IT496: Introduction to Data Mining

Lecture 17

Multinomial Logistic Regression

Arpit Rana

22nd September 2023

Multiclass Logistic Regression

Suppose there are more than two classes. Logistic Regression is one of the few classifiers that can directly handle multiclass classification.

There are three solutions:

- One-versus-Rest
- One-versus-One
- Multinomial Logistic Regression

One-vs-Rest (One-vs-All)

One-versus-Rest (also called 'one-versus-all') involves training |C| binary classifiers, one per class

- for each class $c \in C$
  - create a copy of the training set in which you replace examples (***x***, c) by (***x***, 1) and examples (***x***, c') where c' ≠ c by (***x***, 0).
  - train a binary classifier h^c on this modified training set
- After all these classifiers have been trained, to classify ***x***, we run all the classifiers h^c for each $c \in C$
  - The predicted class of ***x*** is the class c whose classifier h^c predicts 1 with the highest probability.

How many classifiers would we end up building for a dataset where there are three classes?

One-vs-One (Pairwise Classification)

In **One-versus-One** (also called 'pairwise classification'),

- we build a classifier for every pair of classes using only the training data for those two classes.
- after all these classifiers have been trained, when we want to classify $\boldsymbol{x}$, we run all the classifiers and, for each class c, we count how many of the classifiers predict that class.
  - The predicted class of $\boldsymbol{x}$ is the one that is predicted most often.

One-versus-One's advantage over One-versus-Rest is that the individual classifiers do not need to be classifiers that produce probabilities.

Its disadvantage is the number of individual classifiers it must train:

- How many for a dataset that has three classes?
- In terms of $|C|$, how many in general?

Multinomial Logistic Regression

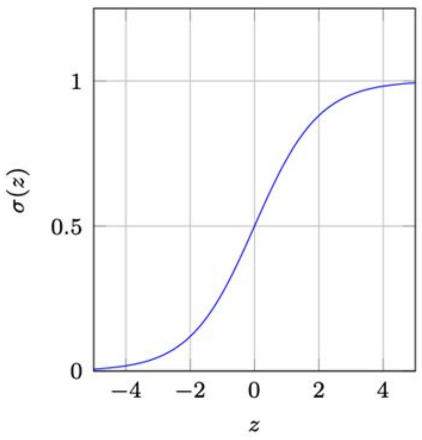
We must modify how the classifier works and the loss function for training.

- Now, instead of one vector of coefficients, β, the classifier has one per class, β_c for each c $\in$ **C**.
- Hence, instead of computing one value, $\boldsymbol{x}\beta$, it computes one per class, $\boldsymbol{x}\beta_c$ for each c $\in$ **C**.
- Now, 'squashing' is more complicated:
    - Not only must each of these values be squashed to [0, 1];
    - but they must also sum to 1.
- So we do not use the *sigmoid function.*

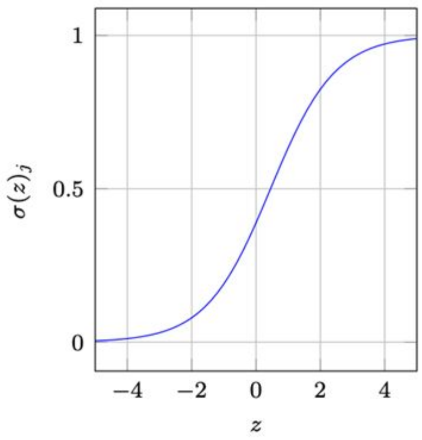
Multinomial Logistic Regression

We use the **softmax function** (but still often designated by $\boldsymbol{\sigma}$)

$$P(\hat{y} = c \,|\, x) = \sigma(x\beta_c) = \frac{e^{x\beta_c}}{\sum_{c \in C} e^{x\beta_c}}$$



Sigmoid Function



Softmax Function

Multinomial Logistic Regression

We use the **softmax function** (but still often designated by $\boldsymbol{\sigma}$)

$$P(\hat{y} = c \,|\, x) = \sigma(x\beta_c) = \frac{e^{x\beta_c}}{\sum_{c \in C} e^{x\beta_c}}$$

- If we put all the different β_c into a single matrix **B**, then we can have a vectorized implementation of this.
- Finally, there is no thresholding this time: the classifier simply predicts the class with the highest estimated probability as below.

$$\underset{c \in C}{\arg\max}\ (\,\sigma(x\beta_c))$$

<u>Tip:</u> To work out the probabilities, you need the softmax formula given above. But if all you want to know the winner, then all you need is $\underset{c \in C}{\arg\max}\ (x\beta_c)$ because the rest of the softmax formula makes no difference to the ordering.

Loss Function

So how does logistic regression learn all these different β_c?

- We need a new loss function: the **cross-entropy loss function** (or sometimes the **categorical cross-entropy function**)

$$J(X, y, B) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c \in C} I\left(y^{(i)} = c\right) \log\left(\sigma\left(x^{(i)}\beta_c\right)\right)$$

  where $I(p)$ is the indicator function that outputs 1 if predicate p is true and zero otherwise.

- The easiest way to get some grasp of this is to realise that when there are just two classes, it is equivalent to the loss function we used earlier.

Multiclass Logistic Regression

Which one is better?

- Sometimes, even when you have a classifier, such as Logistic Regression, that can directly handle multiclass classification, using it in a One-versus-One fashion can be more accurate! (Do you have any ideas why?)
- But one-versus-rest and one-versus-one have higher training costs.

Multiclass Logistic Regression in Scikit-learn

<u>FYR</u>:

- If there are more than two classes, scikit-learn will handle them without you needing to do anything.
- For most of scikit-learn's binary classifiers, it will use one-versus-rest.
- In the case of of scikit-learn's `LogisticRegression` class, it will use Multinomial Logistic Regression — the method explained in the earlier slides.
- For scikit-learn's binary classifiers, if you want to use one-versus-rest, then set `multi-class="ovr"` (which is often the default, but is not the default in the case of `LogisticRegression`).
- There are also classes `OneVsRestClassifier` and `OneVsOneClassifier`.

Instance-based Learning

kNN Regressor/Classifier

Instance-based Learning

Instance-based learners learn by heart: they simply store the examples in the labeled dataset.

- The way they generalize is using *similarity* (or *distance*):
  - given an unseen example $\boldsymbol{x}$,
  - they predict $\hat{y}$
  - from the y-values of examples in the dataset that are similar to $\boldsymbol{x}$.

Let's look at two concrete examples of this: *nearest-neighbour regression* and $\boldsymbol{k}$*-nearest-neighbours regression.*

Nearest Neighbour Regression

To predict the target value $\hat{y}$ for unseen example $\boldsymbol{x}$,

- we find the example ($\boldsymbol{x}'$, y') in the labeled dataset whose distance from $\boldsymbol{x}$ is smallest; and
- we use y' as our prediction.

We also refer to this as a 1-*nearest-neighbour regressor* or just 1NN or *k*NN for *k*=1.

Nearest Neighbour Regression

The problems with **1NN** is that it can be incorrectly influenced by noisy examples:

- If there are examples in the labeled dataset where we have ***incorrectly recorded the feature values***, then we may not find the best example from which to make our prediction.
- If there are examples in the labeled dataset where we have ***incorrectly recorded the target value***, then these will result in incorrect predictions.

k-Nearest-Neighbours Regression

To reduce the influence of noisy examples, we use more than one neighbour:

- we find k examples whose distance from unseen example $\boldsymbol{x}$ is smallest; and
- we use the mean of their y-values as our prediction.

We abbreviate the name of this to kNN, e.g. 3NN is where we use 3 nearest-neighbours.

k-Nearest-Neighbours Regression

There are many variants of kNN.

- A common one, for example, is to use a weighted average of the neighbour's target values.
- The weights could be the inverse of the distances so that more similar examples count for more.

We don't need to implement them for ourselves: scikit-learn has a class `KNeighborsRegressor`.

k-Nearest-Neighbours Classifier

We can use instance-based learning for classification.

As before, we find the *k*-nearest-neighbours.

- For ***regression***, we took the mean of the neighbours' **y**-values.
- For ***classification***, we take a vote: the class with the majority vote wins.
- E.g. to classify Craig, we find 3 similar students. If two of them passed, we predict Craig will pass. Otherwise, we predict Craig will fail.

Why for kNN classification, do we often choose *k* to be an odd number?

k-Nearest-Neighbours Classifier

There are lots of variants of this,

- e.g. we can have weighted majority vote, where the closer a neighbour is, the greater the weight of its vote.

scikit-learn has a class that implements kNN classifier: `KNeighborsClassifier`

Next lecture

Support Vector Machines

25^{th} September 2023