Linear Regression

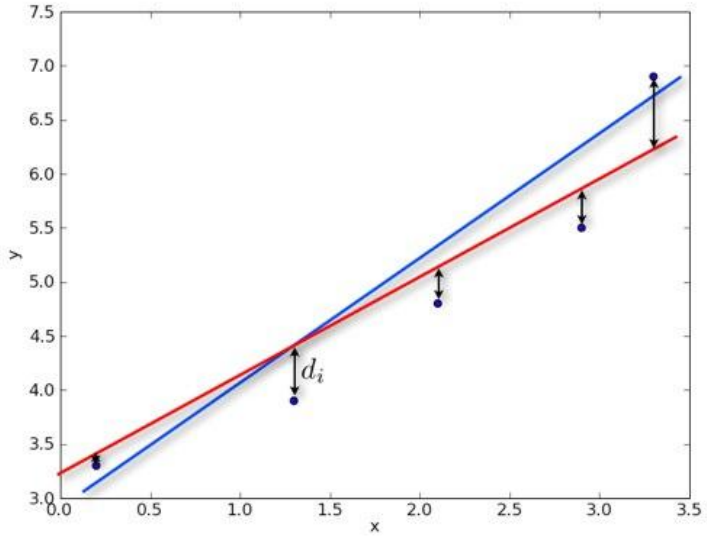
Team AB

Fall 2020
EECS 189/289A

Berkeley
UNIVERSITY OF CALIFORNIA

Simple Linear Regression

Deriving Slope and Intercept

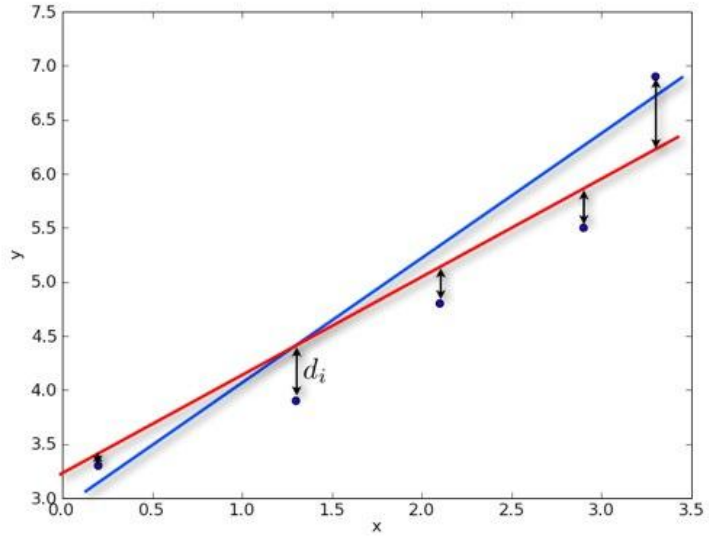


We want to figure out which line works better to estimate the model.

Can you guess which one might work better?

How might you quantitatively figure out which fits better?

Deriving Slope and Intercept



We want to minimize the y-error shown here as a d_i value. Because this error can occur above or below the true y values, we need to minimize the squared error.

Through get partial derivatives and setting them to zero, we arrive at the equations on the next page (see note for more detailed derivation).

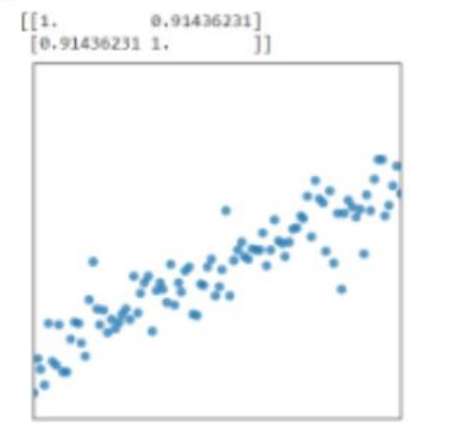
Correlation Coefficient

In the coding assignment, we will go through a visualization showing that these equations do indeed minimize the loss or error from possible linear fits. The note and coding assignment also cover a more data science approach using the correlation coefficient r in arriving to similar linear models.

$$b = \frac{\sum y_i - m \sum x_i}{n}$$

$$m = \frac{\sum x_i y_i - \sum y_i \sum x_i / n}{\sum x_i^2 - (\sum x_i)^2}$$

What is the correlation coefficie
r? The higher the r value (e.g. t
the right 0.914), the more the x
y values fit a linear model.



Built-in Libraries

Though knowing how to derive the equations is important, Python has in-built libraries like sklearn that already contain these models. All you need is a dataset to get some results! We will cover sklearn linear model in the coding assignment.

Multiple Linear Regression

Ordinary Least Squares

Problem: Given labels y and inputs x, we suppose this data has relationship:

$$y_i \approx \hat{y}i = h_w(x_i) = x_i^T w$$

Goal: Want to minimize the error between the predicted label and actual label

$$L(w) = \sum_{i=1}^{n} (x_i^T w - y_i)^2 = \min_w ||\mathbf{Xw} - \mathbf{y}||_2^2$$

Ordinary Least Squares

Recall the following important result: If $L : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable, then any local optimum satisfies $\nabla L(w) = 0$. In the OLS case,

$$\begin{aligned} L(w) &= ||Xw - y||_2^2 \\ &= (Xwy)^T(Xwy) \\ &= (Xw)^T Xw(Xw)^T yy^T Xw + y^T y \\ &= w^T X^T Xw - 2w^T X^T y + y^T y \end{aligned}$$

Ordinary Least Squares

The gradient of L is easily seen to be

$$\begin{aligned}\nabla L(w) &= \nabla_w(w^T X^T X w - 2w^T X^T y + y^T y) \\ &= \nabla_w(w^T X^T X w) - 2\nabla_w(w^T X^T y) + \nabla_w(y^T y) \\ &= 2X^T X w - 2X^T y\end{aligned}$$

where in the last line we have used the symmetry of $X^T X$ to simplify $X^T X + (X^T X)^T = 2X^T X$.

Setting the gradient to $\mathbf{0}$, we conclude that any optimum w OLS satisfies:

$$X^T X w_{ols}^* = X^T y$$

If X is full rank, then $X^T X$ is as well (assuming n d), so we can solve for a unique solution:

$$w_{ols}^* = (X^T X)^1 X^T y$$

Categorical Data and One-Hot Encoding

Data Types

- Numerical: numerical data
  - Ex: movie ticket prices, number of people in household
- Categorical: Non numerical, unordered data
  - Ex: blood type, species, gender
- Ordinal: categorical but ordered data
  - Ex: ratings, sequences

Most machine learning processes cannot handle raw categorical or ordinal data. Thus, we need to process it to usable features.

One-Hot Encoding / Dummy Encoding

We can transform categorical data into numeric data by using one hot encoding (aka dummy encoding).

This means that if we have a categorical variable, say “favorite drinks”, and labels for this variable as ‘soda’, ‘coffee’, ‘juice’, ‘water’, ‘other’. We take every unique label of the data points for favorite drinks and transform them into 1/0 indicator columns.

Example

Favorite Drink
soda
coffee
...
water
coffee

soda	coffee	juice	water	other
1	0	0	0	0
0	1	0	0	0
...	...	...	...	...
0	0	0	1	0
0	1	0	0	0

Methods

While there are a lot of different ways to go about one-hot encoding, there are two methods that require minimal coding and highly efficient.

1. Pandas
2. SKLearn

To demonstrate, we will be using the fruit_info table.

	fruit	price	season	rating
0	apple	5	winter	4
1	orange	6	winter	2
2	banana	2	summer	3
3	raspberry	3	summer	5

You will be able to play around with this table in homework!

Methods: Pandas.get_dummies

```
# returns the one hot encoded table of only column 'fruit'
pd.get_dummies(fruit_info['fruit'])
```

	apple	banana	orange	raspberry
0	1	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	0	1

```
[ ] # returns the full dataframe with one hot encoded columns
pd.get_dummies(fruit_info)
```

	price	rating	fruit_apple	fruit_banana	fruit_orange	fruit_raspberry	season_summer
0	5	4	1	0	0	0	0
1	6	2	0	0	1	0	0
2	2	3	0	1	0	0	1
3	3	5	0	0	0	1	1

Methods: sklearn.preprocessing.OneHotEncoder

```
[ ]  # A simple example : courtesy of scikit-learn.org

     # create and fit the encoder to the dataframe
     enc = OneHotEncoder(handle_unknown = 'ignore')
     enc.fit(fruit_info)
```

```
OneHotEncoder(categories='auto', drop=None, dtype=<class 'numpy.float64'>,
              handle_unknown='ignore', sparse=True)
```

```
[ ]
     enc.categories_
```

```
[array(['apple', 'banana', 'orange', 'raspberry'], dtype=object),
 array([2, 3, 5, 6]),
 array(['summer', 'winter'], dtype=object),
 array([2, 3, 4, 5])]
```

```
[ ]  # get vectorized one-hots from inputs
     enc.transform([['apple', 2, 'winter', 4], ['raspberry', '5', 'spring', '2']]).toarray()
     # notice how since spring was not trained on, it does not get encoded
```

```
array([[1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

Features

Normally, if we know the underlying model is linear, we can solve the least squares problem

$$\min_x ||\mathbf{Xw} - \mathbf{y}||_2^2$$

where $\mathbf{X} = \begin{bmatrix} \mathbf{x_1}^T \\ \mathbf{x_2}^T \\ \cdots \\ \mathbf{x_n}^T \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}$.

However, if we have knowledge that the underlying model is nonlinear, we can featurize our data by creating a function $\phi : \mathbb{R}^l \to \mathbb{R}^d$, called a feature map, which maps each data point $x \in \mathbb{R}^l$ to a featurized data point $\phi(x) \in \mathbb{R}^d$. Then, we will solve the least squares problem,

$$\min_x ||\Phi \mathbf{w} - \mathbf{y}||_2^2$$

where $\Phi = \begin{bmatrix} \phi(\mathbf{x_1})^T \\ \phi(\mathbf{x_2})^T \\ \cdots \\ \phi(\mathbf{x_n})^T \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix}$.

Example: Fitting an Ellipse

We know that ellipses abide by the equation: $ax^2 + bxy + cy^2 + dx + ey = 1$

So, we can augment our (x, y) datapoints as such:

$$\begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 \\ \dots & \dots & \dots & \dots & \dots \\ x_n^2 & \dots & \dots & \dots & y_n \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} 1 \\ \dots \\ \dots \\ \dots \\ 1 \end{bmatrix}$$

Polynomial Featurization

Dimension \ Degree	0	1	2	3	...
1 (univariate)	1	x	x^2	x^3	...
2 (bivariate)	1	x_1, x_2	x_1^2, x_2^2, x_1x_2	$x_1^3, x_2^3, x_1^2x_2, x_1x_2^2$	...
⋮	⋮	⋮	⋮	⋮	⋱

- Polynomials are **universal approximators** (i.e. they can approximate the sin function, the exponential function, etc. via Taylor Expansion)
    - Solves drawback to featurization where the underlying structure of the nonlinearity has to be explicitly specified in the feature map

Chart taken from Note 3 of Nasiriany, S., Thomas, G., Wang, W., Yang, A., Listgarten, J., & Sahai, A. (2019) A Comprehensive Guide to Machine Learning.

Affine Featurization

It turns that $y = cx + d$ can be modeled with the featurization $\phi : x \to (x, 1)$. This is because the equation can be written as $y = \begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}$.

- An “affine featurization” also turns out to just be a polynomial featurization, but of degree 0.

Linear Regression for Classification

Binary Classification

Binary classification case is when K = 2 and the label is either positive or negative, represented by +1 and -1. We can apply linear regression with the following procedure:

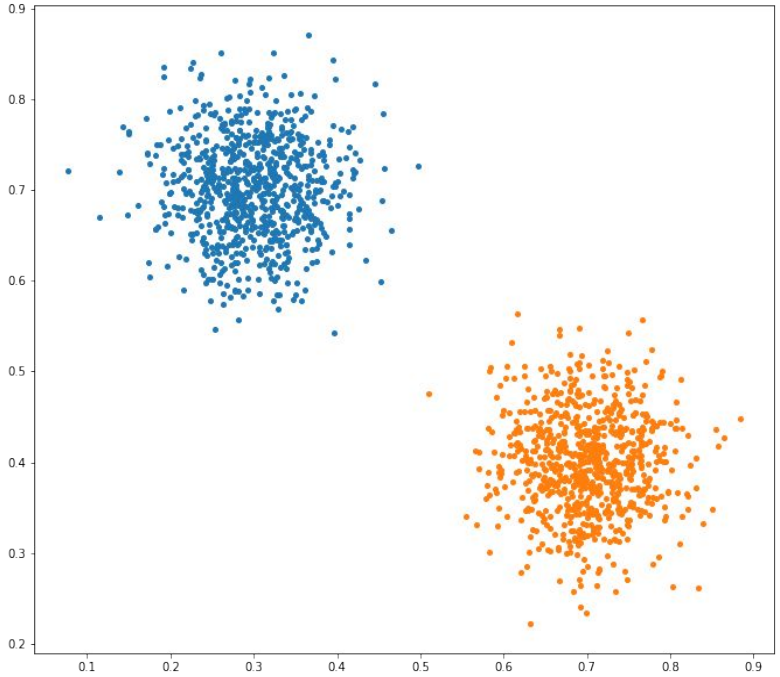
1. Estimate the weight vector by solving the already familiar least squares problem:

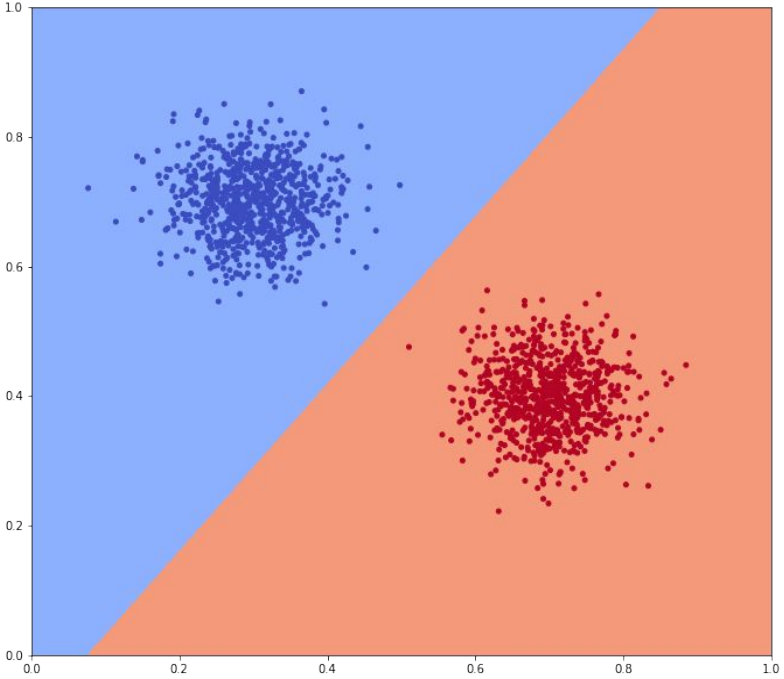
$$\mathbf{w} = \arg\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2$$

2. Given a data point $\mathbf{x}$, compute $\mathbf{w}^T\mathbf{x}$.

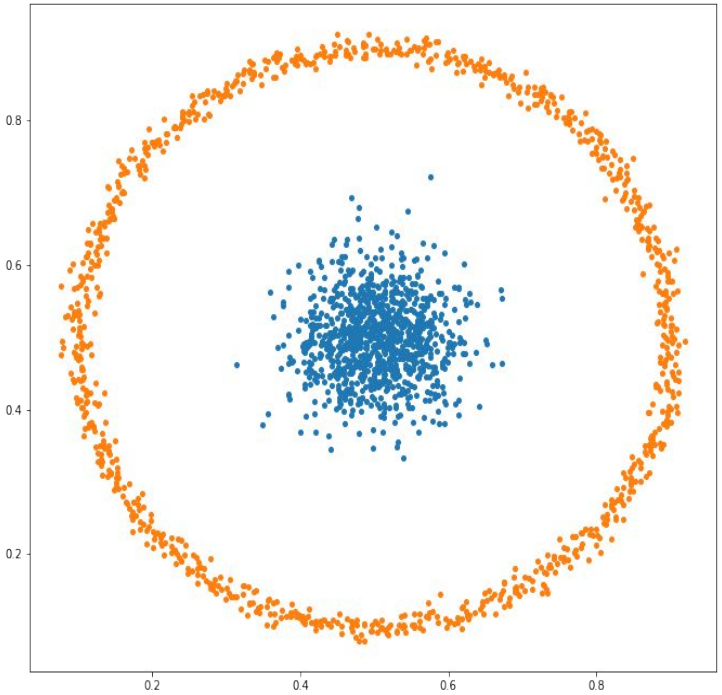
3. Classify $\mathbf{x}$ as $\mathbf{sign}(\mathbf{w}^T\mathbf{x})$.

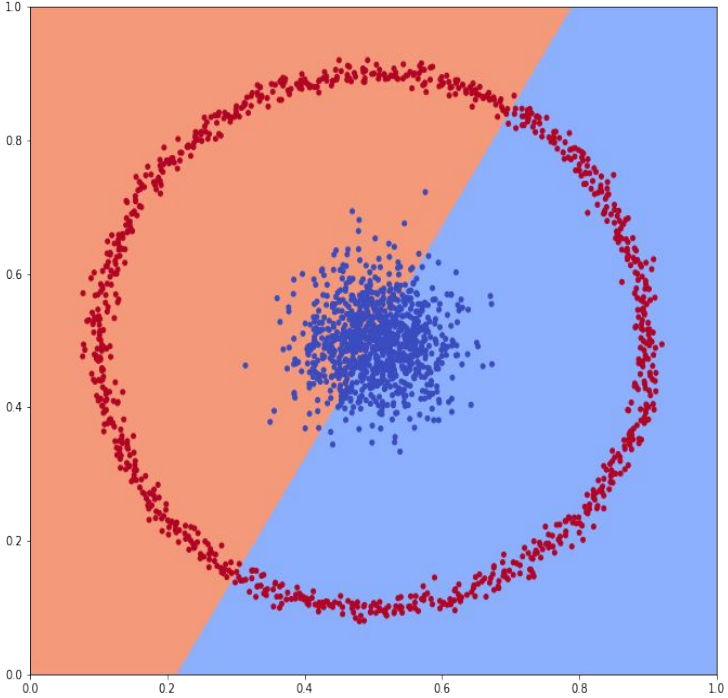
Linearly Separable Data





Non-linearly Separable Data





Multinomial Classification

One possible way to generalize this method to K $>$ 2 classes is to employ the One-vs-Rest approach:

1. For each class k in $\{1, 2, ..., K\}$
   - Estimate the weight vector by solving the already familiar least squares problem:

$$\mathbf{w_k} = \arg\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y_k}||_2^2$$

   - Given a data point $\mathbf{x}$, compute $\hat{y_k} = \mathbf{w}_k^T \mathbf{x}$ for each class k in $\{1, 2, ..., K\}$.
2. Classify $\mathbf{x}$ as

$$\text{class}(\mathbf{x}) = \arg\max_{k \in \{1,2,...,K\}} \hat{y}_k$$

Multinomial Classification