I don't trust linear regressions when its harder to guess the direction of the correlation from the scatter plot than to find new constellations on it.
Let’s move away from decision trees (yay!) and talk about more general learning.

Today we will look at regression a bit (as I have been ignoring it mostly).

This is a concept that you may have encountered before, but not in the context of learning.
Linear Regression

Idea: You have a bunch of data points, and you want to find the line “closest” to them.
Linear Regression

Why linear and not some polynomial?
Linear Regression

Why linear and not some polynomial?

It is a lot harder to “overfit” with a linear fit, yet it still gets the major “trend” of data

Also hard to “visualize” data if high dimension

Another bonus is that it makes the calculations much easier (which is always nice...)
Linear Regression: How To

To find this line, let’s start with the simple case: only one variable

Then our line will look like (call them “h” just like our learning trees):

\[ h_w(x) = y = w_0 + w_1 \cdot x \]

w is \( \{w_0, w_1\} \) as parameters

Then we need to define what “fit to data” means (i.e. how do we calculate how “wrong” a line is, often called the “loss”)

Linear Regression: How To

There are multiple options, but a common choice is the square difference, so “loss” is:

$$\text{Loss}(h_w) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_0 + w_1 \cdot x_j))^2$$

- $y_j$ is actual y-coordinate
- $h_w(x_j)$ is approximated (line) y-coordinate

... where N is the number of examples/points

This makes sense as it penalizes “wrong” answers more the further they are away (two points off by 1 better than one off by 2)
Linear Regression: How To

You can plot this loss function (z-axis) with respect to the choice of $w_0$ and $w_1$
Linear Regression: How To

We want the regression line \((w_0, w_1)\) to have the lowest loss possible.

As the loss function looks convex (it is), the minimum is unique, so from calculus we want:

\[
\frac{\partial}{\partial w_0} \text{Loss}(h_w) = 0 = \frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_0 + w_1 \cdot x_j))^2
\]

bottom is when both \(w_0\) and \(w_1\) derivatives zero

\[
\frac{\partial}{\partial w_1} \text{Loss}(h_w) = 0 = \frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_0 + w_1 \cdot x_j))^2
\]
It is not too hard to do a bit of calculus to find the unique solution for $w_0$ and $w_1$:

\[
\begin{align*}
\begin{align*}
    w_1 &= \frac{N \cdot (\sum_j x_j \cdot y_j) - (\sum_j x_j) \cdot (\sum_j y_j)}{N \cdot (\sum_j x_j^2) - (\sum_j x_j)^2} \\

    w_0 &= \frac{(\sum_j y_j) - w_1 \cdot (\sum_j x_j)}{N}
\end{align*}
\]

all sum from $j=1$ to $N$

Unfortunately, if you want to do polynomials, you might not have a closed form solution like this (i.e. no “easy” exact answer)
Linear Regression: Estimate

You can do a gradient descent (much like Newton’s method)(similar to “hill-climbing”)

\[ w_{\text{new}} = w_{\text{old}} - \alpha \cdot \nabla f(w_{\text{old}}) \]
Linear Regression: Estimate

Again, you need calculus to figure out what direction is “down hill”, so to move the weights \((w_0, w_1, ...)\) towards the bottom:

\[
w_i \leftarrow w_i - \alpha \cdot \frac{\partial}{\partial w_i} Loss(h_{w_i})
\]

Loss function is what we minimizing (convex), so derivative of it ...

... where \(\alpha\) is basically the “step size” (we will often use alpha in a similar fashion, but call it the “learning factor/rate”)

Linear Regression: Estimate

The choice of $\alpha$ is somewhat arbitrary...

You don’t want it too big, but anything small is fine (even better if you shrink it over time)
Linear Regression: Estimate

You can extend this to more than just one variable (or attribute) in a similar fashion:

If we have X as (for attributes a, b, c ...):

\[
X = \begin{bmatrix}
1 & a_1 & b_1 & c_1 & \ldots \\
1 & a_2 & b_2 & c_2 & \ldots \\
1 & a_3 & b_3 & c_3 & \ldots \\
\vdots & \vdots & \vdots & \vdots & \ddots \\
\end{bmatrix}
\]

... and w as:

\[
w^T = [w_0 \ w_1 \ w_2 \ w_3 \ \ldots]
\]
Linear Regression: Estimate

Then if $x_j$ is a single row of $X$:

$$(x_j)^T = [1 \ a_j \ b_j \ c_j \ ...]$$

Then our “line” is just the dot product:

$$w_0 + w_1 \cdot a_j + w_2 \cdot b_j + w_3 \cdot c_j + ... = w \cdot x_j = w^T x_j$$

Just like for the single variable case, we update our $w$’s as:

$$w_i \leftarrow w_i - \alpha \cdot \frac{\partial}{\partial w_i} \text{Loss}(h_w)$$

attribute for the corresponding weight in example, so if updating “$w_2$” then “$b_j$” as in line we do “$w_2 \cdot b_j$”

... after math:

$$w_i \leftarrow w_i + \alpha \cdot \sum_{j=1}^{N} x_{j,i} (y_j - h_w(x_j))$$

$y_j$ is actual output for example/point number $j$
Let’s do an example...

Assume we have the following data points: (assume trying to calculate BMI or something)

<table>
<thead>
<tr>
<th>Age</th>
<th>Height</th>
<th>Calories</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>1500</td>
</tr>
<tr>
<td>22</td>
<td>50</td>
<td>2500</td>
</tr>
<tr>
<td>50</td>
<td>48</td>
<td>2000</td>
</tr>
</tbody>
</table>

We will try to fit (age + height) to estimate Cal
Linear Regression: Example

So we will need to come up with some equation to predict Calories in the form:
\[ w_0 + w_1 \cdot \text{age} + w_2 \cdot \text{height} \]

Initially, we can pick whatever for the \( w \)'s:
\[ w_0 = 550, \ w_1 = -20, \ w_2 = 50 \]

From this we can calculate the loss:

\[
\text{Loss}(h_w) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_0 + w_1 \cdot \text{age} + w_2 \cdot \text{height}))^2
\]
Linear Regression: Example

\[ \text{Loss}(h_w) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_0 + w_1 \cdot \text{age} + w_2 \cdot \text{height}))^2 \]

From our data:
Loss = \((1500 - (550 + -20\times5 + 50\times20))^2\)
+\((2500 - (550 + -20\times22 + 50\times50))^2\)
+\((2000 - (550 + -20\times50 + 50\times48))^2\)
Loss = 17,100
... this is pretty big so, these w’s aren’t great

Let’s update \(w_2\) to make it better!
Linear Regression: Estimate

Use update formula (with $\alpha=0.0001$) for $w_2$:

$$w_2 = 50 + 0.0001 \times \left[ 20 \times (1500 - (550 + -20 \times 5 + 50 \times 20)) \\
+ 50 \times (2500 - (550 + -20 \times 22 + 50 \times 50)) \\
+ 48 \times (2000 - (550 + -20 \times 50 + 50 \times 48)) \right]$$

$$w_2 = 49.79$$

(Note: $\alpha$ chosen small as Loss was large and we don't want $w_2$ to change by a large amount)
Linear Regression: Estimate

\[
Loss(h_w) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_0 + w_1 \cdot \text{age} + w_2 \cdot \text{height}))^2
\]

We can then re-calculate the loss with the new \(w_2 = 49.79\):

\[
\begin{align*}
\text{Loss} &= (1500 - (550 + -20 \cdot 5 + 49.79 \cdot 20))^2 \\
&\quad + (2500 - (550 + -20 \cdot 22 + 49.79 \cdot 50))^2 \\
&\quad + (2000 - (550 + -20 \cdot 50 + 49.79 \cdot 48))^2 \\
\text{Loss} &= 16,447.5
\end{align*}
\]

... which is better than the old \(\text{Loss} = 17,100\).
Linear Regression: Exact

However, you can solve for linear regression exactly even with multiple inputs

Specifically, you can find optimal weights as:

\[ w = (X^T X)^{-1} X^T y \]

This requires you to find a matrix inverse, which can be a bit ugly... but do-able

Thus we estimate our line as: \[ Xw \approx y \]
So for:

<table>
<thead>
<tr>
<th>Age</th>
<th>Height</th>
<th>Calories</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>1500</td>
</tr>
<tr>
<td>22</td>
<td>50</td>
<td>2500</td>
</tr>
<tr>
<td>50</td>
<td>48</td>
<td>2000</td>
</tr>
</tbody>
</table>

\[ X = \begin{bmatrix}
1 & 5 & 20 \\
1 & 22 & 50 \\
1 & 50 & 48 \\
\end{bmatrix} \]

... thus the “best fit” w’s are:

\[
\begin{bmatrix}
w_0 \\
w_1 \\
w_2 \\
\end{bmatrix} = \left( \begin{bmatrix}
1 & 1 & 1 \\
5 & 22 & 50 \\
20 & 50 & 48 \\
\end{bmatrix} \right)^{-1} \begin{bmatrix}
1 & 1 & 1 \\
1 & 5 & 20 \\
1 & 22 & 50 \\
5 & 22 & 50 \\
20 & 50 & 48 \\
\end{bmatrix} \begin{bmatrix}
w_0 \\
w_1 \\
w_2 \\
\end{bmatrix} = \begin{bmatrix}
1 & 1 & 1 \\
5 & 22 & 50 \\
20 & 50 & 48 \\
\end{bmatrix}^{-1} \begin{bmatrix}
1500 \\
2500 \\
2000 \\
\end{bmatrix}
\]
You actually still can overfit even with a linear approximation by using too many variables (can’t overfit “trend”)

Another option to minimize (rather than loss):

\[
\text{Cost}(h_w) = \text{Loss}(h_w) + \lambda \cdot \text{Complexity}(h_w)
\]

... where we will treat \( \lambda \) as some constant

and:

\[
\text{Complexity}(h_w) = L_p(w)
\]

... where \( L_p(w) \) is similar to the p-norm
Side note: “Distance”

The p-norm is a generalized way of measuring distance (you already know some of these)

The definition is of a p-norm:

$$\|x\|_p = (|x_1|^p + |x_2|^p + ...)^{1/p}$$

Specifically in 2 dimensions:

$$\|(x, y)\|_1 = |x| + |y|$$

(Manhattan distance)

$$\|(x, y)\|_2 = \sqrt{x^2 + y^2}$$

(Euclidean distance)
Linear Regression: Overfitting

We drop the exponent for L’s, so in 2D:

\[ \| (x, y) \|_2 = \sqrt{x^2 + y^2} \]
\[ L_2(x, y) = x^2 + y^2 \]

So we treat the weight vector’s “distance” as the complexity (to minimize)

Here \( L_1 \) is often the best choice as it tends to have 0’s on “irrelevant” attributes/variables

... why are 0’s good? Why does it happen?
Linear Regression: Overfitting

This is because the $L_1$ (Manhattan distance) has a sharper “angle” than a circle ($L_2$) has $w_1 = 0$, as on y-axis ... so $w_1$ seems irrelevant (less overfit)
Linear Classification

A similar problem is instead of finding the “closest” line, we want to find a line that separates the data points (assume T/F for data).

This is more similar to what we were doing with decision trees, except we will use lines rather than trees.
Linear Classification

This is actually a bit harder than linear regression as you can wiggle the line, yet the classification stays the same.

This means, most places the derivative are zero, so we cannot do simple gradient descent.

To classify we check if:

\[ h_w(x) = w \cdot x > y \]

... if yes, then guess True... otherwise guess F
Linear Classification

For example, in three dimensions:

\[ w_1 \cdot x + w_2 \cdot y + w_3 \cdot z > c \]

This is simply one side of a plane in 3D, so this is trying to classify all possible points using a single plane...

\[ c = -w_0 \]

y is not “output” atm
Linear Classification

Despite gradient descent not working, we can still “update” weights until convergence as:

$$w_i \leftarrow w_i + \alpha \cdot (y - h_w(x)) \cdot x_i$$

Start weight randomly, then update weight for every example with above equation

... what does this equation look like?
Despite gradient descent not working, we can still “update” weights until convergence as:

\[ w_i \leftarrow w_i + \alpha \cdot (y - h_w(x)) \cdot x_i \]

Start weight randomly, then update weight for every example with above equation

... what does this equation look like? Just the gradient descent (but I thought you said we couldn’t since derivative messed up!)
Linear Classification

If we had only 2 inputs, it would be everything above a line in 2D, but consider XOR on right

There is no way a single line can classify XOR

... what should we do?
Linear Classification

If one line isn’t enough... use more!
Our next topic will do just this...