C_k *a priori*, before seeing any data.
- $p(C_k|\mathbf{x})$ is the class probability *a posteriori*, after observing the data.
- Bayes theorem updates the prior distribution into posterior using the evidence - observed data.

Independence of variables

- If X and Y are **unconditionally independent** then their joint distribution is the product of marginal distributions:

$$X \perp Y \iff p(X,Y) = p(X)p(Y)$$

- Unconditional independence is rare, mostly variables influence each other.
- If this influence is mediated through a third variable Z, then X and Y are **conditionally independent**.

$$X \perp Y|Z \iff p(X,Y|Z) = p(X|Z)p(Y|Z)$$

- Conditional independence does not imply unconditional independence and vice versa:

$$X \perp Y|Z \not\iff X \perp Y$$

Spam detection

- Inputs $\mathbf{x}$ are emails (documents).
- We have m labeled training pairs $(\mathbf{x}_i, y_i)$, $y_i \in \{0, 1\}$
- Want to classify a new email as spam 1 or not spam 0
- According to Bayes rule:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} \propto p(\mathbf{x}|y)p(y)$$

- We can omit the denominator $p(\mathbf{x}$, because this does not involve y.
- In general, the denominator can be computed as:

$$p(\mathbf{x}) = \sum_{y'} p(\mathbf{x}|y')p(y')$$

Feature representation

- Computing likelihood $p(\mathbf{x}|y)$ is hard, because we will never have enough training data to estimate this distribution reliably.
- Instead represent the document as a "bag-of-words":
  - Choose vocabulary V
  - Present each input as $|V|$-dimensional vector, where each position corresponds to a word in vocabulary
  - In each position the value is 1, if the corresponding word appears in the input and 0 otherwise.
- Now the likelihood can be computed as:

$$p(\mathbf{x}|y) = \prod_{j=1}^{|V|} p(x_j|y)$$

Naïve Bayes assumption

- We have:

$$p(\mathbf{x}|y) = \prod_{j=1}^{n} p(x_j|y)$$

- We essentially assume that the features (words) are **conditionally independent given the class label**
- This is called **naïve Bayes assumption**
- The model is called "naive", because we do not expect features actually to be independent nor conditionally independent.
- Still this model usually performs quite well, because it has relatively few parameters ($O(Kn)$) and is thus relatively immune to overfitting.
- Feature representation is lossy, original document cannot be constructed from it.

Naïve Bayes model

- The model has the following parameters:

$$\theta_{j|y=1} = p(x_1 = 1|y = 1)$$
$$\theta_{j|y=0} = p(x_1 = 1|y = 0)$$
$$\theta_y = p(y = 1)$$

- The MLE estimates for the parameters are:

$$\theta_{j|y=1} = \frac{\sum_{i=1}^{m} \mathbb{I}(x_{ij} = 1, y_i = 1)}{\sum_{i=1}^{m} \mathbb{I}(y_i = 1)}$$

$$\theta_{j|y=0} = \frac{\sum_{i=1}^{m} \mathbb{I}(x_{ij} = 1, y_i = 0)}{\sum_{i=1}^{m} \mathbb{I}(y_i = 0)}$$

$$\theta_y = \frac{\sum_{i=1}^{m} \mathbb{I}(y_i = 1)}{m}$$

Prediction with naïve Bayes

- We want to find wether a new email $\mathbf{x}$ is spam or not spam.
- We use Bayes theorem

$$p(y=1|\mathbf{x},\boldsymbol{\theta}) \propto p(\mathbf{x}|y,\boldsymbol{\theta})p(y|\boldsymbol{\theta}) = p(y=1|\boldsymbol{\theta}) \prod_{j=1}^{n} p(x_{ij}|y=1,\boldsymbol{\theta})$$

$$p(y=0|\mathbf{x},\boldsymbol{\theta}) \propto p(\mathbf{x}|y,\boldsymbol{\theta})p(y|\boldsymbol{\theta}) = p(y=0|\boldsymbol{\theta}) \prod_{j=1}^{n} p(x_{ij}|y=0,\boldsymbol{\theta})$$

- We predict the class with highest posterior probability:

$$y^* = \arg \max_{y \in \{0,1\}} p(y|\mathbf{x},\boldsymbol{\theta})$$

Different types of features

- For real-valused features we can fit a Gaussian:

$$p(\mathbf{x}|y=k) = \prod_{j=1}^{n} \mathcal{N}(x_j|\mu_{kj}, \sigma_{kj}^2)$$

- For features with categorical values we can use multinomial distribution:

$$p(\mathbf{x}|y=k) = \prod_{j=1}^{n} \text{Multi}(x_j|\theta_{kj})$$

- Continuous values can also be discretized and modeled as multinomial random variables

Problem with MLE estimates

- What if the test email contains a word with index w that is in vocabulary but was never observed in the training set?
- The parameters involving this word are:

$$p(x_w|y=1) = \frac{\sum_{i=1}^{m} \mathbb{I}(x_{iw}=1, y_i=1)}{\sum_{i=1}^{m} \mathbb{I}(y_i=1)} = 0$$

$$p(x_w|y=0) = \frac{\sum_{i=1}^{m} \mathbb{I}(x_{iw}=1, y_i=0)}{\sum_{i=1}^{m} \mathbb{I}(y_i=0)} = 0$$

- Thus, posterior probabilities for predicting class are 0, because these formulas always have $p(x_w|y=1)$ or $p(x_w|y=0)$ in the product.

Smoothing

- Generally, it is bad idea to estimate any probability to 0 just because we havent' seen some data in the training set.
- This problem can be overcome with **smoothing**
- The general idea of smoothing is to take away some probability mass from the observed values and to preserve it to the unobserved values.

Add-one smoothing

- **Add-one smoothing** is one of the simplest techniques
- We add one for every parameter in the numerator and number of classes k in the denominator

$$\theta_{j|y=1} = \frac{\sum_{i=1}^{m} \mathbb{I}(x_{ij} = 1, y_i = 1) + 1}{\sum_{i=1}^{m} \mathbb{I}(y_i = 1) + 2}$$

$$\theta_{j|y=0} = \frac{\sum_{i=1}^{m} \mathbb{I}(x_{ij} = 1, y_i = 0) + 1}{\sum_{i=1}^{m} \mathbb{I}(y_i = 0) + 2}$$

- You can check that the probabilities still sum to one.
- Add-one smoothing can be generalized so that instead of 1 we add a value of a parameter α

Maximum entropy model

- Essentially multiclass logistic regression
- Also known ad log-linear model
- Allows overlapping features
- Widely used in practical machine learning

Definitions

- Goal is to learn a model for multiclass classification. We have:
  - set of possible inputs X
  - finite set of possible classes Y
  - number of features in the model d
  - a function $f : (X, Y) \to \mathbb{R}^d$ that maps any (x, y) pair to a feature vector $f(x, y)$
  - a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$
- The probability of y conditioned on x, given the model parameters $\boldsymbol{\theta}$ is defined as:

$$p(y|x; \boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta}^T f(x, y)\right)}{\sum_{y' \in Y} \exp\left(\boldsymbol{\theta}^T f(x, y')\right)}$$

Features

- For any pair (x, y), $f(x, y) \in \mathbb{R}^d$ is a feature vector for this pair
- Each component $f_k(x, y)$ in this vector is a feature
- Features allow us to represent different properties on input x in conjuction with y
- Often binary features are used, that is $f_k(x, y) \in \{0, 1\}$
- Consider text classification example. One feature function could be:

$$f(x, y) = \begin{cases} 1, \text{if } x \text{ contains "football" and } y \text{ is "sport"} \\ 0, \text{otherwise} \end{cases}$$

- Typically, features are generated from feature templates. E.g. in text classification we would generate such features for all words and classes that occur in the training data.

More complex features

- Consider the problem of classifying person names according to language
- Training data is in pairs: (Jüri, EST), (James, ENG), (Veronique, FRA).
- What features could we use:
  - Name features:

$$f(x, y) = 1 \text{ if } x = \text{“Veronique”} \text{ and } y = \text{“FRA”}$$

  - Suffix features, e.g.

$$f(x, y) = 1 \text{ if } x \text{ ends with “que” and } y = \text{“FRA”}$$

  - Prefix features:

$$f(x, y) = 1 \text{ if } x \text{ starts with “Jü” and } y = \text{“EST”}$$

  - Character features:

$$f(x, y) = 1 \text{ if } x \text{ contains “ü” and } y = \text{“EST”}$$

Some motivation for the model formula

$$p(y|x;\boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta}^T f(x,y)\right)}{\sum_{y' \in Y} \exp\left(\boldsymbol{\theta}^T f(x,y')\right)}$$

- The exponent $\boldsymbol{\theta}^T f(x,y)$ can have any value depending on the active features and model parameters
- It can be interpreted as a measure of plausibility of a class y given input x
- This measure can be computed for all classes y for any input x
- we would like to transform those measures to a well-formed distribution $p(y|x)$
- This can be done by first exponentiating which guarantees that the result is always larger than zero
- Finally we normalize to ensure the probabilities sum to 1.

Estimating parameters

- The log-likelihood of the training data is:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} p(y_i|x_i;\boldsymbol{\theta})$$

- MLE solution is the one that maximizes log-likelihood:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^d} \ell(\boldsymbol{\theta})$$

- Unfortunately, there is no analytical solution for finding $\boldsymbol{\theta}^*$, therefore iterative methods have to be used.
- We could use the already familiar gradient ascent
- In practice, usually second order methods (e.g. L-BFGS) are used.

Partial derivatives

- Partial derivatives take the following form:

$$\frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{m} f_j(x_i, y_i) - \sum_{i=1}^{m} \sum_{y} p(y|x_i; \boldsymbol{\theta}) f_j(x_i, y)$$

- The first part $\sum_{i=1}^{m} f_j(x_i, y_i)$ is the sum of the j-th observed feature value across the training data
- The second part is the sum of the expected features values across the training data using the current model parameters.

Regularization

- Usually, it is highly beneficial to add the regularization term:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{m} p(y_i|x_i;\boldsymbol{\theta}) - \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$$

- Regularization penalizes large parameter values and thus avoids overfitting
- Partial derivatives with regularization:

$$\frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{m} f_j(x_i, y_i) - \sum_{i=1}^{m}\sum_{y} p(y|x_i;\boldsymbol{\theta}) f_j(x_i, y) - \lambda\theta_j$$