

Review of linear models

Linear and logistic regression

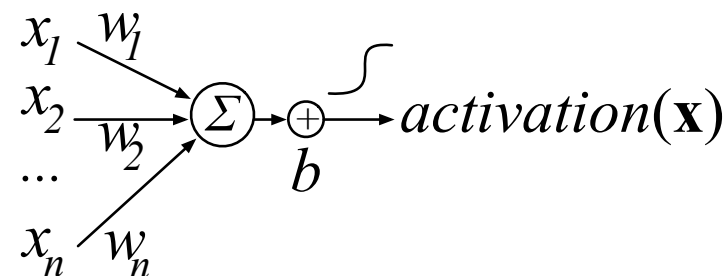
Terence Parr

MSDS program

University of San Francisco

Why do we study linear models?

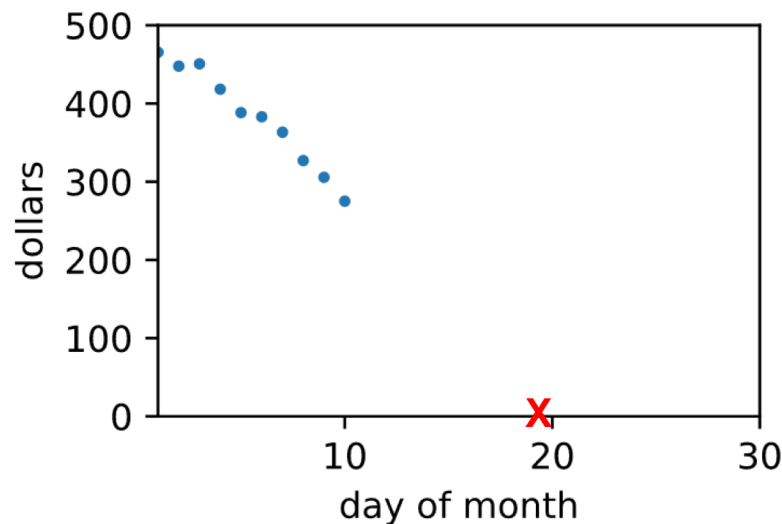
- Simple, interpretable, super fast, can't be beat for linear relationships
- Usually a lower bound on power but they often form the basis of other more powerful techniques, such as LOESS and...
- Combining multiple linear models into a lattice with a nonlinear function as glue yields a neural network; those are insanely useful and powerful
- Logistic regression model is a 1-neuron neural network
- LM can only find separating hyperplane and classes must be contiguous, which is rarely true for more than 1 or 2 vars



Linear regression

What problem are we solving?

- In college, I was given a fixed \$500 for food every month
- I wanted to know, at current rate of pizza consumption, how fast I'd run out of money so I plotted it and “eyeballed” zero x point

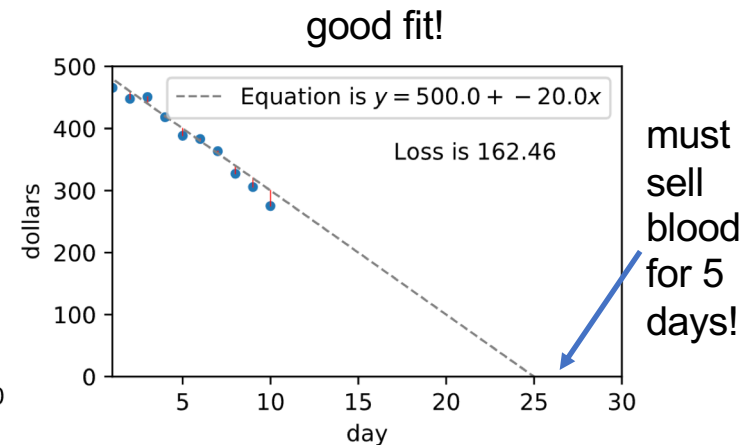
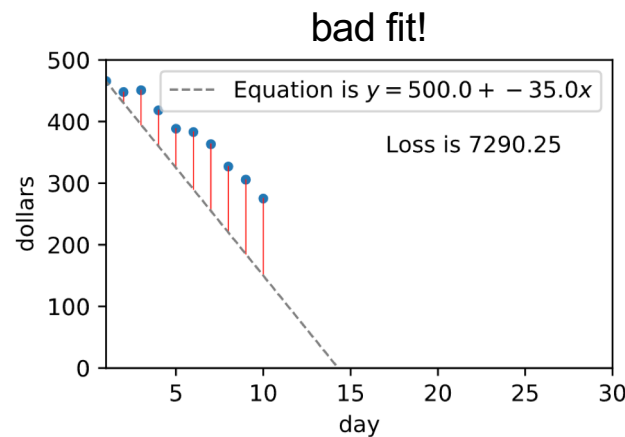
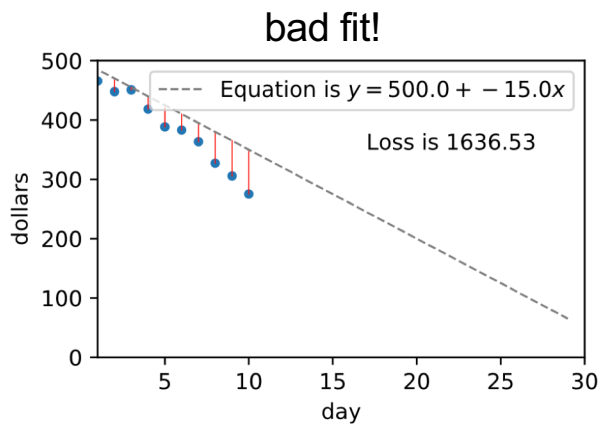


[Car computers that show number of miles remaining are solving the same problem]

See <https://github.com/parr/msds621/blob/master/notebooks/linear-models/sklearn-linear-models.ipynb>

Draw line, manually finding coefficients

- I knew to draw line to project into future, but how can we figure out slope of line? (y-intercept is clearly the starting amount)
- Measure *cost/loss* by computing average squared residual error then just wiggle slope until we find min loss



See <https://github.com/parr/msds621/blob/master/notebooks/linear-models/sklearn-linear-models.ipynb>

Review of linear regression notation

- Given (X, y) where X is $n \times p$ explanatory matrix and y is target or response vector, we seek coefficients that describe best hyper plane through (X, y) data
- Each row $x^{(i)}$ in X maps to $y^{(i)}$ and $x^{(i)} = [x_1, x_2, \dots, x_p]$

$$\hat{y} = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p = \beta_0 + \sum_{i=1}^p \beta_i x_i$$

- In vector notation, β is column vector $[\beta_1, \beta_2, \dots, \beta_p]$

$$\hat{y} = \beta_0 + \mathbf{x} \cdot \vec{\beta} = \beta_0 + \mathbf{x}\vec{\beta}$$

Augment with “1” trick

- Adding β_0 is messy so augment x with 1:

$$x' = [1, x_1, x_2, \dots, x_p]$$

then β is column vector

$$\beta = [\beta_0, \beta_1, \beta_2, \dots, \beta_p]$$

and we get the much simpler equation: $\hat{y} = \mathbf{x}' \vec{\beta}$

$1 \ x_1 \ \dots$	β_0
	β_1
	\vdots

Training/fitting linear model means finding optimal coefficients

- Finding optimal β amounts to finding vector β that minimize the mean-squared error, which is our loss function:

$$MSE(\beta) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

- Ignoring $1/n$ and substituting $\hat{y} = \mathbf{x}'\vec{\beta}$, we get:

$$\mathcal{L}(\beta) = \sum_{i=1}^n (y^{(i)} - (\mathbf{x}'^{(i)} \cdot \beta))^2 = (\mathbf{y} - \mathbf{X}'\beta) \cdot (\mathbf{y} - \mathbf{X}'\beta)$$

rows augmented

Solutions for finding linear model β

- Loss function is a (convex) quadratic with exact, symbolic solution and you've learned how to solve for coefficients directly
 - Well, if $n > p$ and no weak/nonpredictive columns (X full rank)
- Many regularized and logistic regression loss functions have no direct solutions, though
- You'll use an iterative solution (gradient descent) for all regression problems in your project

Training/testing of linear models in Python

- Boston dataset example into a notebook:

<https://github.com/parr/msds621/blob/master/notebooks/linear-models/sklearn-linear-models.ipynb>

```
boston = load_boston()
X, y = boston.data, boston.target

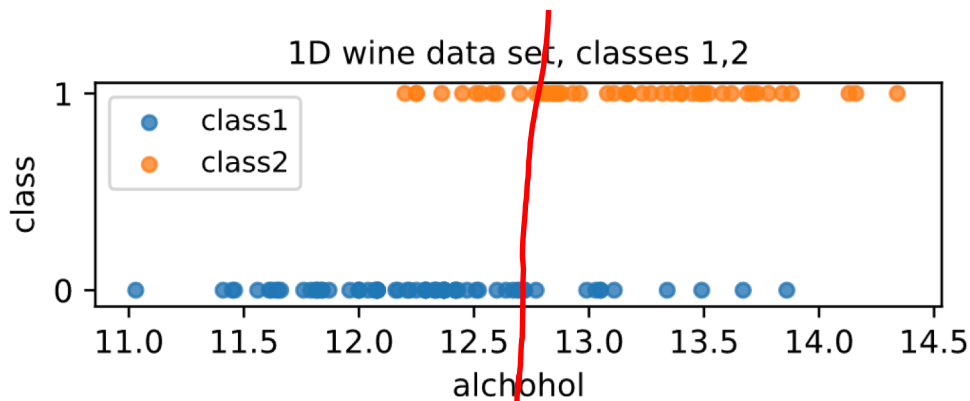
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.2)

lm = LinearRegression()          # OLS
lm.fit(X_train, y_train)
s = lm.score(X_test, y_test) #  $R^2 = 0.66$ 
```

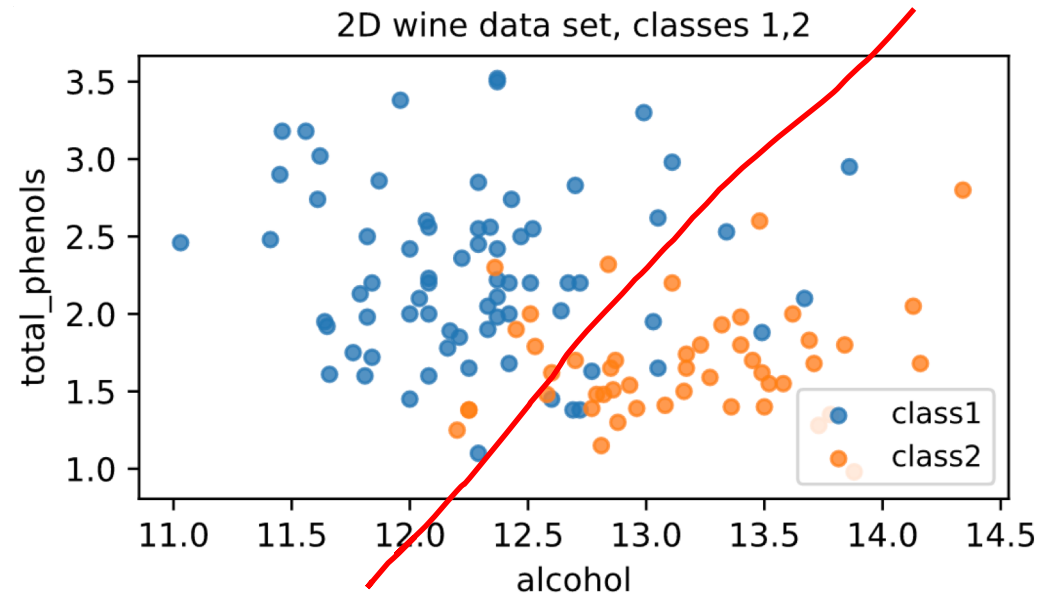
Logistic regression

Review of logistic regression

- For classification, response y is discrete int value like $\{0,1\}$
- Need separating hyperplane between points in different classes

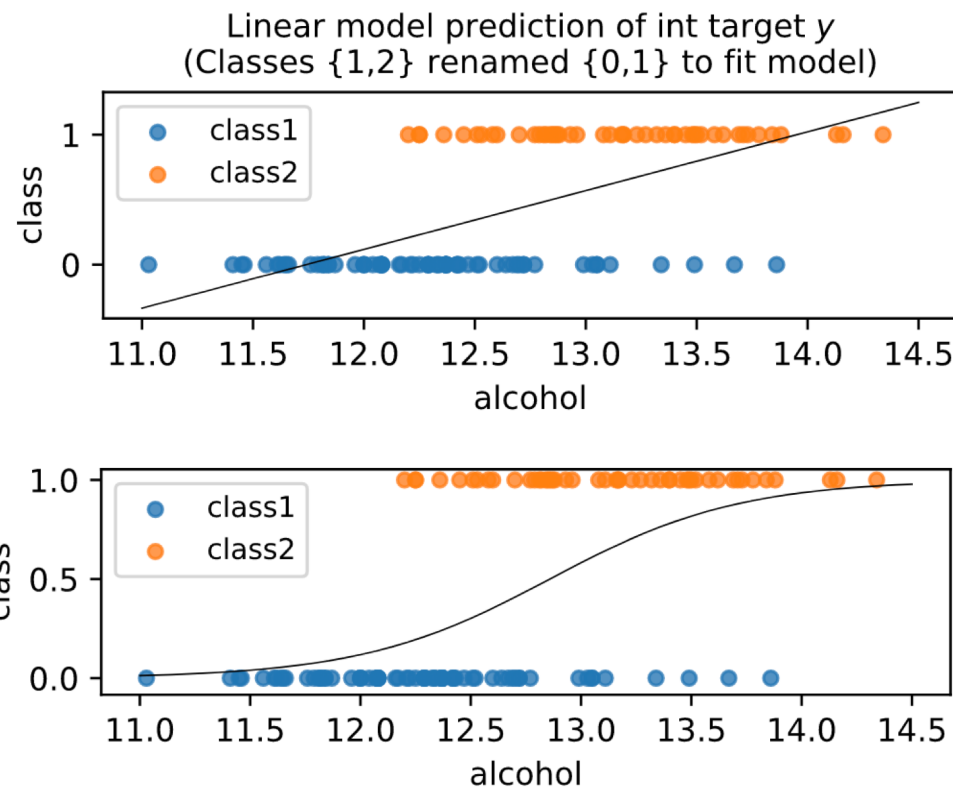


- Showing hard cutoffs but a smooth transition from class 1 to class 2 would be better



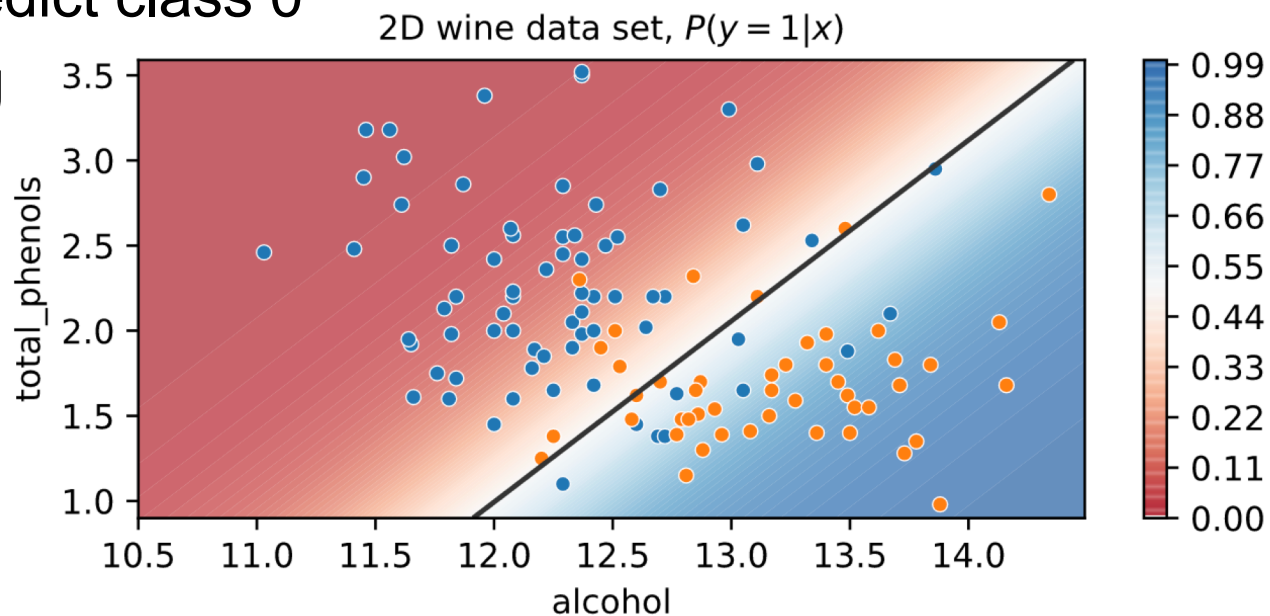
1D logistic regression

- Could use linear regression, but line would exceed $[0,1]$ range
- Sigmoid is a much better transition from class 0 to class 1 and gives probability of class 1: $P(y = 1|x)$
- Training sends output of linear model into sigmoid and maximizes max likelihood to find coefficients



2D wine data set example, 2 features

- Logistic regression yields $P(y = 1|\mathbf{x})$
- Classifier built on top of logistic prediction; $P(y = 1|\mathbf{x}) \geq 0.5$ predict class 1 else predict class 0
- Black line is separating plane, but output of model is smooth transition, not hard threshold, from 0 to 1



UNIVERSITY OF SAN FRANCISCO

See <https://github.com/parr/msds621/blob/master/notebooks/linear-models/sklearn-linear-models.ipynb>

Logistic regression notation

- Sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}$$

- Substituting vectorized linear eqn into sigmoid:

$$p(\mathbf{x}') = \sigma(\mathbf{x}'\beta) = \frac{1}{1 + e^{-\mathbf{x}'\beta}}$$

- Using odds = $p/(1-p)$, subst in $p(\mathbf{x}')$, simplify, take log; we get:

$$\log(odds) = \mathbf{x}'\beta$$

Solving for logistic β

- Same idea: define loss function (negative of max likelihood in this case) and solve for β that gives min loss value
- Loss function has nonlinearity in it now though:

$$\mathcal{L}(\beta) = \sum_{i=1}^n \left\{ y^{(i)} \mathbf{x}'^{(i)} \beta - \log(1 + e^{\mathbf{x}' \beta}) \right\}$$

- Logistic regression requires an iterative solution because there is no direct, symbolic solution, unlike linear regression

Training/testing of logistic regression models in Python

- Wine dataset example from into a notebook:

<https://github.com/parr/msds621/blob/master/notebooks/linear-models/classifier-regularization.ipynb>

```
wine = load_wine()
df_wine = pd.DataFrame(data=wine.data,
                        columns=wine.feature_names)
df_wine['y'] = wine.target
df_wine = df_wine[df_wine['y']<2] # do 2-class problem {0,1}
X, y= df_wine.drop('y', axis=1), df_wine['y']
normalize(X)
lm = LogisticRegression(solver='lbfgs', max_iter=1000)
lm.fit(X.values, y) # uses regularization by default
lm.score(X.values, y)
```