Introduction to Machine Learning (by Implementation)

Lecture 6: Multinomial Logistic Regression

Ian J. Watson

University of Seoul

University of Seoul Graduate Course 2019

Housekeeping

- Next week is KPS and also midterm exam week
- We won't have midterms, I will be at KPS, so we won't have class
- I know people need to prepare, so I will mark this weeks work just before the next class in 2 weeks time
- You can leave after the lecture if you have preparations to do

Introduction

- Last week, we looked at classification by logistic regression
  - Idea: build a classifier that gives 0 for one class, 1 for a second class
  - Use multiple regression on the input variables, then pass the output through a logistic function (turn-on curve)
  - Maximize the classifier by maximizing the log-likelihood function
  - $\log L(\beta|x_i, y_i) = y_i \log f(x_i, \beta) + (1 - y_i) \log(1 - f(x_i, \beta))$
  - That is, $f(x_i, \beta)$ is being interpreted as classifier's probability for x_i belonging to class 1, so we maximized
    if $y_i = 1$, $\log L(\beta|x_i, 1) = \log f(x_i, \beta)$,
    if $y_i = 0$, $\log L(\beta|x_i, 0) = \log(1 - f(x_i, \beta))$
    where $f(x_i, \beta) = \frac{1}{1+e^{-x_i \cdot \beta}}$
- We were restricted to looking at two categories
- Today, we will build on this to classify into several categories
- First, lets talk about regularization

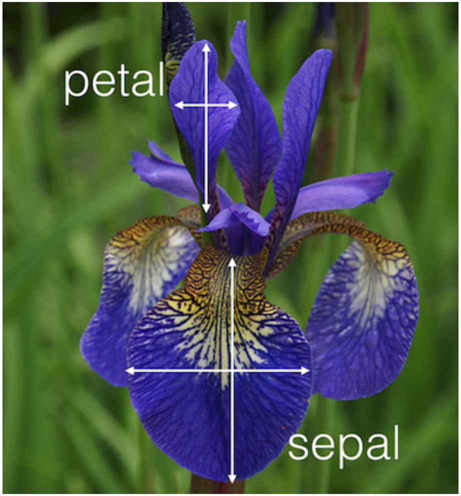
Regularization

- Should have seen in the last weeks that the gradient descent can be very sensitive to the initial parameters
- Due to large, difficult phase space when maximizing the functions over hundreds of measurements with additional random errors thrown in
- To deal with this, and tame the beast, we can *regularize* the model
    - That is, add additional terms to the likelihood to control the size of the parameters, tame instabilities and prevent overfitting
- Today we will look at implementing *ridge regression*, then we will talk about *multinomial regression* or multi-label regression

Ridge Regression

- Last week, we maximized the likelihood function of the dataset, or equivalently, minimized the sum of the negative log likelihood
  - Probabilities for each datapoint multiply, so you must add datapoint likelihoods in the log
- We can add a Gaussian constraint to the β terms, to keep them near 0
  - Equivalent to summing the square of $\hat{\beta}$ when thinking of log-likelihood
  - Take the log-likelihood from last week, add $\kappa \hat{\beta} \cdot \hat{\beta}$, take the gradient log-likelihood from last week, add $2 \kappa \beta_i$ to the i'th component
- Keeps the β small unless they are overcome by the improvement in the likelihood
  - So also has the effect of telling you which variables are actually important, those that only affect the likelihood in a minor way will stay near 0, and only those that give great improvement (good description of the data), will move away from 0
- Introduce a new parameter κ which controls the "strength" of the restoring force, the larger the κ, the harder it is to overcome
  - Nb. usually κ is called λ, but in python `lambda` is a special word
- Convergence should be much faster and more stable!

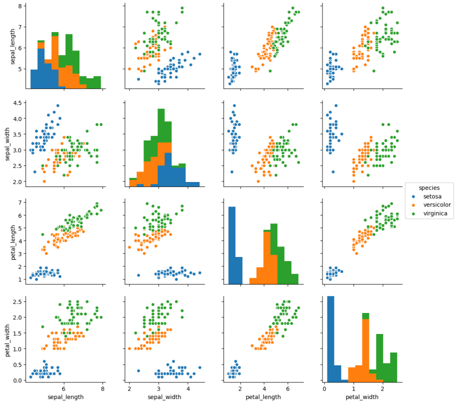
Multinomial Example: Fisher's Irises



- The iris dataset is a classic classification task, first studied by Fisher in 1936.
- The goal is, given features measured from a particular iris, classify it into one of three species
  - Iris setosa, virginica, versicolor.
- The variables are: Sepal width and length, petal width and length (all in cm).

Fisher's Irises: dataset

Lets view the basic variables we have. Setosa (blue) looks easily separable by the petal length and width, but versicolor and virginica are a little tricky.



- We can, however, see that versicolor and virginica could probably be separated well by a logistic classifier, but then it will mix up setosa (single turn on curve)
- But, how could we build a classifier to distinguish all 3 categories at once?
  - This is *multinomial classification*

Probability Vector and One-hot encoding

- We want our classifier able to say which of several categories a datapoint belongs to: 0 = setosa, 1 = virginica, 2 = setosa
- We could label categories by integers, 0, 1, 2, 3, but then how to interpret an output?
  - What would a classifier output of 2.4 mean? Close to 2 but also 3? What about 1, its right out?
- Need an alternate way to encode the output
- We will make a vector the size of the number of categories
  - For Fisher's irises, this means a 3-vector
- We require the sum of entries to be 1, and then interpret each entry as the probability of belonging to a particular category
  - (0.9, 0.1, 0) is 90% prob. for setosa, 10% virginica, and 0% versicolor
- This relates to a *one-hot encoding*: at truth level everything will be 0, except one position representing the true category, which is 1
  - We know our example is setosa, so the y "truth" vector will be (1, 0, 0)
- Given the probability vector, we will interpret the highest-probability entry as the prediction of the classifier

Multinomial Logistic Regression

- Given k non-overlapping categories we will build logistic classifiers to build a scheme distinguish all k categories, $P(i)$ for prob. to be cat. i
- Build $k-1$ logistic classifiers, comparing category i to category 0 for $i \in [1, 2 \ldots k-1]$, Category 0 is our *reference category*
- This will give $k-1$ $\hat{\beta}_i$, one for each of the non-reference categories
- The logistic classifiers compare one category against 0: $P(i|0 \vee i) = \frac{1}{1+e^{-\beta_i \cdot x}}$ and $P(0|0 \vee i) = \frac{e^{-\beta_i \cdot x}}{1+e^{-\beta_i \cdot x}}$ so $\frac{P(i|0 \vee i)}{P(0|0 \vee i)} = e^{\beta_i \cdot x}$
  - Introduce the requirement that in the full joint-classifier, this probability ratio must still hold
- So, joining together all the non-reference categories:
  - $P(1) = P(0)e^{\beta_1 \cdot x}$, $P(2) = P(0)e^{\beta_2 \cdot x}$, $\ldots$, $P(k) = P(0)e^{\beta_k \cdot x}$
- We also have the requirement that the probability must sum to 1
  - $P(0) + P(1) + \ldots + P(k-1) = 1$
- Thus, for the reference $P(0) = 1 - \sum_{i \neq 0} P(i) = 1 - P(0) \sum_{i \neq 0} e^{\beta_i \cdot x}$
- Rearranging gives $P(0) = \frac{1}{1+\sum_{i \neq 0} e^{\beta_i \cdot x_i}}$, and so $P(i) = \frac{e^{\beta_i \cdot x_i}}{1+\sum_{j \neq 0} e^{\beta_j \cdot x_j}}$
- So, if we train logistic classifiers to distinguish one category against the reference, we can join them all together to form a k-category classifier

Exercises: ridge regression

- Take your `logistic_regression_sgd` from last week, and modify it: `logistic_ridge_regression_sgd(x0, x1, alpha0, kappa, iterations)`
  - The `f` given to the stochastic minimizer should be $NLL + \kappa|\hat{\beta}|^2$
  - The `df` given to the stochastic minimizer should be $\nabla NLL + 2\kappa\hat{\beta}$ (note, vector addition)
- Use `logistic_ridge_regression_sgd` to find beta for the Iris' in category 0 vs 1 and 0 vs 2. Check the accuracy and then dump the parameters you find into `results_01.txt` and `results_02.txt`
  - The file `Fisher.txt` has the data, the first column is the true classification, the rest are the independent variables mentioned earlier
  - The first column has 0 for setosa, 1 for virginica, 2 for versicolor
- Do the same with categories 0 vs 2 and 1 vs 2, dump the params into `results_02.txt`, `results_12.txt`

Exercises: multinomial logistic regression

- By now, you should have something like an `inner(x, beta)` function, which calculates our $\beta \cdot x$ output: $\beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$
- Write `multi_logistic(x, betas)` which takes in $k-1$ $\hat{\beta}$ and outputs k results:
  - $\left(\frac{1}{1+\sum_i e^{\hat{\beta}_i \cdot \hat{x}}}, \frac{e^{\hat{\beta}_1 \cdot \hat{x}}}{1+\sum_i e^{\hat{\beta}_i \cdot \hat{x}}}, \ldots, \frac{e^{\hat{\beta}_{k-1} \cdot \hat{x}}}{1+\sum_i e^{\hat{\beta}_i \cdot \hat{x}}}\right)$
  - Where, when we do below, each β_i is the result of a i vs 0 regression
- Write `multi_best(x)` which takes in the probability vector and returns a one-hot encoded output of the highest probability output
- Write `multi_accuracy(x, y, betas)` which finds the accuracy (use `multi_best`) of the multinomial logistic regression using the `betas`
  - The `y` should be one-hot encoded
- Output the accuracy of a multinomial regression on 0 vs 1 + 0 vs 2, and 0 vs 2 + 1 vs 2, using the β values from the ridge regression. Which gives a better result, and why? Write the value of `multi_accuracy` and your answer to "why" in a file `multi.txt`
  - You will need to think carefully about the one-hot encoding
  - Think carefully about which category is the reference in each case