Nov 18 Classifiers Intro

use: In class MDP Value Iteration.

1. Homework 4 posted! Due Nov 25.

Last time we learned:

1. Active Reinforcement Learning.

Do M's minute form right now

Definition: An **active learning** agent is allowed to choose between different policies as it tries to find an optimal policy.

1. Start with a policy π
2. Learn how good that policy is in terms of rewards.
3. Adapt and improve upon π

Definition: An $\varepsilon-$greedy agent is a way to force exploration.

1. Keeps track of expected payouts $Q[k]$
2. Picks a *random* action with probability ε (10%?)
3. Picks the current best estimated action with probability $1-\varepsilon$

Definition: An **active learning** agent is allowed to choose between different policies as it tries to find an optimal policy.

1. Start with a policy π
2. Learn how good that policy is in terms of rewards.
3. Adapt and improve upon π

Definition: An $\varepsilon-$greedy agent is a way to force exploration.

1. Keeps track of expected payouts $Q[k]$
2. Picks a *random* action with probability ε
3. Picks the current best estimated action with probability $1 - \varepsilon$

Active functions

Option B: use a function to make exploration more appealing *for a while*. We can then tell an agent to "always choose the action with the highest f, where we provide an inflated reward R^+ to states with small sample sizes:

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & otherwise \end{cases}$$

Define $Q(s,a) :=$ The *expected value* of doing action a in state s. For any state,

$$U(s) = \max_a Q(s,a)$$

Active $Q-$learning: **Update** $Q(s,a)$ by finding the difference between $Q(s,a)$ and its observed successor $Q(s',a)$.

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a) \right]$$

Active functions

Option B: use a function to make exploration more appealing *for a while*. We can then tell an agent to "always choose the action with the highest f, where we provide an inflated reward R^+ to states with small sample sizes:

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & otherwise \end{cases}$$

Define $Q(s,a)$:= The *expected value* of doing action a in state s. For any state,

$$U(s) = \max_a Q(s,a)$$

Active $Q-$learning: **Update** $Q(s,a)$ by finding the difference between $Q(s,a)$ and its observed successor $Q(s',a)$. α 1/n where n is the # of times we've already taken action a from state s.

$$\underbrace{Q_{i+1}(s,a)}_{new\ update} = \underbrace{Q_i(s,a)}_{old\ value} + \alpha \Big[\underbrace{R(s)}_{immediate\ reward} + \underbrace{\gamma \max_{a'} Q_i(s',a')}_{value\ of\ where\ we\ went} - \underbrace{Q_i(s,a)}_{difference} \Big]$$

$Q-$learning:

$$Q_{i+1}(s,a) = Q_i(s,a) + \alpha\left[R(s) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)\right]$$

Initialize some starter policy $\pi(s)$, initial action-utilities $Q(s,a)$ for the known states. Suppose we start a training episode.

1. Set initial state s_0.
2. Add our percept $(s, R(s))$ to the known state space
3. Sample some subsequent state s'
   - 3.1 This first sample is generated from the *initialized* policies.
   - 3.2 This gives a new state s'. We have an initialized set of Q values for s', so we know $Q_i(s',a')$.
   - 3.3 We can now update $Q_{i+1}(s,a)$ as above.
   - 3.4 Crucially, we can *also* now update $\max_{a'} Q(s,a')$ for our *original* state s.

Then we repeat this process (our new state s' becomes our decision-location s) until the training episode ends.

Q-Learning

So this is a double loop:

FOR MANY TRAINING EPISODES:

FOR STEPS UNTIL THAT EPISODE IS COMPLETED:

1. Sample some subsequent state s' *from the exploration function* $f(u, n)$. The "best-case" exploration values $R+$ can be *very* optimistic... what kinds of models might we have for these sorts of things?
2. Since we know $Q_i(s', a')$, we can now update $Q_{i+1}(s, a)$ as prior slide.
3. *Also* update $\max_{a'} Q(s, a')$ for our *original* state s. One way: run a loop over the old state:
   - 3.1 $bestQ = -Inf$
   - 3.2 for all $a in A(s)$, $bestQ = \max(bestQ, Q[s', a'])$

$\forall a \in A(s)$

Now that we're also using our exploration function, our agent will both *explore* and try to *exploit* once n is sufficiently large.

Active Learning Underview

We discussed 3 different ways to balance the exploration/exploitation tradeoff.

1. $\varepsilon-$ greedy:
   force exploration ε proportion of the time.
2. Functional rewards for exploration:
   encourage exploration until we've run each action-state pair N_e times.
3. $Q-$Learning.
   update action-pair utilities (combined with 2.) to *learn* while converging to an optimal action.

Linear Regression Review:

Definition: *Simple Linear Regression* (SLR)

The *Simple Linear Regression* model is a model of the form

With 3 assumptions on ε:

Linear Regression Review:

Definition: *Simple Linear Regression* (SLR)

The *Simple Linear Regression* model is a model of the form

1. $y = mx + b$

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

intercept, slope

With 3 assumptions on ε:

Linear Regression Review:

Definition: *Simple Linear Regression* (SLR)

The *Simple Linear Regression* model is a model of the form

1.
$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

With 3 assumptions on ε:

2.
$$\text{Cov}[\varepsilon_i, \varepsilon_j] = 0 \qquad \forall i, j$$

Independence of errors

Linear Regression Review:

Definition: *Simple Linear Regression* (SLR)

The *Simple Linear Regression* model is a model of the form

1. $$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

   With 3 assumptions on ε:

2. $$\text{Cov}[\varepsilon_i, \varepsilon_j] = 0 \qquad \forall i, j$$

   Independence of errors

3. $$Var(\varepsilon_i) = \sigma^2 \qquad \forall i$$

   Homoskedasticity of errors

Linear Regression Review:

Definition: *Simple Linear Regression* (SLR)

The *Simple Linear Regression* model is a model of the form

1. $$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

   With 3 assumptions on ε:

2. $$\text{Cov}[\varepsilon_i, \varepsilon_j] = 0 \qquad \forall i, j$$

   Independence of errors

3. $$Var(\varepsilon_i) = \sigma^2 \qquad \forall i$$

   Homoskedasticity of errors

4. $$\varepsilon_i \sim N(0, 1)$$

   Normality

error is i.i.d $N(0, \sigma^2)$

Linear Regression Process:

The *Simple Linear Regression* model is a model of the form

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Given sample data, which consists of n observed pairs, $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$, construct an estimated "line of best fit":

$$y = \hat{\beta}_0 + \hat{\beta}_1 x$$

estimators of slope & intercept

$\hat{\beta}_0; \hat{\beta}_1$ are called the *least squares estimates*. They are those values that minimize SSE or sum of squared errors.

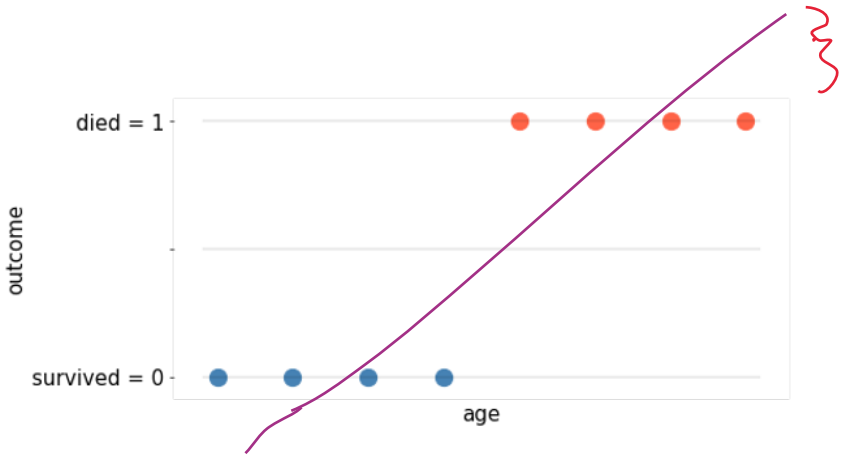
data

$$\hat{\beta}_0, \hat{\beta}_1 = \underset{\beta_0, \beta_1}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - (\beta_0 - \beta_1 X_i))^2$$

our line

Binary Outcomes

Goal: predict whether a passenger survives or not as a linear function of their age.

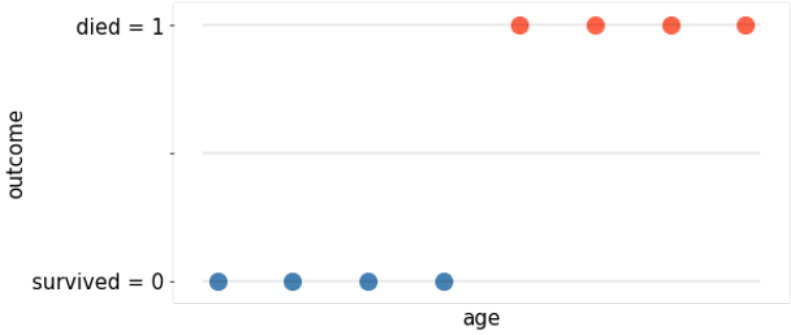


Binary Outcomes

Goal: predict whether a passenger survives or not as a linear function of their age.

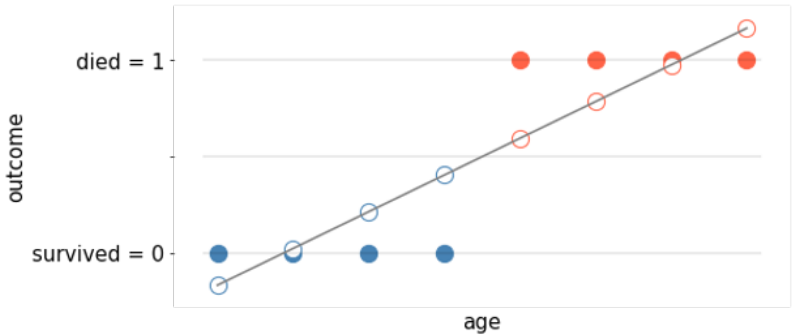
$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$$

Single input of X :=age, output desired is a prediction of {0,1}={Survived, Died}



Binary Outcomes... as a line?

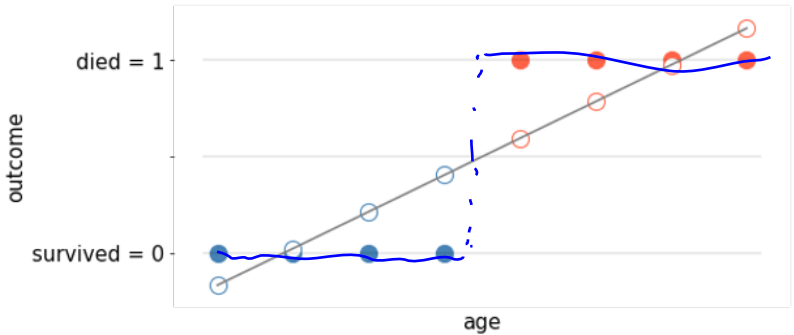
Goal: predict whether a passenger survives or not as a linear function of their age:



Binary Outcomes... as a line?

Goal: predict whether a passenger survives or not as a linear function of their age:
Did it work? We could use this line to do some kind of classification, e.g. use a piecewise function like

$$y = \begin{cases} 0 & \text{if } x < \text{threshold} \\ 1 & \text{if } x \geq \text{threshold} \end{cases}$$



Binary Outcomes and error

Directly using Y here is a bit awkward though.

1. What do our errors represent in general?
2. How do we interpret linear fits less than zero and greater than 1?
3. We really want a measure here that behaves more like *probability.*

Binary Outcomes and error

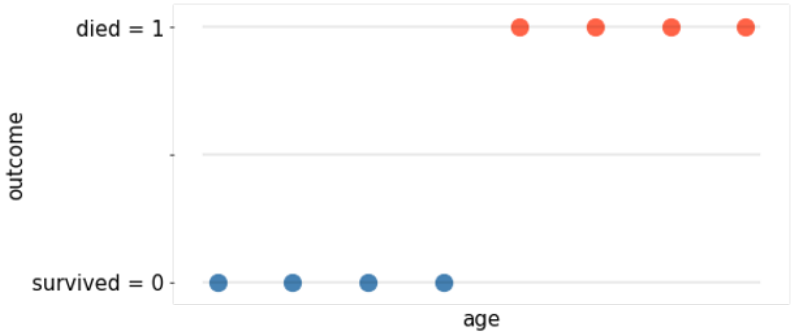
X (input)	y (prob)
∞	1
0	.5
$-\infty$	0

Directly using Y here is a bit awkward though.

1. What do our errors represent in general?
2. How do we interpret linear fits less than zero and greater than 1?
3. We really want a measure here that behaves more like *probability.*
4. Let's reach into the math toolbox... we need a function that an input (*domain*) of (0,1) and will give us a range of $\mathbb{R}$... or vice versa, and we can invert it.

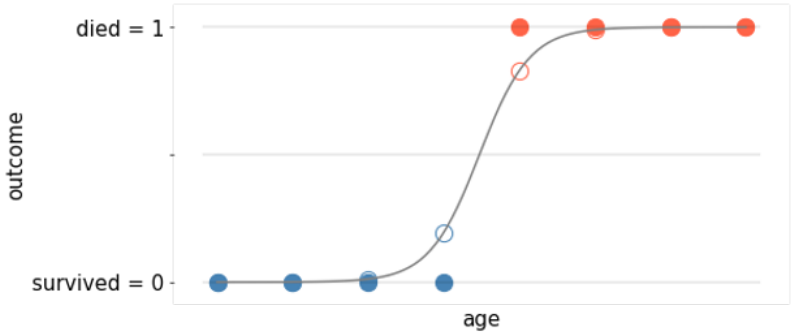
Binary Outcomes: a better fit

Goal: predict whether a passenger survives or not as a linear function of their age:
Let's sketch what we might want a probability-based function to look like:



The Sigmoid

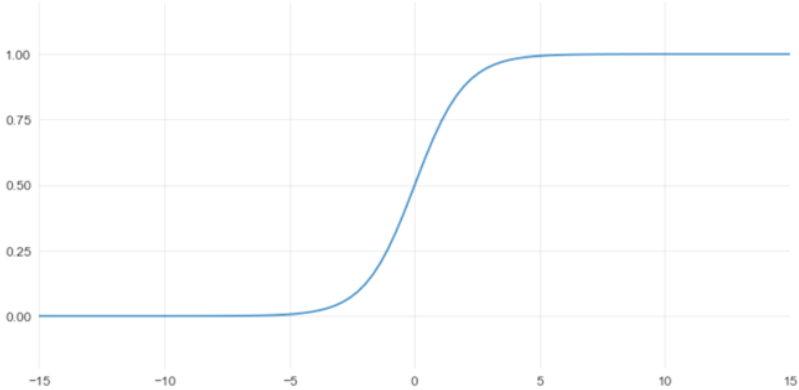
Goal: predict whether a passenger survives or not as a linear function of their age:
How's this thing look?



The Sigmoid

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

has the properties we want.



How would we move it left-to-right? How would we make it steeper or shallower?

The Sigmoid

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

$$\frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

1. We can use something like $\text{sigm}(\beta_0 + \beta_1 x)$: this gives us dials to both control where the sigmoid moves from 0 to 1 and how steeply it does so.
2. Because the output of the sigmoid is (0,1), we can treat it's outputs like probabilities.
3. It's really smooth. This means we're probably not naturally over-fitting!

Model:

$$P(Y = 1|X) = \text{sigm}(\beta_0 + \beta_1 X)$$

Recall C.P. tables:
Bayes Net
HMM
MDP.

Finding our Sigmoid

Model:

$$P(Y = 1|X) = \text{sigm}(\beta_0 + \beta_1 X)$$

Plan:

learn the weights of $\beta_0 + \beta_1 X$ from the data by estimating them:
I.E. find suitable $\hat{\beta}_0$; $\hat{\beta}_1$

Classify: Gain the ability to describe point x according to

$$\hat{y} = \begin{cases} 1 & \text{if sigm}(\hat{\beta}_0 + \hat{\beta}_1 X) \geq .5 \\ 0 & \text{if sigm}(\hat{\beta}_0 + \hat{\beta}_1 X) < .5 \end{cases}$$

Finding our Sigmoid

This is actually a regression problem! But we have to take the inverse of the sigmoid function to solve it so it looks like $g(y) = \beta_0 + \beta_1 x$. The function that does is is the concept of *odds*.

In statistics, the *odds* of an event occuring is defined as the ratio of the probability of an event occuring divided by the probability of it not occuring. It is often then flipped to be guaranteed larger than 1, at least for colloquial use. So

Odds

In statistics, the *odds* of an event occuring is defined as the ratio of the probability of an event occuring divided by the probability of it not occuring. It is often then flipped to be guaranteed larger than 1, at least for colloquial use. So

$$\text{Odds} =$$

Example: If $p = .75$, the odds are _____;
and we might say: _____.

Example: If $p = .1$, the odds are _____;
and we might say: _____.

Odds

In statistics, the *odds* of an event occuring is defined as the ratio of the probability of an event occuring divided by the probability of it not occuring. It is often then flipped to be guaranteed larger than 1, at least for colloquial use. So

$$\text{Odds} = \frac{p}{1-p}$$

Example: If $p = .75$, the odds are $\frac{3/4}{1/4} = 3$;
and we might say: 3 to 1 in favor.

Example: If $p = .1$, the odds are $\frac{.1}{.9} = \frac{1}{9}$;
and we might say: 9 to 1 against.

Odds are this is useful

In logistic regression, we're using a model of the probability p, or

$$p = P(Y = 1|X) = \text{sigm}(\beta_0 + \beta_1 X)$$

What does this mean in terms of odds?

$$\text{Odds} = \frac{p}{1-p}$$

Odds are this is useful

In logistic regression, we're using a model of the probability p, or

$$p = P(Y = 1|X) = \text{sigm}(\beta_0 + \beta_1 X)$$

What does this mean in terms of odds? (using z as our line, for shorthand)

$$\begin{aligned} \text{Odds} &= \frac{p}{1-p} \\ &= \frac{\frac{1}{1+e^{-z}}}{1 - \frac{1}{1+e^{-z}}} \end{aligned}$$

Odds are this is useful

In logistic regression, we're using a model of the probability p, or

$$p = P(Y = 1|X) = \text{sigm}(\beta_0 + \beta_1 X)$$

What does this mean in terms of odds?

$$\begin{aligned}
\text{Odds} &= \frac{p}{1-p} \\
&= \frac{\frac{1}{1+e^{-z}}}{1 - \frac{1}{1+e^{-z}}} \\
&= \frac{1}{1 + e^{-z} - 1} = e^z = e^{\beta_0 + \beta_1 X}
\end{aligned}$$

Odds are this is useful

$\ln\left(\frac{y}{1-y}\right) = \beta_0 + \beta_1 X$

fit with:

Statsmodels.OLS

.fit()

Taking the log of both sides gives

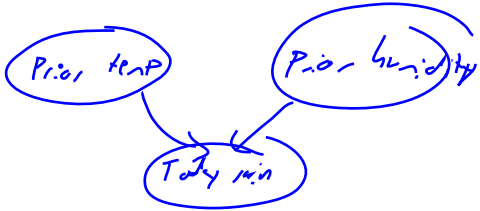
$$\ln \text{Odds} = \beta_0 + \beta_1 X$$

which is a simple linear regression problem!

Only this time, a one-unit increase in X represents an increase of β_1 in the *Log-Odds* of y.

If y had begun as a probability between 0 and 1 instead of exactly 0 or 1, we could have taken $y_{new} = \ln \frac{y}{1-y}$ as a transformation of y that would move it from (0,1) to (∞, ∞). This is the same intepretation as logistic regression, but we can't actually apply that transformation since the actual data values are 0 and 1!

Big Logistic Regression



The simple logistic regression model is then $P(Y = 1|X) = \text{sigm}(\beta_0 + \beta_1 X)$.

In reality, we will have many features or predictors. For example:

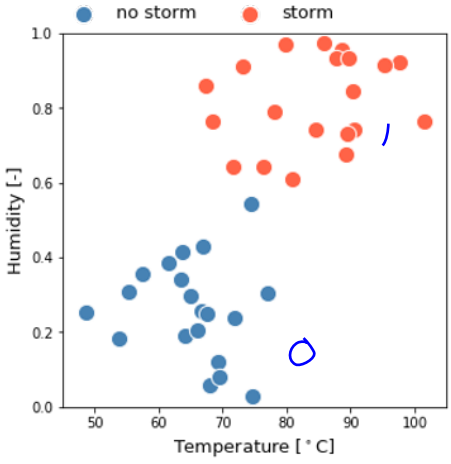
Predict the probability of precipitation or a storm

Given temperature, pressure, humidity, whether it rained yesterday (or a week ago!), etc.

Big Logistic Regression

Predict Was there a storm (if storm then, $Y = 1$)?

Features X_1 :=Temp, X_2 :=humidity



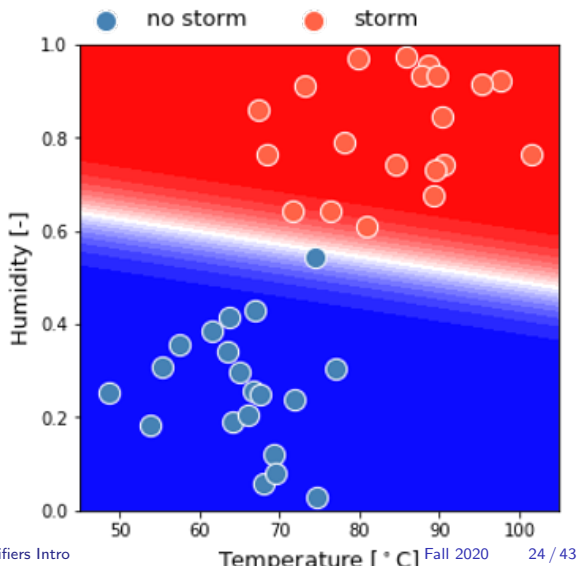
Model:

$$P(\text{Storm}) = \text{sigm}(\beta_0 + \beta_1 X_1 + \beta_2 X_2)$$

Prediction Surface

The result might be a *surface* of log-odds values which we could transform back to probabilities.

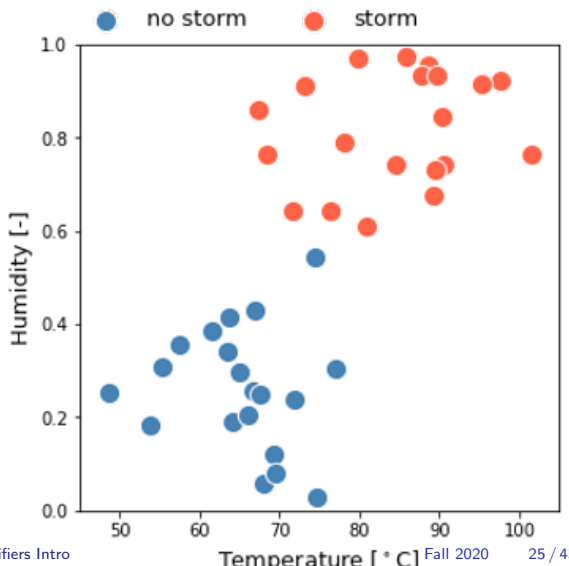
$P(\text{storm} \mid \text{temp}, \text{hum}) \rightarrow$



Prediction Boundary

Of course, we also might only be using the line that defines the boundary: that's the criteria we use to decide whether our best guess $\hat{y} = 1$ or not.

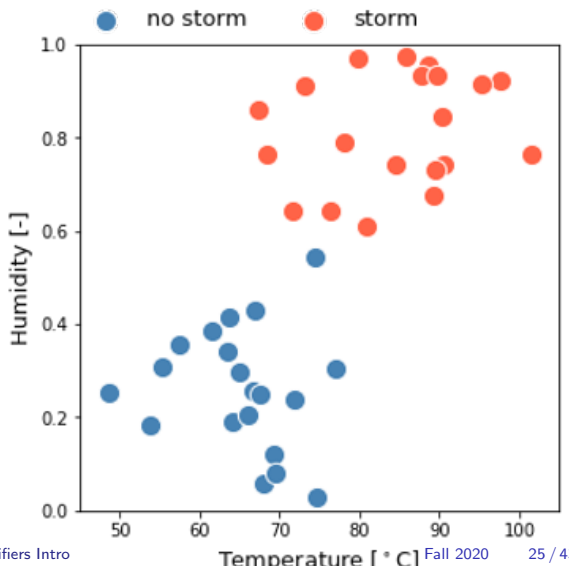
The sigmoid function happens to have a nice property that $f'(z) = f(z)(1 - f(z))$, which can make for very useful calculus!



Prediction Boundary

Of course, we also might only be using the line that defines the boundary: that's the criteria we use to decide whether our best guess $\hat{y} = 1$ or not.

The sigmoid function happens to have a nice property that $f'(z) = f(z)(1 - f(z))$, which can make for very useful calculus! **Steepest** at .5... sounds useful.



Prediction Boundary

The goal of finding a prediction boundary: like a line that might satisfy some appealing properties is a major algorithmic concern in CSCI4622: Machine Learning.

In that class, we rewrite the problem as "how do we best draw a line (or (hyper-)plane!) to separate the 0's and the 1's." Which side of the line a new point falls on becomes our classifier. This is called a support-vector machine (SVM).

Optimization

In many problems, we're tasked then with finding the "best" mathematical object, which typically means something like "maximum probability" or "minimum error/cost".

In regression, we find the best estimators of the vector β in the model

$$\underline{Y} = \underline{\beta}\boldsymbol{X} + \varepsilon$$

by minimizing the sum of squared errors

$$\text{SSE} = \sum_{i=1}^{n} \left(y_i - (\beta_0 + \beta_1 x_{1,i} + \cdots + \beta_p x_{p,i})\right)^2$$

Logistic regression is the same, it just minimizes the errors or differences between y and a sigmoid function of the X values.

Many Variables, many tools

In the presences of lots of data and even more so, **lots of features**, these computations are difficult. The direct solutions rely on multivariate calculus, and lead slow, memory-intensive, and occasionally numerically unstable linear algebra.

More commonly, we use an iterative method that *converges* after many steps to the desired β values. We did one of these methods in Hill-climbing, which is a for of *gradient ascent*. *Stochastic* hill climbing is more like simulated annealing, and might involve random jumps!

This might lead to a model that uses 3-4 tools in this class at once! If you want to make a weather-predictive AI, you might:

1. Start with an HMM for measurements *given* weather and weather *given* the prior day(s) properties.
2. Create a linear model or logistic classifier to answer the weather *given* the prior day(s) question.
3. Use a hill-climbing function maximizer to ensure that your classifier is the *best* classifier possible!

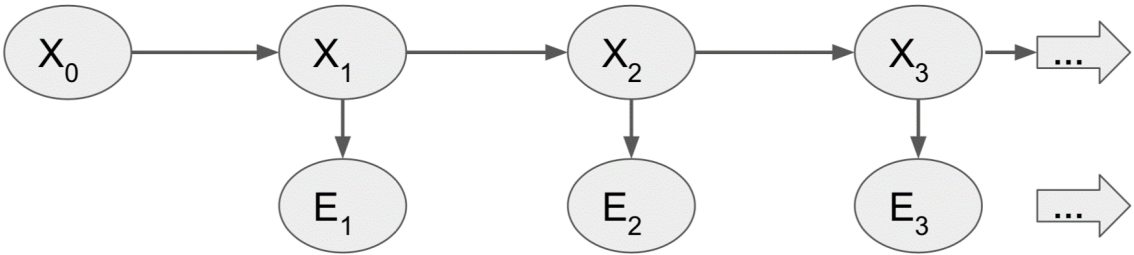
Ties to Our Class

We've seen a lot of models in this class that can make use of this!
In *all* of our Markov Models, we might have time series data to begin with. This allows us to create a model of the conditional probabilities involved:

What is the probability of classifying the day the "does it rain?" on day 1 *as a function* of the weather on day 0? This is a linear model where the response variable is comes from the *features* prior to it in the Bayes Net or HMM.

Similarly, we can do this with an MDP: what kind of underlying probability do we want on our conditional probability *given an action* tables? If the randomness should depend on lots of factors at once - especially continuous ones - we may want a linear model underlying them!

Creating a design matrix



Creating a design matrix:Bayes Nets

Classification Underview

At their roots, many classification problems are about *discretization.* We take processes and measurements that might be continuous, and often turn them into discrete sets of cutoffs. Other examples we've see that might call for a linear model:

1. Is the weather *bad enough* to bring an Umbrella? Or to go for a run?
2. Which set of a discrete set of moves should we take... but trying to apply it to e.g. an autonomous vehicle.
3. Heuristic modeling: sometimes linear or *predictive* models can provide incredibly accurate heuristics (at the possible cost of admissibility/consistency!)

Moving Forward

- Coming up:
  1. Some notebooks Friday
  2. We talk more AI classifiers! NLP, Perceptrons, and then NNs!