Linear Regression Notes

EEAS 189/289A Project T Final

Team AB

Anika Rede

Jiyoo Jeong

Nikolay Velkov

Patrick Yin

Qiaoan Yang

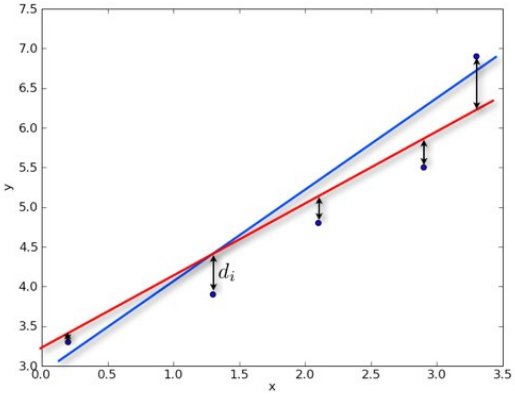
1 Simple Linear Regression

In machine learning, we want to predict relationships in the data. The simplest version of this can be seen in linear regression where a function fits the form $y = f(x)$. The true relationship (i.e. values of slope and intercept) are unknown to us, so we try to derive those from the data whereby we create a relationship from the data of $y = h(x)$ for a hypothesis state. We will explore a couple perspectives on linear regression, starting from deriving the slope and intercept to understanding what correlation is and how it plays in from a data science perspective.

1.1 Derive Slope and Intercept

Given the graphic below, how do we decide which line works best? We want to compare the errors of the red line versus the blue line and use the line with the least amount of error. But instead of just summing over the errors, we sum over the errors squared. Why is this the case?

This accounts for when error occurs above or below the line so the sign of error (positive or negative) does not factor into when we try to minimize error.



Thus we get the following equation:

$$S = \sum_{i=0}^{n} d_i^2$$

Now we want to get the best linear fit using the following equation: $y = mx + b$

Now we calculate the error and error squared as follows:

$$d_i = y_i - y = y_i - mx_i - b$$
$$d_i^2 = (y_i - mx_i - b)^2$$

Now to find the minimum of the error, we want to take the partial derivative of the error squared with respect to the slope and intercept respectively.

Partial derivative with respect to m:

$$\frac{\partial S}{\partial m} = \frac{\partial}{\partial m} \sum (y_i - mx_i - b)^2 \tag{1}$$

$$\frac{\partial S}{\partial m} = \sum [2(y_i - mx_i - b)(-x_i)] \tag{2}$$

$$\sum y_i x_i - \sum m x_i^2 - \sum b x_i = 0 \tag{3}$$

$$\sum y_i x_i - m \sum x_i^2 - b \sum x_i = 0 \tag{4}$$

$$\frac{\partial S}{\partial m} = \sum [2(y_i - mx_i - b)(-x_i)] \tag{5}$$

Partial derivative with respect to b:

$$\frac{\partial S}{\partial b} = \frac{\partial}{\partial b} \sum (y_i - mx_i - b)^2 \tag{6}$$

$$\frac{\partial S}{\partial b} = \sum [2(y_i - mx_i - b)(-1)] \tag{7}$$

$$\sum y_i - m \sum m x_i - b \sum 1 = 0 \tag{8}$$

$$\sum y_i - m \sum m x_i - bn = 0 \tag{9}$$

Optimal slope and intercept equations:

$$b = \frac{\sum y_i - m \sum x_i}{n} \tag{10}$$

$$m = \frac{\sum x_i y_i - \sum y_i \sum x_i / n}{\sum x_i^2 - (\sum x_i)^2} \tag{11}$$

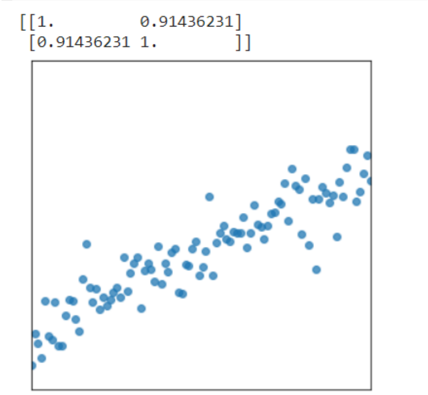
Thus we arrive at the equations for the slope and intercept when minimizing error.

In the coding assignment, we introduce the term "bias" which is the difference between the expected value of a model and the true value being estimated.

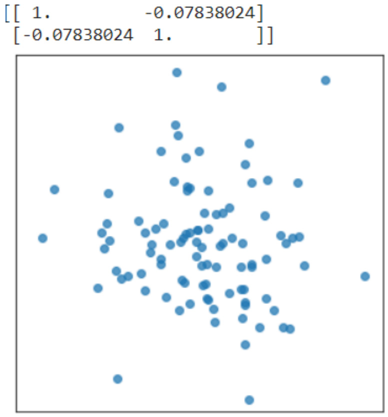
1.2 Data Science Perspective

When calculating these linear models, many sources often cite a value called r^2. What does this represent? We will discuss that here.

In data science, the value r^2 represents the squared value of the correlation coefficient which visually shows correlation between x and y data. For example, in the data below the first would output a large r value but the second would not. This coefficient is based on standard units with values ranging from -1 to 1 where $r = 1$ would have a scatter perfectly straight line sloped up and $r = -1$ scattered perfectly sloping down. In between, $r = 0$ shows no linear association, i.e. uncorrelated x and y data.



```
[[1.         0.91436231]
 [0.91436231 1.        ]]
```



```
[[ 1.         -0.07838024]
 [-0.07838024  1.        ]]
```

In the coding assignment, we discuss what this means for various types of data sets to indicate where linear regression might work better and where it might not.

Finally, we see that we can apply linear regression even to exponential curves using the ln function to the y values. Given $y = e^{mx+b}$, we need to transform the data appropriately to fit to a linear model: $ln(y) = mx + b$. Thus, when applicable we can apply transformations to the data and fit to the linear regression model.

2 Multiple Linear Regression

Now we want to see if we have several explanatory variables to predict the outcome of a response variable, which we usually called **Multiple Linear Regression(MLR)** In essence, Multiple regression is the extension of ordinary least-squares (OLS) regression that involves more than one explanatory variable. Therefore, let talk about the Ordinary Least Squares first!

2.1 Ordinary Least Squares

Ordinary least squares (OLS) is one of the simplest regression problems, but it is well-understood and practically useful. It is a **linear regression** problem, which means that we take h_w to be of the form $h_w(x) = x^T w$. We want

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

for each $i = 1, ..., n$. This set of equations can be written in matrix form as

$$y \approx XwR$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_1}^T \\ \mathbf{x_2}^T \\ \cdots \\ \mathbf{x_n}^T \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{bmatrix} \text{ and } \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_d \end{bmatrix}.$$

In words, the matrix $X \in \mathbb{R}^{n \times d}$ has the input data point x_i as its ith row. This matrix is sometimes called the **design matrix**. Usually $n \geq d$, meaning that there are more data points than measurements.

There will in general be no exact solution to the equation **y = Xw** (even if the data were perfect, consider how many equations and variables there are), but we can find an approximate solution by minimizing the sum (or equivalently, the mean) of the squared errors:

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

Now that we have formulated an optimization problem, we want to go about solving it. We will see that the particular structure of OLS allows us to compute a closed-form expression for a globally optimal solution, which we denote w_{ols}^*.

Calculus is the primary mathematical workhorse for studying the optimization of differentiable functions. Recall the following important result: If $L : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable, then any local optimum satisfies $\nabla L(w) = 0$. Intuitively, this is just a multidimensional generalization of how $\frac{df}{dx}(x_0) = 0$ implies x_0 is at at local minimum for function f. In the OLS case,

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

Using the following results from matrix calculus:

$$\nabla_x(a^T x) = a$$

$$\nabla_x(x^T Ax) = (A + A^T)x$$

For now, you can take the following results as granted. The gradient of L is easily seen to be

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

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 **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:

$$w^*_{ols} = (X^T X)^{-1} X^T y$$

Note: Although we write $(X^T X)^{-1}$, in practice one would not actually compute the inverse; it is more numerically stable to solve the linear system of equations above (e.g. with Gaussian elimination).

Below, we will rigorous prove to you why this OLS solution actually reaches the global minimum of the loss function. Note that you are not expected to know the terms convexity or positive semi-definite. So it is ok if you do not fully understand the math justifying why x reaches a global minimum. Instead think about intuitively how our loss function is really just a multidimensional generalization of a squared error. And note that square functions (i.e. $f(x) = x^2$), when plotted out, seem to have a global minimum (in fact, $\frac{d^2 f}{dx^2} = 2$, so this makes sense!). With this in mind, proceed with courage!

In this derivation we have used the condition $\nabla L(w^*) = 0$, which is a *necessary* but not *sufficient* condition for optimality. We found a critical point, but in general such a point could be a local minimum, a local maximum, or a saddle point. Fortunately, in this case the objective function is **convex**, which implies that any critical point is indeed a global minimum. To show that L is **convex**, it suffices to compute the **Hessian** of L, which in this case is

$$\nabla^2 L(w) = 2X^T X$$

and show that this is positive semi-definite:

$$\forall w, w^T(2X^T X)w = 2(Xw)^T Xw = 2||Xw||_2^2 \geq 0$$

The definition of convexity is described below for reference. Remember that this material is out of scope!

Definition (convex functions): A function f : C $\to$ $\mathbb{R}$ defined on a **convex set** C $\subseteq$ $\mathbb{R}^n$ is called **convex (or convex over C)** if for any $x, y \in C, \lambda \in [0,1]$,

$$f(\lambda + (1-\lambda)y) \leq \lambda(x) + (1-\lambda)f(y)$$

.

The fundamental inequality is illustrated in Figure belong.

The left quadratic function is convex ($f(x,y) = x^2 + y^2$), while the middle and right functions are nonconvex.

Theorem (convexity and strict convexity of quadratic functions with positive semidefinite matrices). Let $f : \mathbb{R}^n \to \mathbb{R}$ be the quadratic function given by $f(x) = x^T Ax + 2b^T x + c$, where $A \in \mathbb{R}^{n \times n}$ is symmetric, $b \in \mathbb{R}^{\ltimes}$,and $c \in \mathbb{R}$.Then f is convex (strictly convex) if and only if $A \geq 0 (A > 0)$.

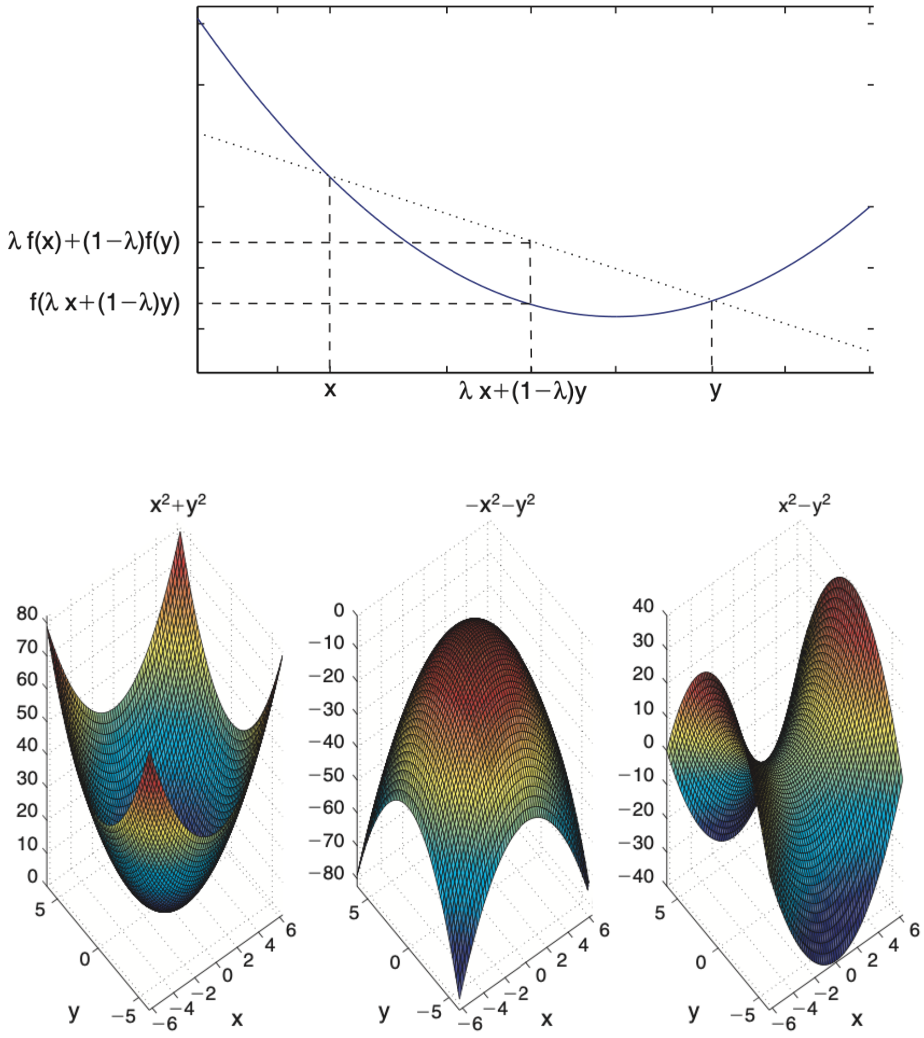


Figure 1: Figure: Image provided courtesy of Amir Beck. [11]

If you didn't know any information about the optimization, you could take EECS 127 or IND ENG 160 before this class!

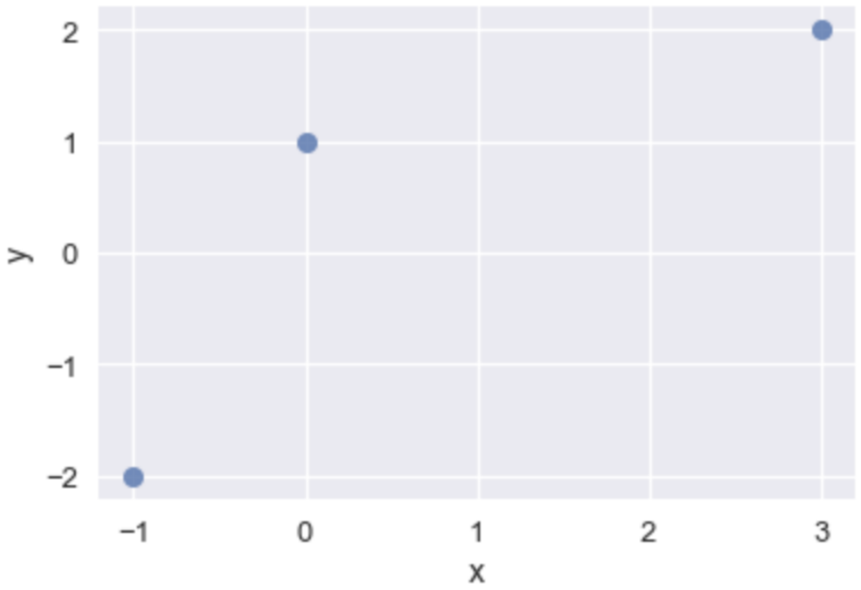


Figure 2: Figure: Image provided courtesy of Joseph E. Gonzalez. [2]

2.2 Geometric Perspective

Now, let's develop an intuition for why it matters that $\hat{y}$ is restricted to the linear combinations of the columns of **X**. Although the span of any set of vectors includes an infinite number of linear combinations, infinite does not mean any—the linear combinations are restricted by the basis vectors.

As a reminder, here is our loss function and scatter plot:

$$L(\theta, x, y) = ||y - X\theta||^2$$

By inspecting our scatter plot, we see that no line can perfectly fit our points, so we cannot achieve 0 loss. Thus, we know that **y** is not in the plane spanned by **X** and 1, represented as a parallelogram below.

Since our loss is distance-based, we can see that to minimize$L(\theta, x, y) = ||y - X\theta||^2$, we want $X\theta$to be as close to **y** as possible.

Mathematically, we are looking for the projection of **y** onto the vector space spanned by the columns of **X**, as the projection of any vector is the closest point in ***Span(X)*** to that vector. Thus, choosing θ such that $\hat{y}$ = $X\theta$ = $proj_{span(X)}y$ is the best solution.

To see why, consider other points on the vector space, in purple.

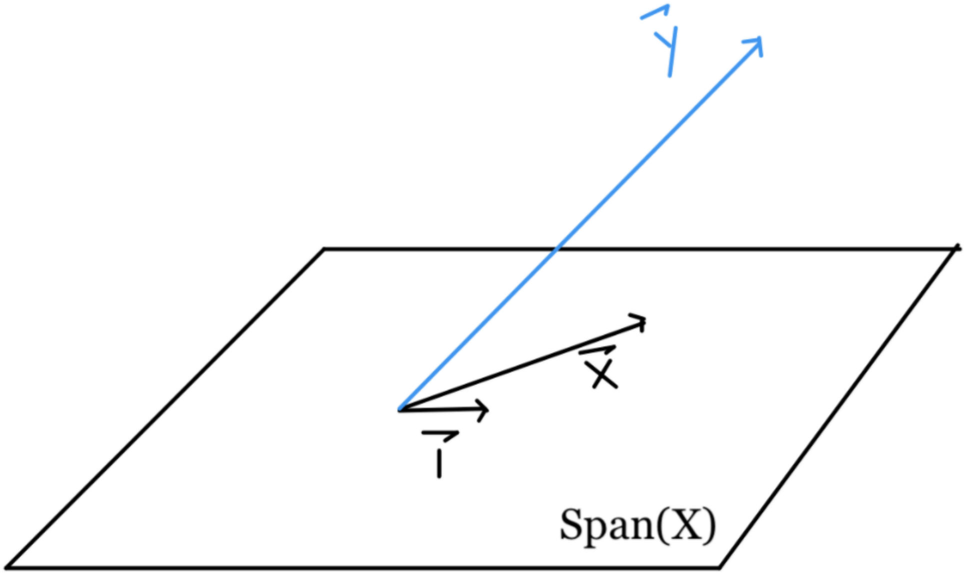


Figure 3: Figure: Image provided courtesy of Joseph E. Gonzalez. [2]

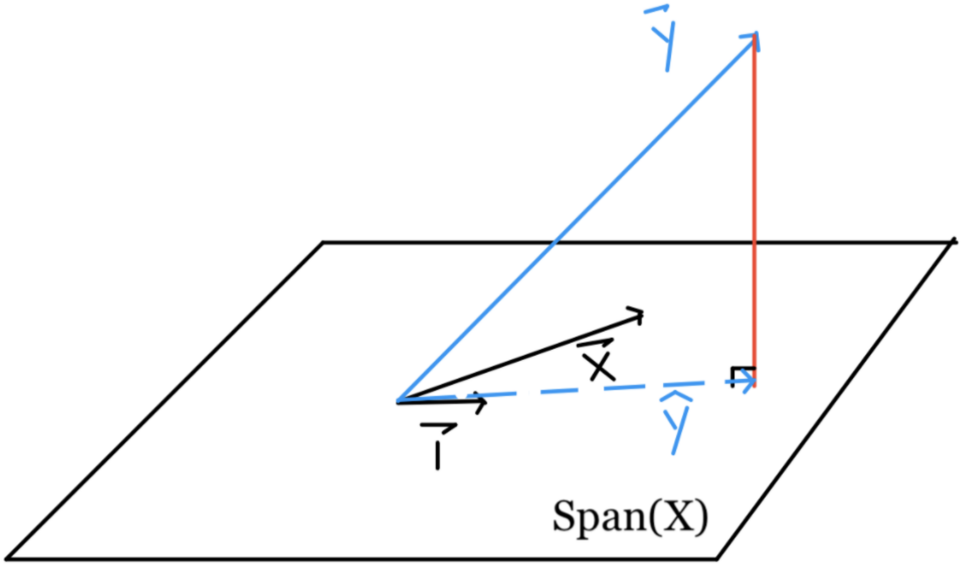


Figure 4: Figure: Image provided courtesy of Joseph E. Gonzalez. [2]

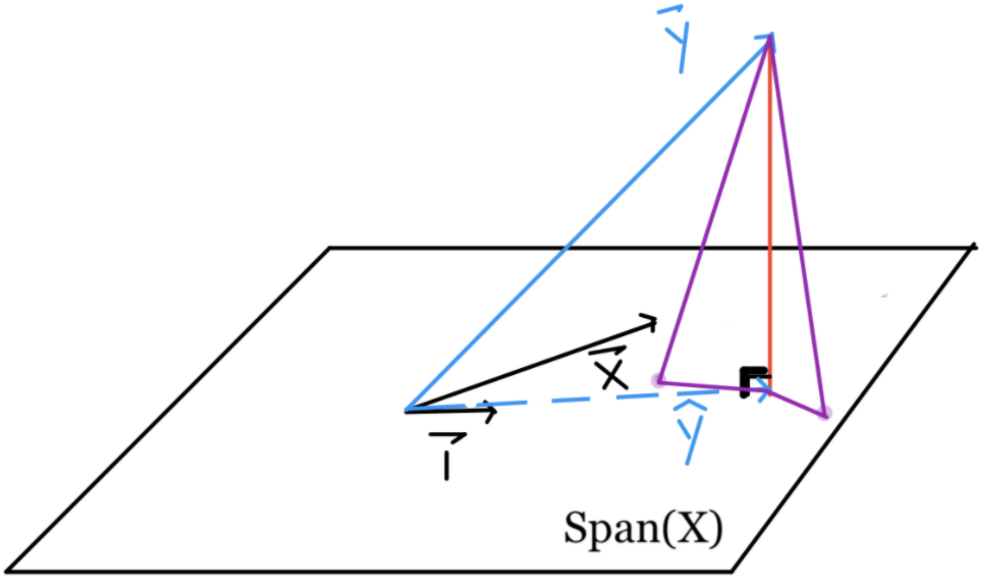


Figure 5: Figure: Image provided courtesy of Joseph E. Gonzalez. [2]

By the Pythagorean Theorem, any other point on the plane is farther from y than $\hat{y}$ is. The length of the perpendicular corresponding to $\hat{y}$ represents the least squared error.

3 Categorical Data and One-Hot Encoding

3.1 Categorical Data

Categorical data are variables that are classified by labels rather than strictly numerical values. Often these variables are called nominal variables. For example, a "pet" variable can have values like "dog", "cat", or "iguana". Similar to categorizing your classes by subject, or the clothes you wear by occasion, humans have a natural tendency to associate data with nominal variables.

Other forms of data include numerical, as we have seen before, and ordinal data, which are categorical data which have rankings or order. Ex: star ratings on yelp.

So how do computers see and use categorical data?

Some algorithms can work with categorical data directly, for example, decision trees (which you will learn about later topics in ML). However, most algorithms won't be able to perform efficient and accurate predictions with categorical data. Our job now as avid machine learning enthusiasts is to translate categorical data into a way the majority of machine learning algorithms can efficiently provide modelling and analysis, which brings us to our module topic: One Hot Encoding.

3.2 One-Hot Encoding

The term "One Hot" comes from how categories are indicated in this form, with the row record's are lit up as a '1' if it falls into the category of question, and '0' otherwise. In the academic community, this is also called 'dummy encoding'.

The vector form of one-hot encoding can be represented with indicators. As we have seen in previous courses, indicators are functions that return a 1 or 0 based on conditions.

To one-hot encode, we transform a single column of classes in the form of a categorical variable A, with values $k_1, k_2, k_3...k_n$, into n columns of indicator values. Each k_i column has the indicator:

$$1_{k_i,a_i} = \begin{cases} 1, & \text{if } a_i = k_i \\ 0, & \text{otherwise} \end{cases}$$

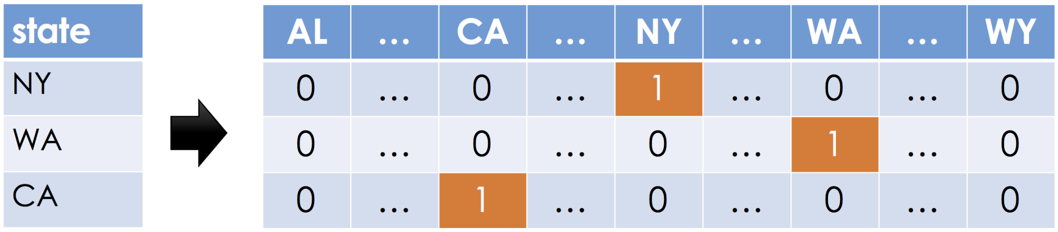
Here is a visual representation of what one-hot encoding looks like:

$$A_{variable} = \begin{bmatrix} \mathbf{a_1} \\ \mathbf{a_2} \\ ... \\ \mathbf{a_m} \end{bmatrix}$$

$Set(A_{variable}) = K = \{k_1, k_2, k_3, ..., k_n\}$, a set of unique labels, k_i, from $A_{variable}$

$$A_{variable} \rightarrow \begin{bmatrix} 1_{k_1,a_0} & 1_{k_2,a_0} & 1_{k_3,a_0} & \dots & 1_{k_{n-1},a_0} & 1_{k_n,a_0} \\ 1_{k_1,a_1} & 1_{k_2,a_1} & 1_{k_3,a_1} & \dots & 1_{k_{n-1},a_1} & 1_{k_n,a_1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1_{k_1,a_m} & 1_{k_2,a_m} & 1_{k_3,a_m} & \dots & 1_{k_{n-1},a_m} & 1_{k_n,a_m} \end{bmatrix}$$

In practice, the tables we encode will look like this:



state
NY
WA
CA

AL	…	CA	…	NY	…	WA	…	WY
0	…	0	…	1	…	0	…	0
0	…	0	…	0	…	1	…	0
0	…	1	…	0	…	0	…	0

Figure 6: Figure: Image provided courtesy of Joseph E. Gonzalez. [5]

There are various methods for one-hot encoding, manually, using pandas, and SK-Learn libraries to efficiently and effectively create vectorized categories. These procedures will be covered thoroughly in upcoming assignments.

4 Features

We now know that solving the least squares problem can be a great fit for linear problems where $y = cx$. However, what if the underlying model was nonlinear like $y = cx^2$ or affine like $y = cx + d$? Then our underlying assumptions about the model being linear when doing least squares falls apart.

But it turns out, least squares can still be used for nonlinear models by the augmenting the data with new features. Rigorously, assume we have n datapoints $\{\mathbf{x_i}, y_i\}_{i=1,...,n}$ where $x_i \in \mathbb{R}^l$. Normally, if we know the underlying model is linear, we can solve the least squares problem

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

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

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

By introducing nonlinearities into our feature map, we are essentially able to using linear least-squares to model nonlinear relationships. In a sense, we are really just lifting our data into a higher-dimensional space which some underlying structure that we specify and then doing linear regression on this higher-dimensional space. In doing so, from the perspective of the original space, we have found a clever way to do nonlinear modelling.

4.1 Fitting an Ellipse

In the iPython notebook, we will be predicting the orbit of certain planets with least squares. We will being using the knowledge of Carl Friedrich Gauss, a great mathematician and physicist, who used least squares to predict where certain planets would be in their orbit. Using Kepler's laws of planetary motion, Gauss knew that the shape of the orbit of planets followed the equation:

$$ax^2 + bxy + cy^2 + dx + ey = f$$

where x and y were the coordinate points of the planet in orbit. Since there is one free variable in this equation, so Gauss divided the whole equation by f to get:

$$ax^2 + bxy + cy^2 + dx + ey = 1$$

With this underlying model known, we can create our feature map and do least-squares! Since we know that the underlying structure of the model abides by the equation above, we can now treat x and y both as datapoints and lift these datapoints to a higher dimensional space as such:

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

For this specific problem, note that the regression problem has slightly changed here in that our data is now (x, y) and our ground truth output is just the number 1. To be clear, feature map here is $\phi : (x, y) \to (x_1^2, x_1y_1, y_1^2, x_1, y_1)$. With this new augmented data matrix, we can do least squares to solve the problem, which you will do in the iPython notebook.

4.2 Polynomial Featurization

The ellipse example motivates a larger class of features called polynomial features. Polynomial features are specified by their degree and dimension. A 1-dimensional nth degree polynomial feature map is $\phi : x \to (1, x, x^2, ..., x^n)$. Our ellipse example is similar to the 2-dimensional nth degree polynomial feature map, which is $\phi : (x, y) \to (1, x, y, x^2, xy, y^2, ..., x^n, x^{n-1}y, ..., xy^{n-1}, y^n)$. The dth dimensional nth degree polynomial feature space can be generalized by adding more variables.

Polynomial featurization solves a really important problem in featurized regression. The drawback to featurized regression is that the underlying structure of the nonlinearity has to be explicitly specified in the feature map. However, in many cases, the nonlinearity in the model is unknown. Polynomial featurization solves this problem because, through Taylor expansion, we know that polynomials are a universial approximator. In other words, any nonlinearity, such as the sin function or the exponential function, can be approximated by using a high enough degree polynomial.

Let's go back to our simple example of $y = cx^2$. Now that we know polynomial featurization, we know that we can model this equation by using a polynomial feature map. But how about $y = cx + d$?

4.3 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 $y = \begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}$. This d term is called the bias term. Technically, this is actually also a polynomial featurization of degree 0.

[1] was a source of inspiration and reference for the writing of this featurization section.

5 Linear Regression for Classification

Another important problem in machine learning is **classification**. Classification is different from regression in that we are interested in assigning each data point a class from a discrete set (e.g. "cat" or "dog"), instead of assigning it a continuous value. Specifically, assume we have access to data $\mathcal{D} = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$ of n data points $\mathbf{x_i} \in \mathbb{R}^d$ and their corresponding **labels** $y_i \in \{1, 2, ..., K\}$, enumerated for convenience. We are interested in finding a function $\hat{y} = f(\mathbf{x})$ that, when fed any arbitrary data point $\mathbf{x}$, correctly classifies that data point as $\hat{y}$, one of the K discrete classes. The function $f(\mathbf{x})$ is called the **decision function**.

For this assignment, we are going to restrict our attention to linear regression and see how even this regression technique can be used to solve classification problems.

5.1 Binary Classification

Let's start by looking at the binary classification case, which is when $K = 2$ and the label is either positive or negative, represented by +1 and -1. Since we are using linear regression, the decision function will be of the form $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. A problem, however, is that the output can be a continuous value instead of strictly 1 or -1. One way to handle that is to estimate the parameters $\mathbf{w}$, compute $\mathbf{w}^T\mathbf{x}$, and classify $\mathbf{x}$ to be **sign**($\mathbf{w}^T\mathbf{x}$). In other words, we assign $\mathbf{x}$ to class 1, if $f(\mathbf{x})$ is positive and to class -1, otherwise:

$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{if } \mathbf{w}^T\mathbf{x} \leq 0 \end{cases}$$

Here we have adopted the convention that if $\mathbf{w}^T\mathbf{x} = 0$ we choose class -1, but in reality those points can be classified in either of the two classes. The set of points $\{\mathbf{x}|\mathbf{w}^T\mathbf{x} = 0\}$ defines a hyperplane, called the **decision boundary**, that separates points in one class from those in the other. Whether it is possible to find such a linear decision boundary that perfectly separates the data depends on the input data itself as well as the choice of features. If there exists a hyperplane that separates all the sample points in class 1 from all the points in class -1, we say the data **linearly separable**. Finally, we can summarize the method of binary classification using linear regression as follows:

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

$$\mathbf{w} = \text{arg min}_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2$$

2. Given a data point $\mathbf{x}$, compute $\mathbf{w}^T\mathbf{x}$.
3. Classify $\mathbf{x}$ as **sign**($\mathbf{w}^T\mathbf{x}$).

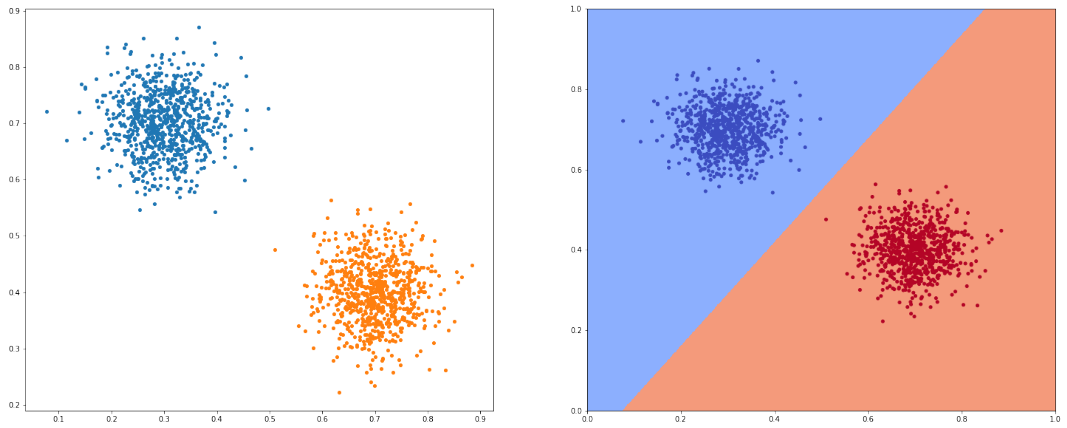


Figure 7: Linearly separable binary data (left) and the linear regression decision boundary (right).

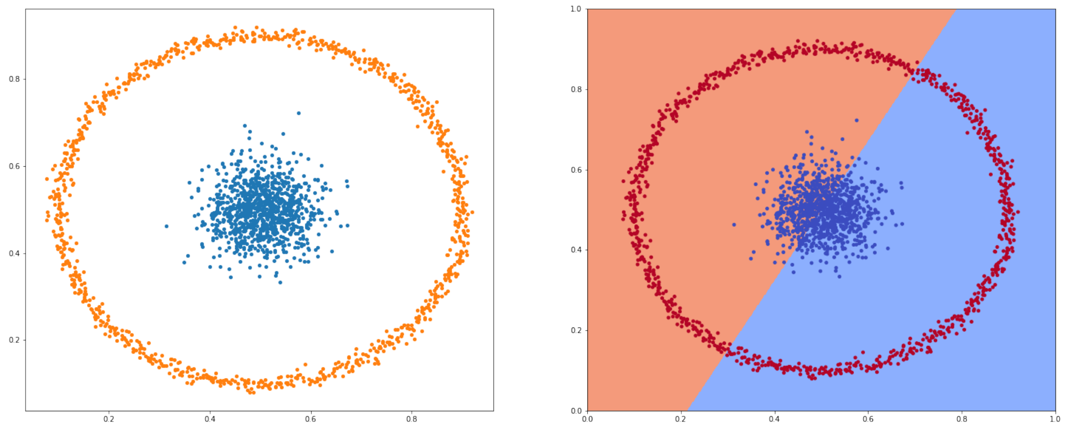


Figure 8: Non-linearly separable binary data (left) and the linear regression decision boundary (right).

5.2 Multinomial Classification

Let's generalize our method to the case where we have more than 2 classes, also known as **multinomial** or **multiclass** classification. One possible way to extend the approach from binary classification is to compute $\mathbf{w}^T\mathbf{x}$ and round it to the nearest number from 1 to K. However, this presents a problem in that we give the classes an ordering even if no such natural ordering exists. For example, in the notebook you will attempt to classify penguins as one of three species which we can label as 1,2, and 3. With this numerical encoding, the first species is represented as less than the second species, which in turn is less than the third one. As a result, if we have a penguin that is a cross between the first and third species, we might end up simply classifying it as the second species.

An alternative approach it to use a strategy called **One-vs-All** or **One-vs-Rest**. The idea is to have a classifier for each class that separates this class from all the others, hence the name of this technique. We will represent the points of that class as positive and all other points as negative. This means we will have K weight vectors w_k and K label vectors y_k, $k = 1, 2, ..., K$. In the end we will pick the class whose decision function produces the highest value. This method allows us to circumvent the ordering issue and we can interpret it as choosing the class which we are most confident in.

Finally, we can summarize the method of multinomial classification using linear regression as follows:

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

   $$\mathbf{w_k} = \text{arg min}_{\mathbf{w}} ||\mathbf{Xw} - \mathbf{y_k}||_2^2$$

   - 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

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

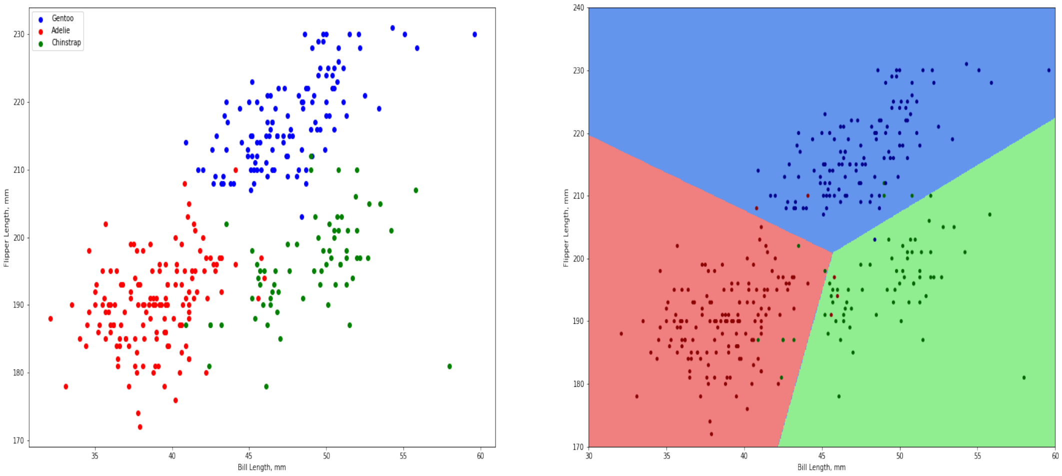


Figure 9: Visualization of penguins dataset from the notebook (left) and the linear regression decision boundaries (right).

References

[1] Nasiriany, S., Thomas, G., Wang, W., Yang, A., Listgarten, J., & Sahai, A. (2019) A Comprehensive Guide to Machine Learning. Retrieved November 30, 2020, from https://www.eecs189.org/static/notes/n3.pdf

[2] Joseph, A. D., Adhikari, A., & Rampure, S. (Fall 2020) Data 100 Lecture 12 - Simple Linear Regression, UC Berkeley. Retrieved November 30, 2020, from http://www.ds100.org/fa20/lecture/lec12/

[3] nkuhta. (2017) Linear-Regression-Python Github. Retrieved November 30, 2020, from https://github.com/nkuhta/Linear-Regression-Python

[4] Wagner, D. (Fall 2020) Data 8 Linear Regression. Retrieved November 30, 2020, from https://docs.google.com/presentation/d/1do4KBtTIsKBT4muk0ir2NnEGDo8F3Q1ALlhHLIWSF6Q/edit#slide=id.g610d9f86d0_0_5

[5] Nasiriany, S., Thomas, G., Wang, W., Yang, A., Listgarten, J., & Sahai, A. (2019) A Comprehensive Guide to Machine Learning. Retrieved November 30, 2020, from https://www.eecs189.org/static/notes/n2.pdf

[6] Allain, Rhett. (2011) Linear Regression By Hand. Retrieved November 30, 2020, from https://www.wired.com/2011/01/linear-regression-by-hand/

[7] Grobler, Jaques. Linear Regression Example. Retrieved November 30, 2020, from https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html#sphx-glr-auto-examples-linear-model-plot-ols-py

[8] Lau, S., Gonzalez, J., amp; Nolan, D. (2020). Principles and Techniques of Data Science 100, UC Berkeley. Retrieved November 30, 2020, from https://www.textbook.ds100.org/ch/13/linear_projection.html

[9] Lau, S., Gonzalez, J., amp; Nolan, D. (2020). Principles and Techniques of Data Science 100, UC Berkeley. Retrieved November 29, 2020, from https://www.textbook.ds100.org/ch/14/feature_one_hot.html

[10] Gonzalez, J. (2020). Principles and Techniques of Data Science 100 Feature Engineering Notebook, UC Berkeley. Retrieved November 29, 2020, from http://www.ds100.org/sp17/assets/notebooks/linear_regression/Feature_Engineering_Part1.html

[11] Amir Beck (2014). Introduction to Nonlinear Optimization Theory, Algorithms, and Application with MATLAB.