W261-Fall 2018 Final Project

Click Through Rate Prediction on Display Ads

Ben Thompson, Kevin Gifford, Dan VanLunen, Matt Prout

Intro

- Click-through Rate (CTR) = $
- Scalability of problem solving
  - Work locally when possible
  - Work with smaller datasets to iterate faster
  - Divide and conquer
- Balance choosing a model we can implement vs. best predictions
- Balance creating a pipeline that can scale vs. perfect transformations

Logistic Regression

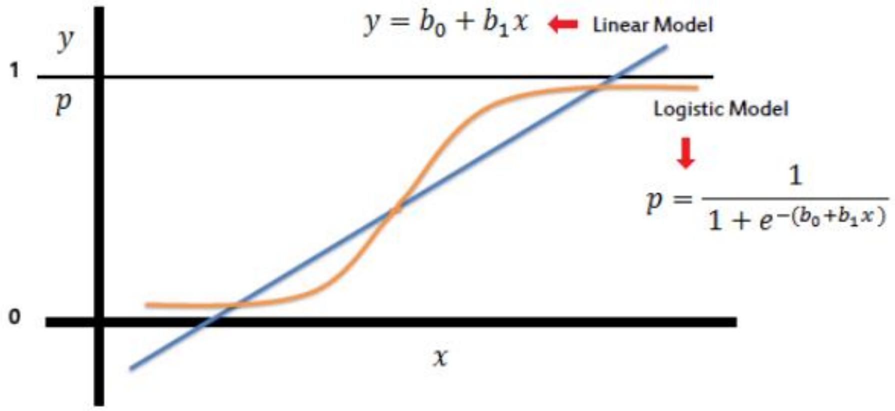
$$\ln \frac{p}{1-p} = \beta X + e$$

Log odds are a linear function of the parameters

Parameters have easy interpretation

Better for estimating binary class probability than linear model because fixed between (0,1)

Linear separability assumption likely incorrect so worse predictions, but with many features can still do well



Logistic Regression

No closed form solution for the maximum likelihood

Gradient ascent works for log likelihood

Embarrassingly parallel sum made of independent pieces

Can add in regularization easily as well

$$L(\beta) = \Pi_{i=1}^{N} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

$$l(\beta) = \sum_{i=1}^{N} y_i \ln p(x_i) + (1 - y_i) \ln(1 - p(x_i))$$

$$= \sum_{i=1}^{N} y_i \beta x_i - \beta x_i - \ln(1 + e^{-\beta x_i})$$

$$\nabla_\beta l = \sum_{i=1}^{N} x_i (y_i - \frac{1}{1 + e^{-\beta x_i}})$$

$$J = l(\beta) - \lambda \sum_{j=1}^{m} \beta_j^2$$

$$\nabla_\beta J = \sum_{i=1}^{N} x_i (y_i - \frac{1}{1 + e^{-\beta x_i}}) - 2\lambda\beta_j$$

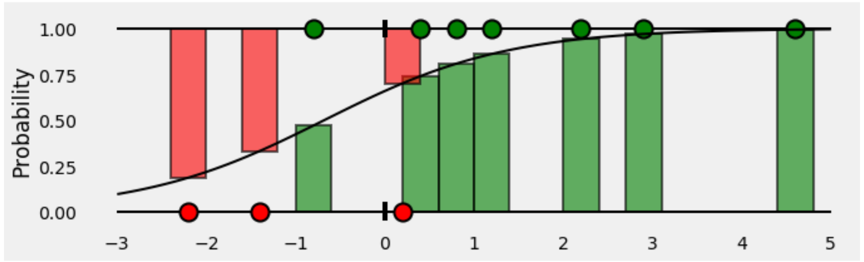
Logistic Regression

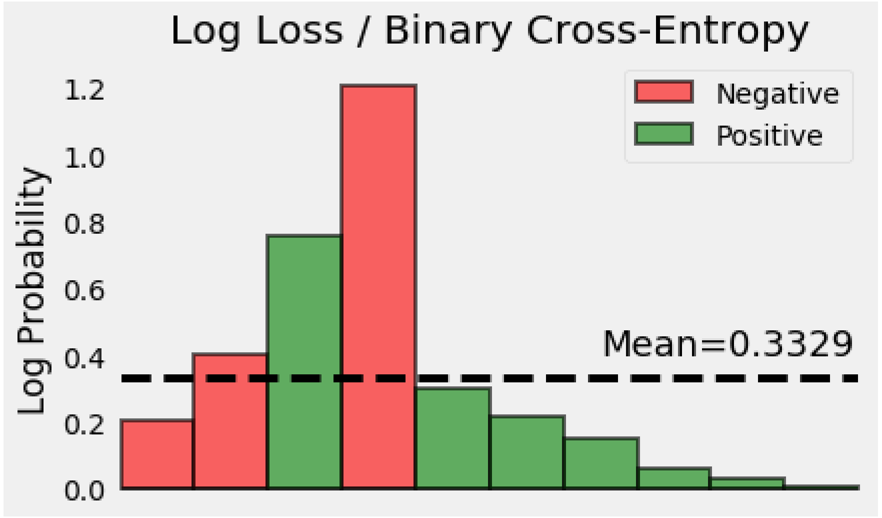
$$\hat{p} = \frac{1}{1 + e^{-\hat{\beta} x_i}}$$

$$logloss = \frac{1}{N} \sum_{i=1}^{N} -y_i \log \hat{p}_i - (1 - y_i) \log(1 - \hat{p}_i)$$

Predictions and Loss

Logloss takes into account confidence





EDA

- Examined the class balance
    - Noticed that is subsampled (approximately 1 click : 3 no clicks), so it is more balanced for training.
- Looked at the correlation between the numerical features and label
    - This can indicate whether a feature is predictive of the outcome, also help us avoid features that are collinear in our final model.
- Examined every feature that we were given
    - There are 13 numerical fields, 26 categorical fields, 1 label
    - Missing values needed to be handled, as logistic regression cannot handle missing data
        - We looked at the number of null values for each feature
        - We dropped the column if there were too many (> 5%) missing values
        - If the number of missing values was small, we imputed if the field was numeric, or dropped the observations if the field was categorical

EDA, continued

- Examined every feature that we were given, cont
    - Categorical fields needed to be encoded so they could be used by logistic regression
        - We looked at the number of unique values for each field
        - If the field had a small number of unique values, we would one-hot-encode (OHE) using the top 10 most frequent values
        - If the field had a large number of unique values, we would use the ‘hash trick’
- Created histograms and boxplots for click/no click conditions
    - They gave a visual picture of the distribution of the data, and also whether there was any obvious difference for when the user clicked the data v. when they did not.

Data Cleaning

- The data cleaning was guided by the EDA
- We used DataFrames as the data came as a table, and it made it easier to manage the fields to be transformed
- Dropping fields was a matter of calling a function in the DataFrame API call for each column
- Since we are using gradient descent, the numerical data needed to be scaled to a common range or normalized.
- We wrote the normalizing and imputation code, which required several passes on the data
    - The first pass determined the means and standard deviations of the selected features
    - For each field with null values to impute, we impute with the mean
    - A final pass normalized the features

Data Cleaning, continued

- We wrote the code to encode the categorical data which required two passes on the data
    - The first pass through the data determined the top N values for each OHE category, and stored these values in a dictionary
    - The second pass converted categories to their respective OHE or feature hashed representation
    - The final form was a numpy array which could be used by the model for training / testing

Training

- Initial training conducted on 1% sample of the data to establish preliminary coefficients, using both L1 and L2 regularization.
- Preliminary coefficients used as priors for training the full dataset. Using this method, training the full model required many fewer iterations to converge than training the sampled dataset.
- Using L2 regularization yielded better log loss performance, faster convergence, and easier tuning than L1 regularization.

Conclusion:

- Divide and conquer works very well!
- Cross-check everything
  - Runtime on cluster vs. local
  - Loss vs. accuracy
  - Transformations on 1% vs. 100%
- Think ahead
  - What can I cache
  - How can I reuse, or piggy back on, functions
  - What is easiest path to deciding my next step