Machine Learning (Lecture 4)

UEM/IEM Summer 2018

Representation: Qualities of Good Features

- We've explored ways to map raw data into suitable feature vectors, but that's only part of the work. We must now explore what kinds of values actually make good features within those feature vectors.

Representation: Qualities of Good Features

- **Avoid rarely used discrete feature values**:
  - Good feature values should appear more than 5 or so times in a data set. Doing so enables a model to learn how this feature value relates to the label.
  - That is, having many examples with the same discrete value gives the model a chance to see the feature in different settings, and in turn, determine when it's a good predictor for the label.
  - For example, a house_type feature would likely contain many examples in which its value was victorian:

house_type: victorian

Representation: Qualities of Good Features

- Conversely, if a feature's value appears only once or very rarely, the model can't make predictions based on that feature.
- For example, unique_house_id is a bad feature because each value would be used only once, so the model couldn't learn anything from it:

unique_house_id: 8SK982ZZ1242Z

Representation: Qualities of Good Features

- **Prefer clear and obvious meanings**
  - Each feature should have a clear and obvious meaning to anyone on the project. For example, consider the following good feature for a house's age, which is instantly recognizable as the age in years:

    house_age: 27
  - Conversely, the meaning of the following feature value is pretty much indecipherable to anyone but the engineer who created it:

    house_age: 851472000
  - In some cases, noisy data (rather than bad engineering choices) causes unclear values. For example, the following user_age came from a source that didn't check for appropriate values:

    user_age: 277

Representation: Qualities of Good Features

- **Don't mix "magic" values with actual data**
  - Good floating-point features don't contain peculiar out-of-range discontinuities or "magic" values. For example, suppose a feature holds a floating-point value between 0 and 1. So, values like the following are fine:

    quality_rating: 0.82

    quality_rating: 0.37
  - However, if a user didn't enter a quality_rating, perhaps the data set represented its absence with a magic value like the following:

    quality_rating: -1
  - To work around magic values, convert the feature into two features:
    - One feature holds only quality ratings, never magic values.
    - One feature holds a boolean value indicating whether or not a quality_rating was supplied. Give this boolean feature a name like is_quality_rating_defined.

Representation: Qualities of Good Features

- Account for upstream instability
  - The definition of a feature shouldn't change over time. For example, the following value is useful because the city name*probably* won't change. (Note that we'll still need to convert a string like "br/sao_paulo" to a one-hot vector.)

    city_id: "br/sao_paulo“
  - But gathering a value inferred by another model carries additional costs. Perhaps the value "219" currently represents Sao Paulo, but that representation could easily change on a future run of the other model:

    inferred_city_cluster: "219"

Representation: Cleaning Data

- Apple trees produce some mixture of great fruit and wormy messes.
- Yet the apples in high-end grocery stores display 100% perfect fruit.
- Between orchard and grocery, someone spends significant time removing the bad apples or throwing a little wax on the salvageable ones.
- As an ML engineer, you'll spend enormous amounts of your time tossing out bad examples and cleaning up the salvageable ones.
- Even a few "bad apples" can spoil a large data set.

Representation: Cleaning Data

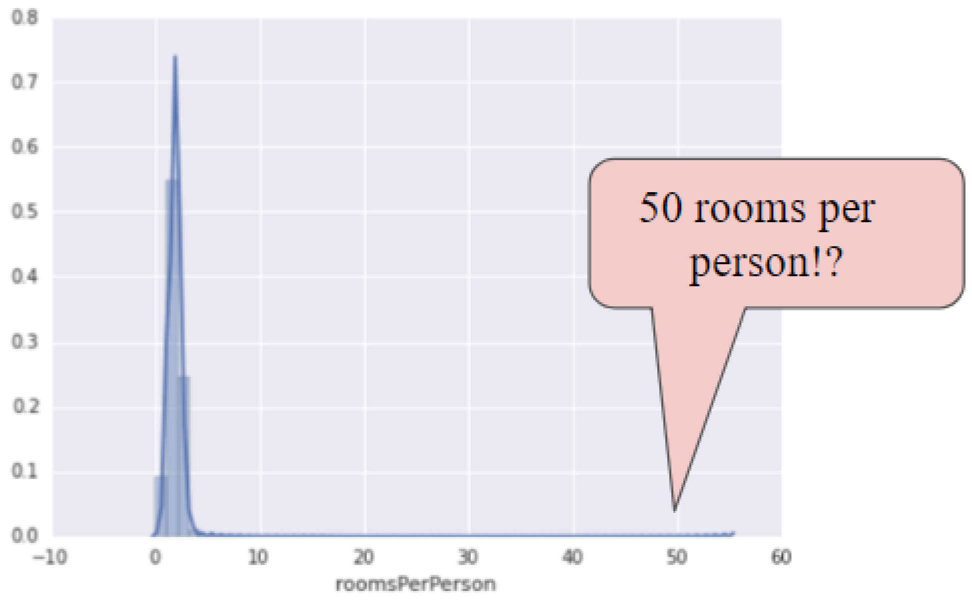
- **Scaling feature values**
  - **Scaling** means converting floating-point feature values from their natural range (for example, 100 to 900) into a standard range (for example, 0 to 1 or -1 to +1).
  - If a feature set consists of only a single feature, then scaling provides little to no practical benefit.
  - If, however, a feature set consists of multiple features, then feature scaling provides the following benefits:
    - Helps gradient descent converge more quickly.
    - Helps avoid the "NaN trap," in which one number in the model becomes a NaN (e.g., when a value exceeds the floating-point precision limit during training), and—due to math operations—every other number in the model also eventually becomes a NaN.
    - Helps the model learn appropriate weights for each feature. Without feature scaling, the model will pay too much attention to the features having a wider range.

Representation: Cleaning Data

- **Scaling feature values** (cont’d)
  - You don't have to give every floating-point feature exactly the same scale.
  - Nothing terrible will happen if Feature A is scaled from -1 to +1 while Feature B is scaled from -3 to +3.
  - However, your model will react poorly if Feature B is scaled from 5000 to 100000.

Representation: Cleaning Data

- **Handling extreme outliers**
  - The following plot represents a feature called roomsPerPerson from the California Housing data set. The value of roomsPerPerson was calculated by dividing the total number of rooms for an area by the population for that area. The plot shows that the vast majority of areas in California have one or two rooms per person. But take a look along the x-axis.

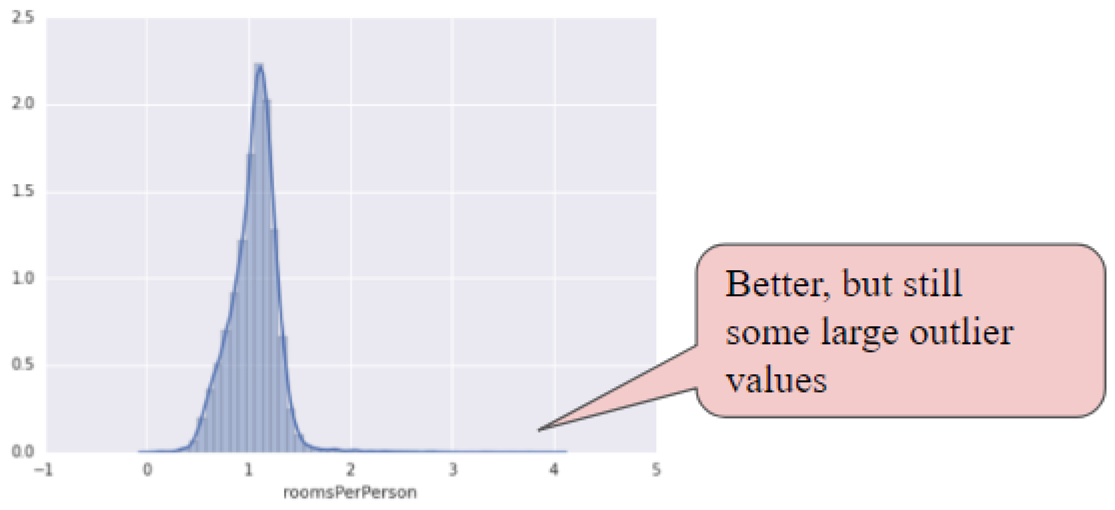


roomsPerPerson = totalRooms / population

Figure 1. A verrrrry lonnnnnnng tail.

Representation: Cleaning Data

- **Handling extreme outliers** (cont’s)
  - How could we minimize the influence of those extreme outliers? Well, one way would be to take the log of every value:

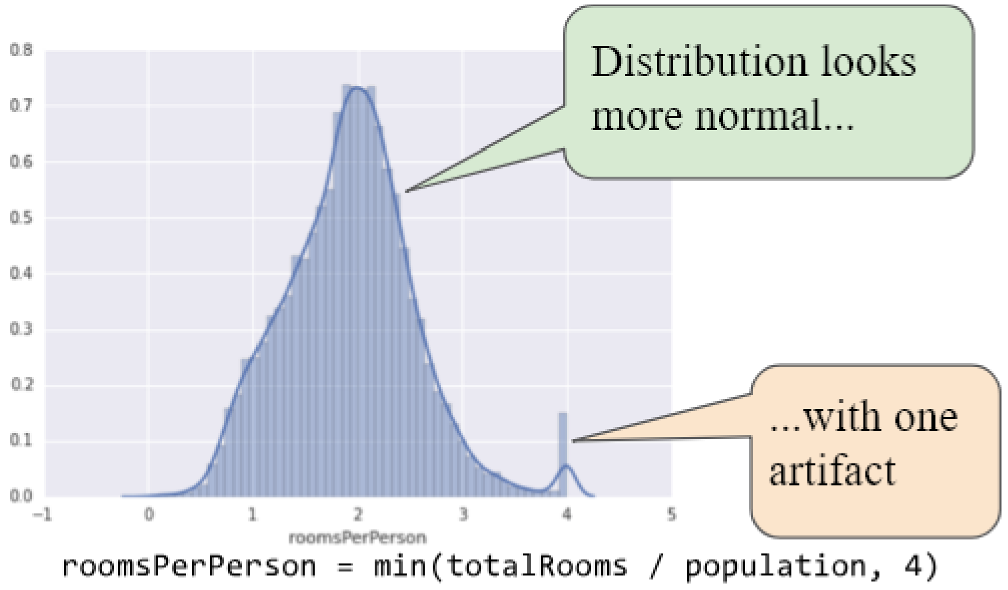


```
roomsPerPerson = log((totalRooms / population) + 1)
```

Figure 2. Logarithmic scaling still leaves a tail.

Representation: Cleaning Data

- **Handling extreme outliers** (cont’s)
  - Log scaling does a slightly better job, but there's still a significant tail of outlier values. Let's pick yet another approach. What if we simply "cap" or "clip" the maximum value of roomsPerPerson at an arbitrary value, say 4.0?



```
roomsPerPerson = min(totalRooms / population, 4)
```

Figure 3. Clipping feature values at 4.0

Representation: Cleaning Data

- Clipping the feature value at 4.0 doesn't mean that we ignore all values greater than 4.0.
- Rather, it means that all values that were greater than 4.0 now become 4.0.
- This explains the funny hill at 4.0. Despite that hill, the scaled feature set is now more useful than the original data.

Representation: Cleaning Data

- **Binning**
  - The following plot shows the relative prevalence of houses at different latitudes in California. Notice the clustering—Los Angeles is about at latitude 34 and San Francisco is roughly at latitude 38.

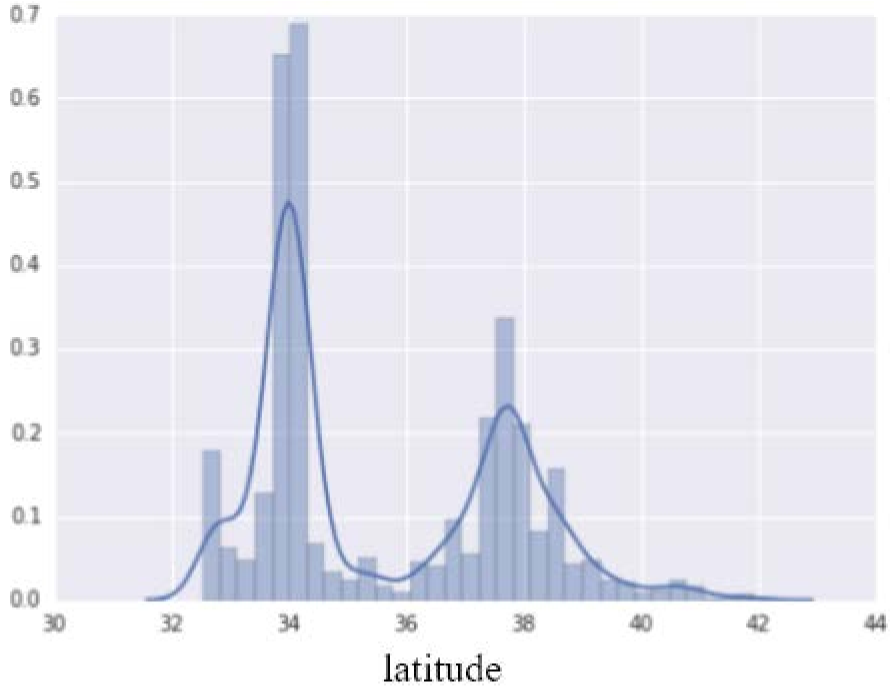


Figure 4. Houses per latitude.

Representation: Cleaning Data

- **Binning** (cont'd)
  - In the data set, latitude is a floating-point value.
  - However, it doesn't make sense to represent latitude as a floating-point feature in our model.
  - That's because no linear relationship exists between latitude and housing values.
  - For example, houses in latitude 35 are not 35/34 more expensive (or less expensive) than houses at latitude 34.
  - And yet, individual latitudes probably are a pretty good predictor of house values.

Representation: Cleaning Data

- **Binning** (cont'd)
  - To make latitude a helpful predictor, let's divide latitudes into "bins" as suggested by the following figure:

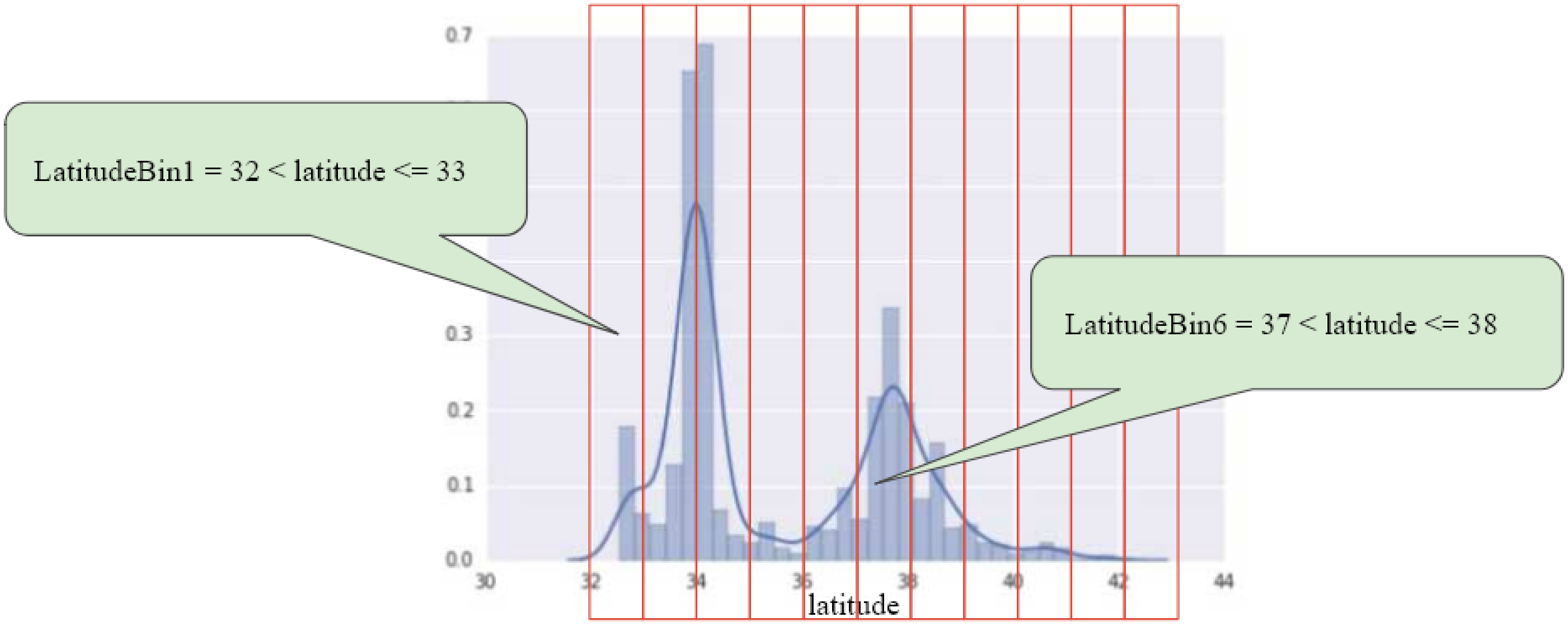


Figure 5. Binning values

Representation: Cleaning Data

- **Binning** (cont'd)
  - Instead of having one floating-point feature, we now have 11 distinct boolean features (LatitudeBin1, LatitudeBin2, ..., LatitudeBin11).
  - Having 11 separate features is somewhat inelegant, so let's unite them into a single 11-element vector.
  - Doing so will enable us to represent latitude 37.4 as follows:

    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
  - Thanks to binning, our model can now learn completely different weights for each latitude.

Representation: Cleaning Data

- **Scrubbing**
  - Until now, we've assumed that all the data used for training and testing was trustworthy.
  - In real-life, many examples in data sets are unreliable due to one or more of the following:
    - **Omitted values.** For instance, a person forgot to enter a value for a house's age.
    - **Duplicate examples.** For example, a server mistakenly uploaded the same logs twice.
    - **Bad labels.** For instance, a person mislabeled a picture of an oak tree as a maple.
    - **Bad feature values.** For example, someone typed in an extra digit, or a thermometer was left out in the sun.

Representation: Cleaning Data

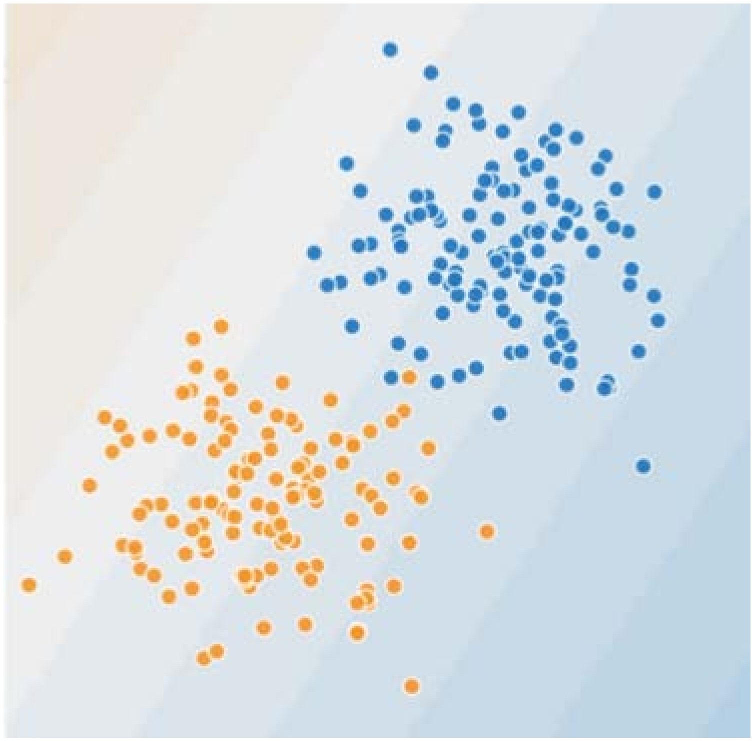
- **Scrubbing** (cont’d)
  - Once detected, you typically "fix" bad examples by removing them from the data set.
  - To detect omitted values or duplicated examples, you can write a simple program. Detecting bad feature values or labels can be far trickier.
  - In addition to detecting bad individual examples, you must also detect bad data in the aggregate. Histograms are a great mechanism for visualizing your data in the aggregate. In addition, getting statistics like the following can help:
    - Maximum and minimum
    - Mean and median
    - Standard deviation
  - Consider generating lists of the most common values for discrete features.
  - For example, do the number of examples with country:uk match the number you expect. Should language:jp really be the most common language in your data set?

Feature Crosses

- A **feature cross** is a **synthetic feature** formed by multiplying (crossing) two or more features. Crossing combinations of features can provide predictive abilities beyond what those features can provide individually.

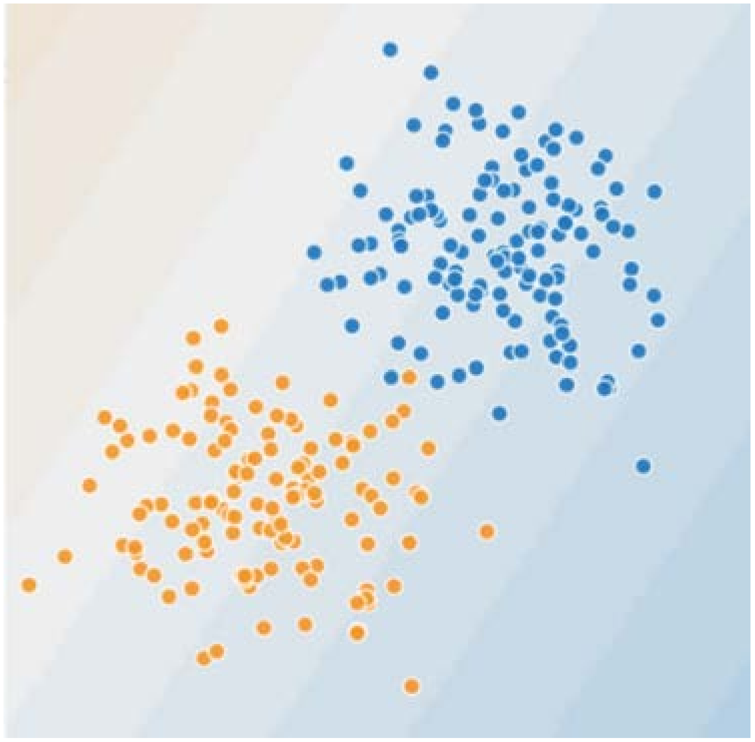
Feature Crosses: Encoding Nonlinearity

- Is this a linear problem?



Feature Crosses: Encoding Nonlinearity

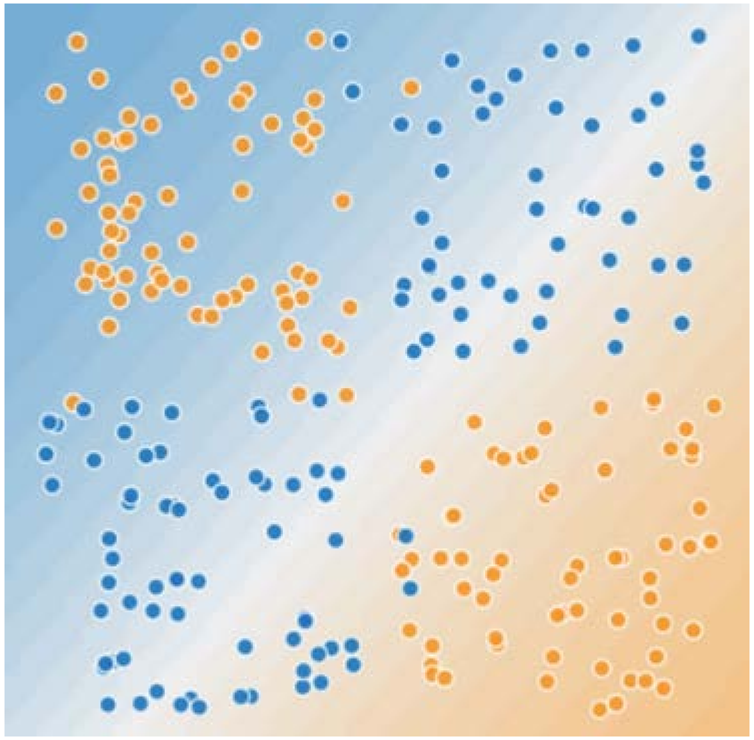
- Is this a linear problem?



  - Can you draw a line that neatly separates the sick trees from the healthy trees? Sure. This is a linear problem. The line won't be perfect. A sick tree or two might be on the "healthy" side, but your line will be a good predictor.

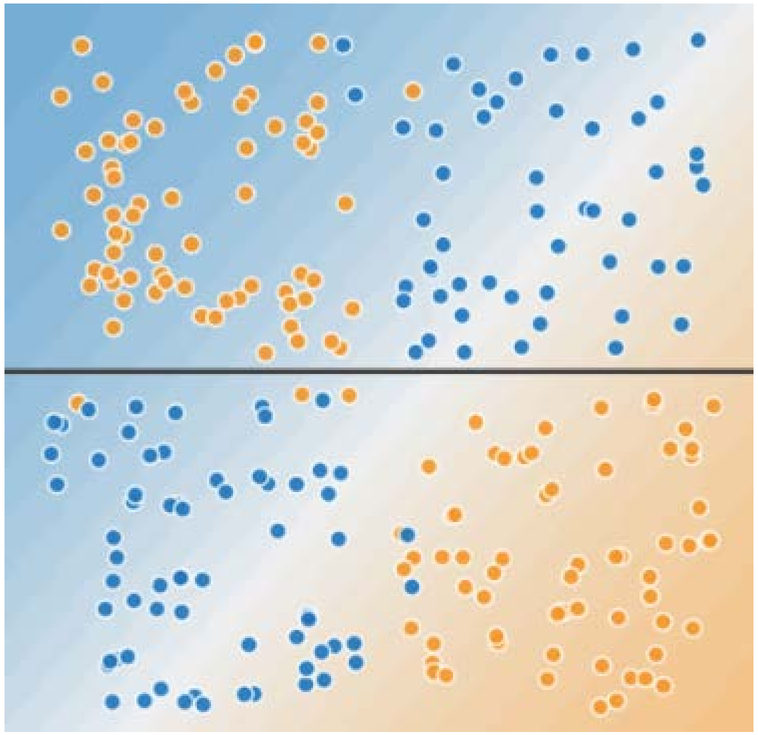
Feature Crosses: Encoding Nonlinearity

- Is this a linear problem?



Feature Crosses: Encoding Nonlinearity

- Is this a linear problem?



  - Can you draw a single straight line that neatly separates the sick trees from the healthy trees? No, you can't. This is a nonlinear problem. Any line you draw will be a poor predictor of tree health.

Feature Crosses: Encoding Nonlinearity

- To solve the nonlinear problem, create a feature cross.
- A **feature cross** is a synthetic feature that encodes nonlinearity in the feature space by multiplying two or more input features together. (The term cross comes from cross product.)
- Let's create a feature cross named x_3 by crossing x_1 and x_2:

$$x_3=x_1x_2$$

- We treat this newly minted x3 feature cross just like any other feature. The linear formula becomes:

$$y=b+w_1x_1+w_2x_2+w_3x_3$$

- A linear algorithm can learn a weight for w_3 just as it would for w_1 and w_2. In other words, although w_3 encodes nonlinear information, you don't need to change how the linear model trains to determine the value of w_3.

Feature Crosses: Encoding Nonlinearity

- Kinds of feature crosses
  - We can create many different kinds of feature crosses. For example:
    - [A X B]: a feature cross formed by multiplying the values of two features.
    - [A x B x C x D x E]: a feature cross formed by multiplying the values of five features.
    - [A x A]: a feature cross formed by squaring a single feature.
  - Thanks to stochastic gradient descent, linear models can be trained efficiently.
  - Consequently, supplementing scaled linear models with feature crosses has traditionally been an efficient way to train on massive-scale data sets.

Feature Crosses: Crossing One-Hot Vectors

- So far, we've focused on feature-crossing two individual floating-point features.
- In practice, machine learning models seldom cross continuous features.
- However, machine learning models do frequently cross one-hot feature vectors.
- Think of feature crosses of one-hot feature vectors as logical conjunctions.

Feature Crosses: Crossing One-Hot Vectors

- For example, suppose we have two features: country and language.
- A one-hot encoding of each generates vectors with binary features that can be interpreted as country=USA, country=France or language=English, language=Spanish.
- Then, if you do a feature cross of these one-hot encodings, you get binary features that can be interpreted as logical conjunctions, such as:

  country:usa AND language:spanish

Feature Crosses: Crossing One-Hot Vectors

- As another example, suppose you bin latitude and longitude, producing separate one-hot five-element feature vectors. For instance, a given latitude and longitude could be represented as follows:

  binned_latitude = [0, 0, 0, 1, 0]

  binned_longitude = [0, 1, 0, 0, 0]
- This feature cross is a 25-element one-hot vector (24 zeroes and 1 one).
- The single 1 in the cross identifies a particular conjunction of latitude and longitude. Your model can then learn particular associations about that conjunction.

Feature Crosses: Crossing One-Hot Vectors

- Suppose we bin latitude and longitude much more coarsely, as follows:

```
binned_latitude(lat) = [
  0  < lat <= 10
  10 < lat <= 20
  20 < lat <= 30
]

binned_longitude(lon) = [
  0  < lon <= 15
  15 < lon <= 30
]
```

- Creating a feature cross of those coarse bins leads to synthetic feature having the following meanings:

```
binned_latitude_X_longitude(lat, lon) = [
  0  < lat <= 10 AND 0  < lon <= 15
  0  < lat <= 10 AND 15 < lon <= 30
  10 < lat <= 20 AND 0  < lon <= 15
  10 < lat <= 20 AND 15 < lon <= 30
  20 < lat <= 30 AND 0  < lon <= 15
  20 < lat <= 30 AND 15 < lon <= 30
]
```

Feature Crosses: Crossing One-Hot Vectors

- Now suppose our model needs to predict how satisfied dog owners will be with dogs based on two features:
  - Behavior type (barking, crying, snuggling, etc.)
  - Time of day
- If we build a feature cross from both these features:

```
[behavior type X time of day]
```

then we'll end up with vastly more predictive ability than either feature on its own.

Feature Crosses: Crossing One-Hot Vectors

- For example, if a dog cries (happily) at 5:00 pm when the owner returns from work will likely be a great positive predictor of owner satisfaction.
- Crying (miserably, perhaps) at 3:00 am when the owner was sleeping soundly will likely be a strong negative predictor of owner satisfaction.
- Linear learners scale well to massive data.
- Using feature crosses on massive data sets is one efficient strategy for learning highly complex models.
- Neural networks provide another strategy.

Feature Crosses

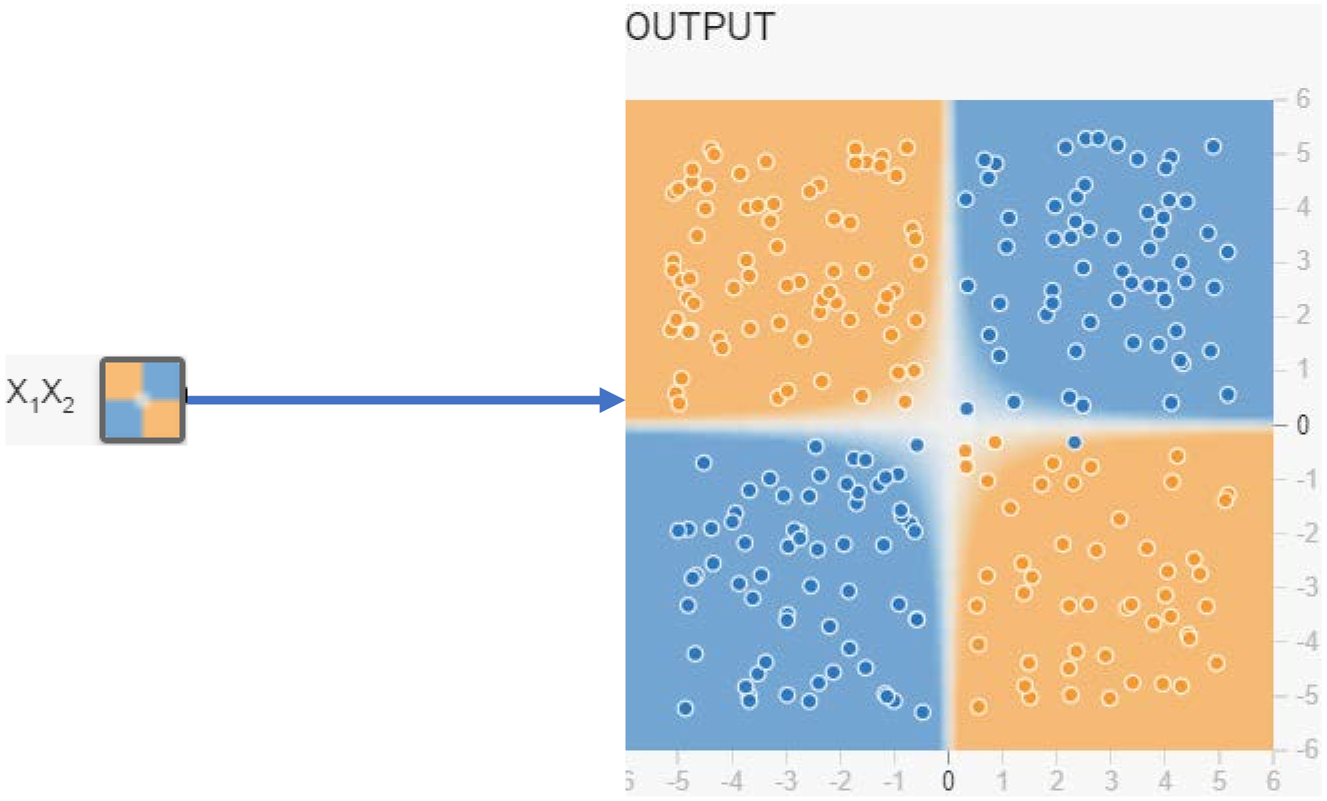
- Try to create a model that separates the blue dots from the orange dots by *manually* changing the weights of the following three input features:
  - x_1
  - x_2
  - $x_1 x_2$ (a feature cross)

Feature Crosses

- Try to create a model that separates the blue dots from the orange dots by *manually* changing the weights of the following three input features:
  - x_1
  - x_2
  - $x_1 x_2$ (a feature cross)

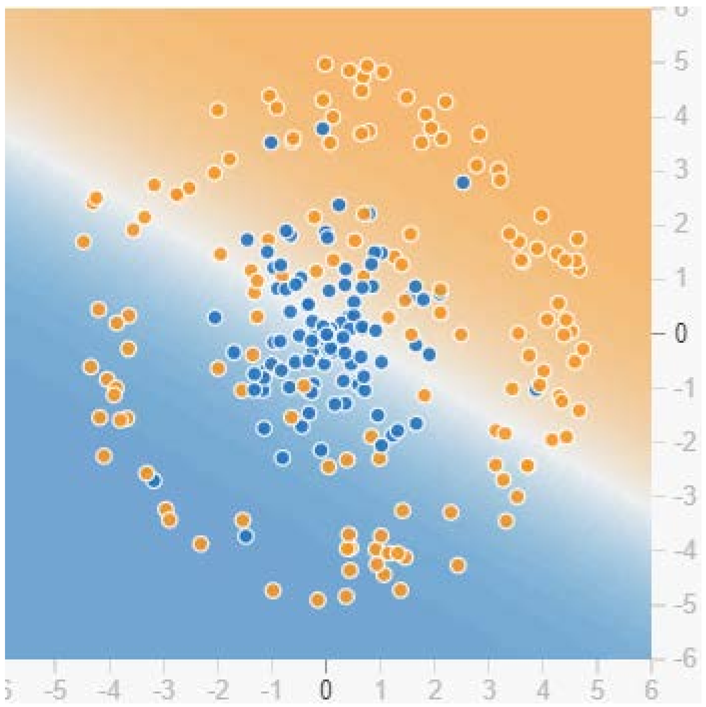
  - $w_1 = 0$
  - $w_2 = 0$
  - $x_1 x_2 = 1$ (or any positive value)

Feature Crosses



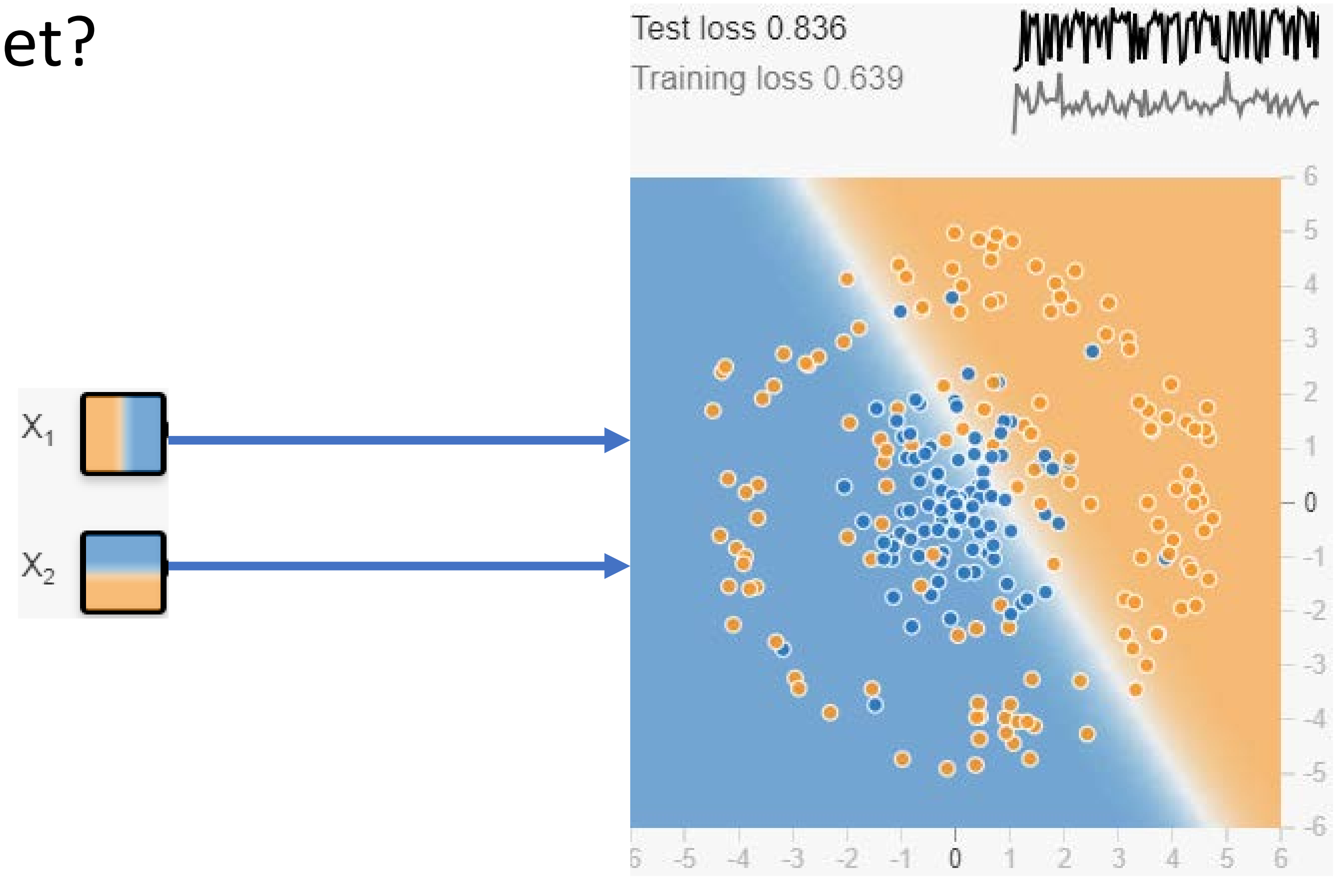
Feature Crosses

- Now let's try with some advanced feature cross combinations. The data set looks a bit like a noisy bullseye from a game of darts, with the blue dots in the middle and the orange dots in an outer ring.



Feature Crosses

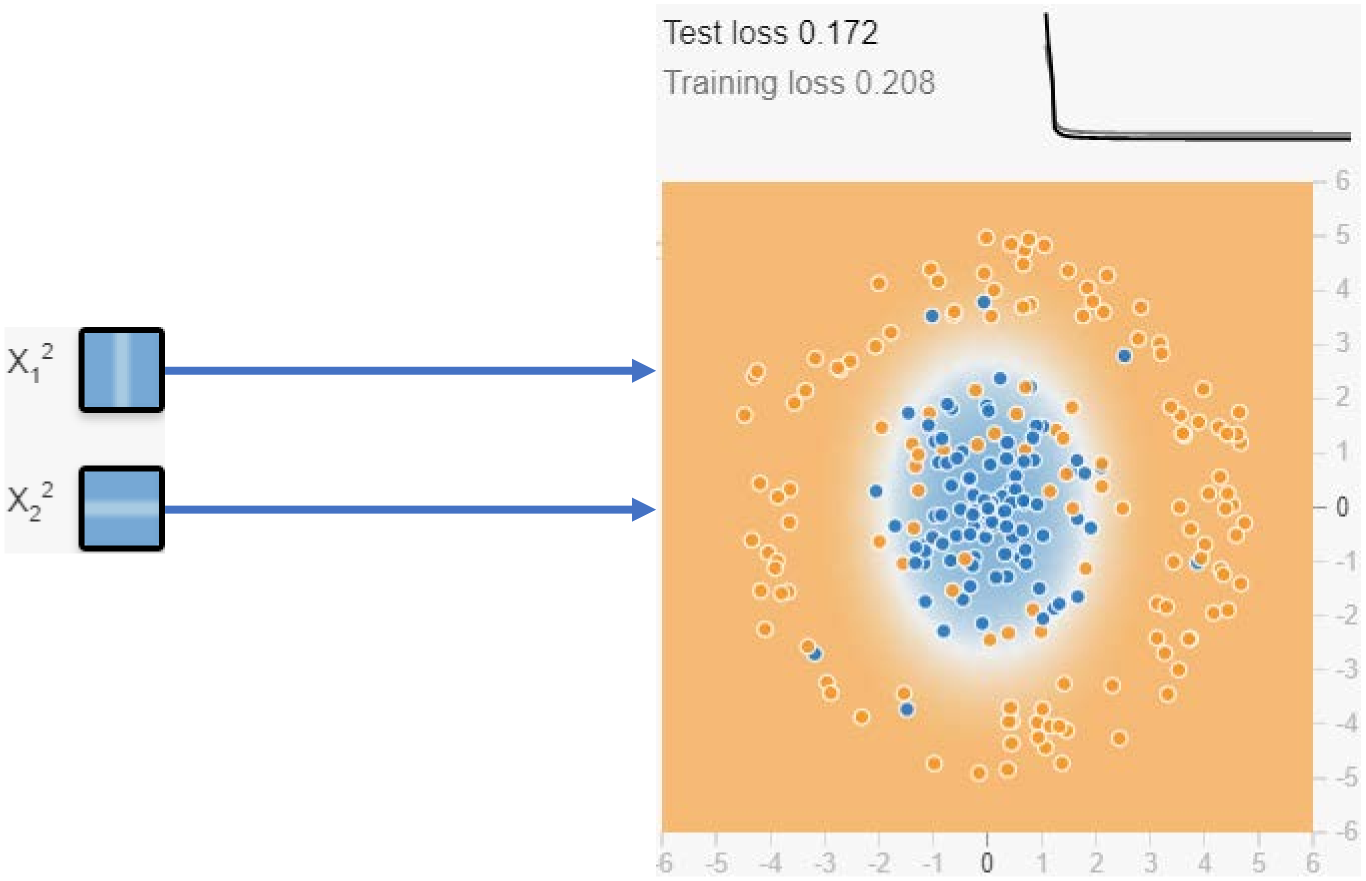
- Run this linear model as given.
- Can a linear model produce effective results for this data set?



No. A linear model cannot effectively model this data set. Reducing the learning rate reduces loss, but loss still converges at an unacceptably high value.

Feature Crosses

- Using both x12 and x22 as feature crosses. (Adding x1x2 as a feature cross doesn't appear to help.)
- Reducing the Learning rate, perhaps to 0.001.



Quiz

- Different cities in California have markedly different housing prices. Suppose you must create a model to predict housing prices. Which of the following sets of features or feature crosses could learn city-specific relationships between roomsPerPerson and housing price?

a) One feature cross: [latitude X longitude X roomsPerPerson]

b) Three separate binned features: [binned latitude], [binned longitude], [binned roomsPerPerson]

c) Two feature crosses: [binned latitude X binned roomsPerPerson] and [binned longitude X binned roomsPerPerson]

d) One feature cross: [binned latitude X binned longitude X binned roomsPerPerson]

Quiz

- Different cities in California have markedly different housing prices. Suppose you must create a model to predict housing prices. Which of the following sets of features or feature crosses could learn city-specific relationships between roomsPerPerson and housing price?

a) One feature cross: [latitude X longitude X roomsPerPerson]

b) Three separate binned features: [binned latitude], [binned longitude], [binned roomsPerPerson]

c) Two feature crosses: [binned latitude X binned roomsPerPerson] and [binned longitude X binned roomsPerPerson]

d) One feature cross: [binned latitude X binned longitude X binned roomsPerPerson]

Reference

- This lecture note has been developed based on the machine learning crash course at Google, which is under *Creative Commons Attribution 3.0 License*.