Feature Crosses

Feature Engineering
Machine Learning on Google Cloud Platform

Lak Lakshmanan

Learn how to...

Learn how to...

Recognize where feature crosses are a powerful way to help machines learn

Learn how to...

Recognize where feature crosses are a powerful way to help machines learn

Implement feature crosses in TensorFlow

Learn how to...

Recognize where feature crosses are a powerful way to help machines learn

Implement feature crosses in TensorFlow

Incorporate feature creation as part of your ML pipeline

Learn how to...

Recognize where feature crosses are a powerful way to help machines learn

Implement feature crosses in TensorFlow

Incorporate feature creation as part of your ML pipeline

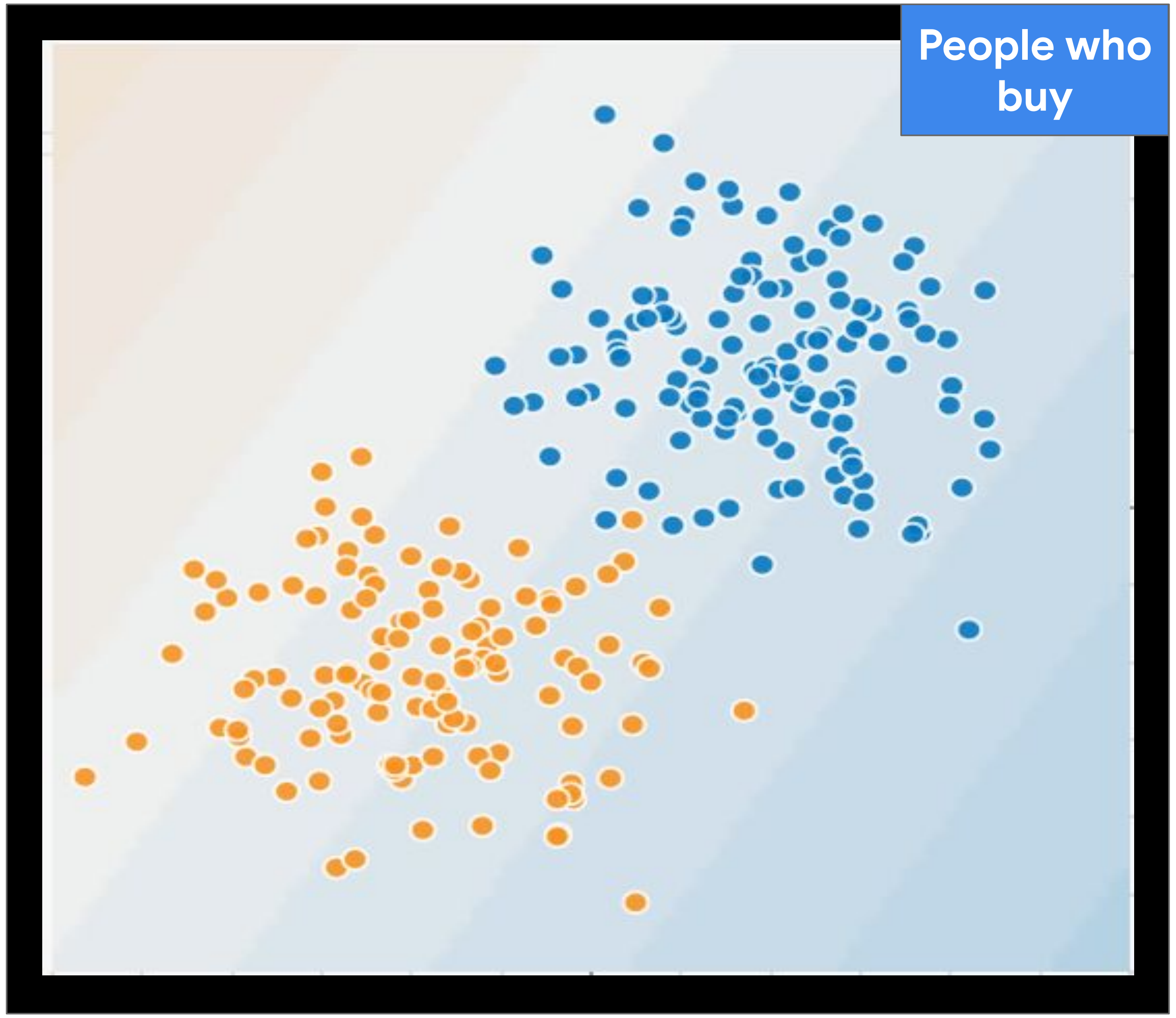
Improve the taxifare model using feature crosses

Why feature crosses?

Lak Lakshmanan

Can you draw a line that separates these two classes?

Can you draw a line that separates these two classes?



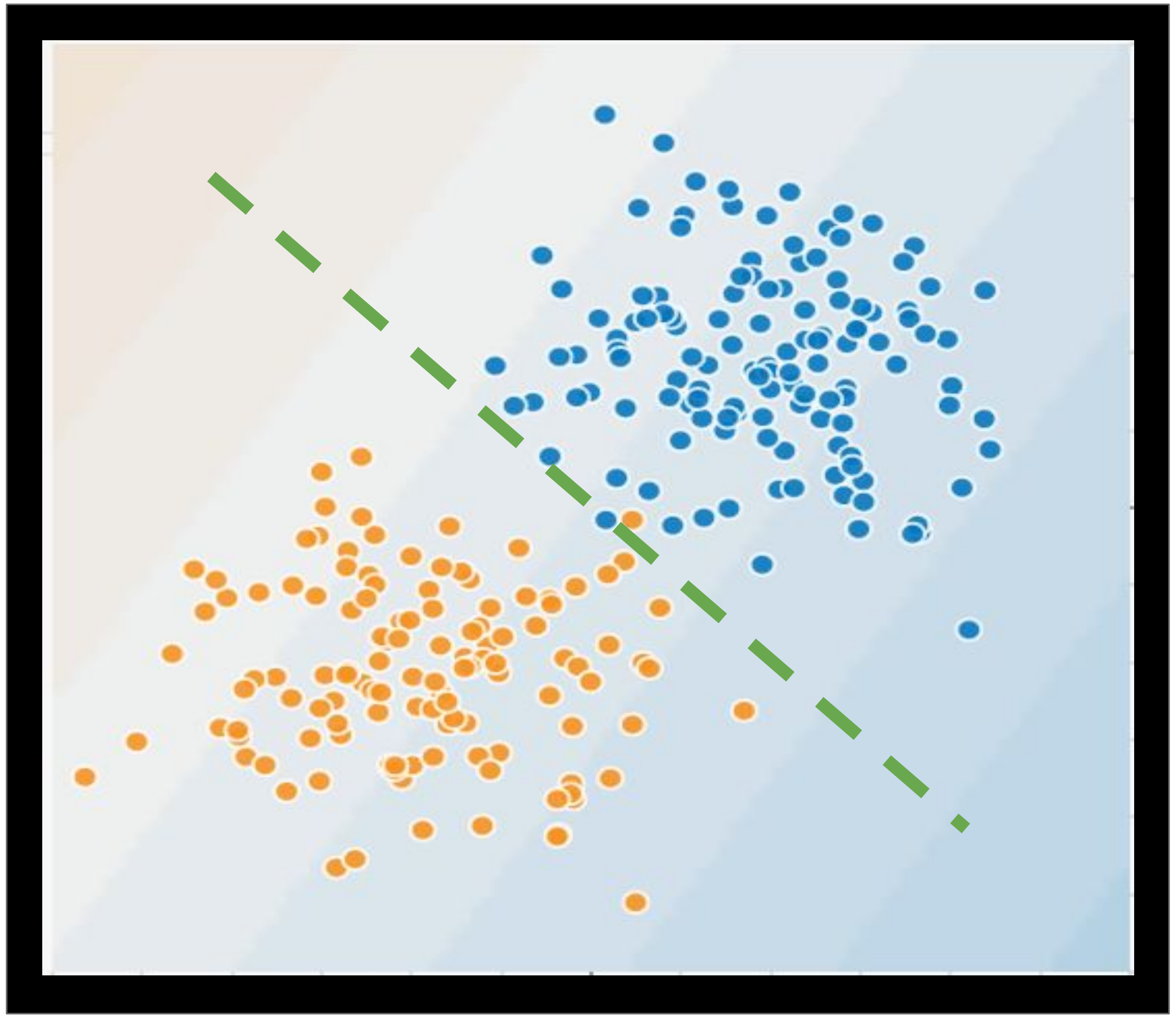
Can you draw a line that separates these two classes?

Can you draw a line that separates these two classes?

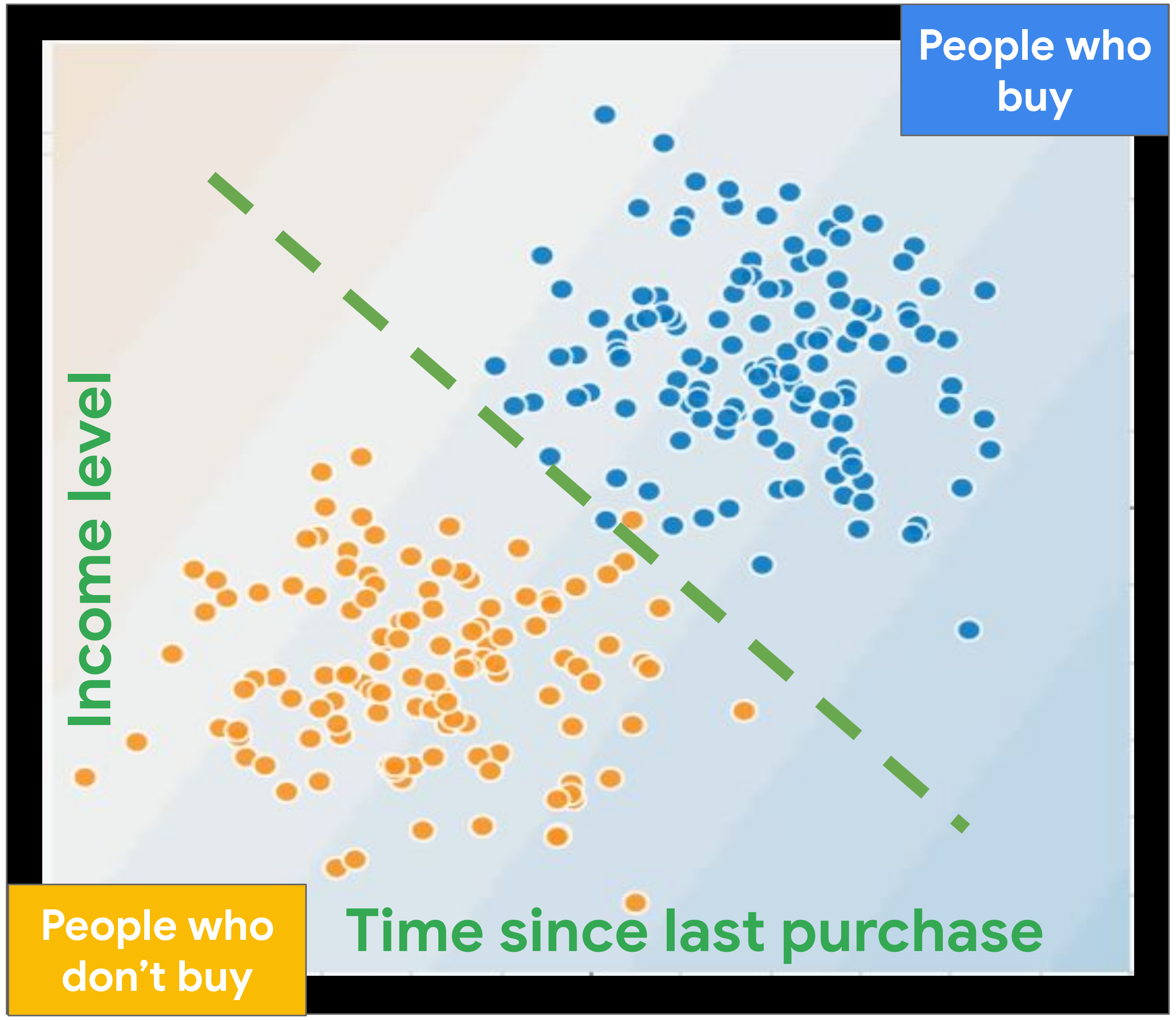
Can you draw a line that separates these two classes?

Can you draw a line that separates these two classes?

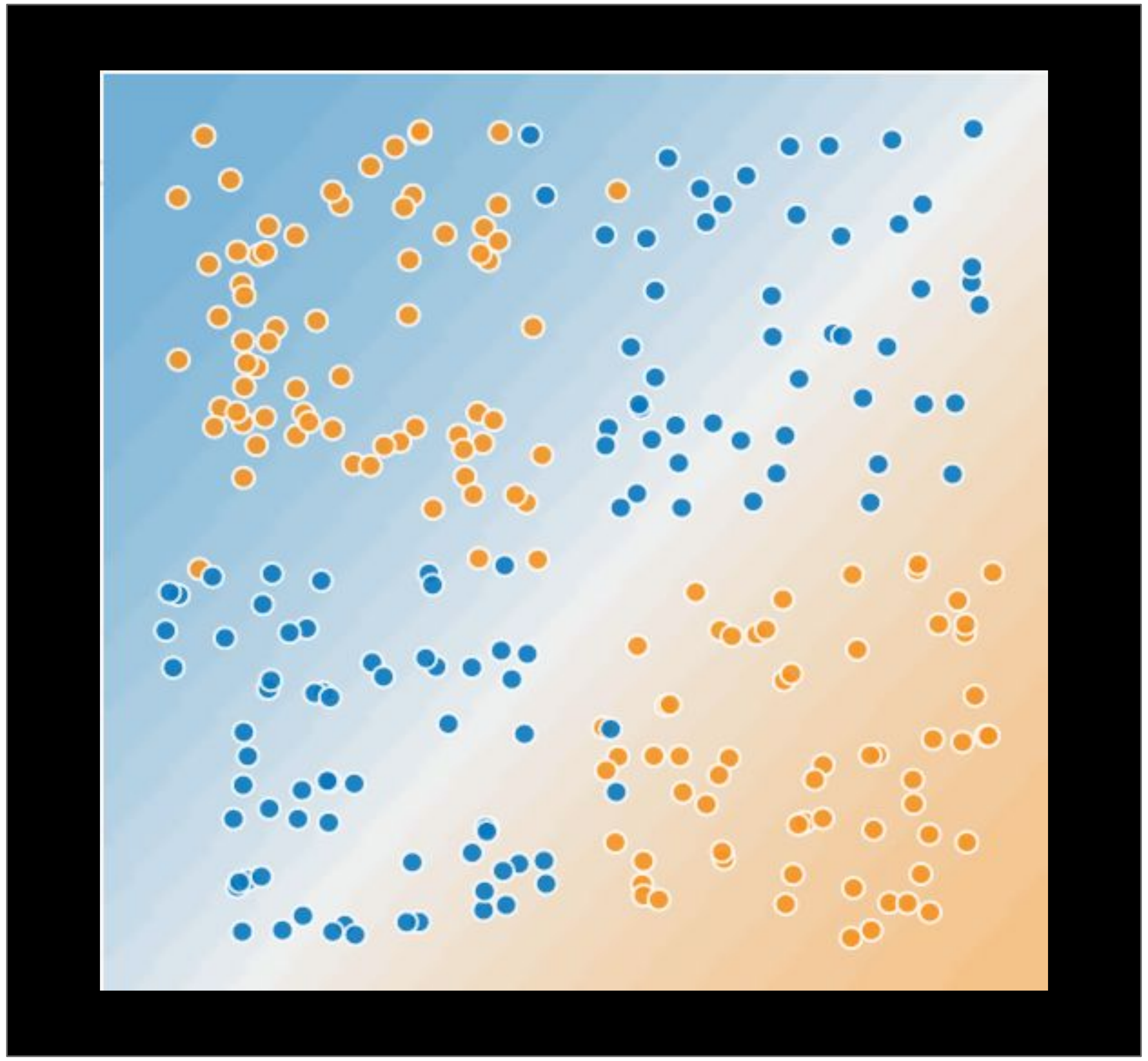
Can you draw a line that separates these two classes?



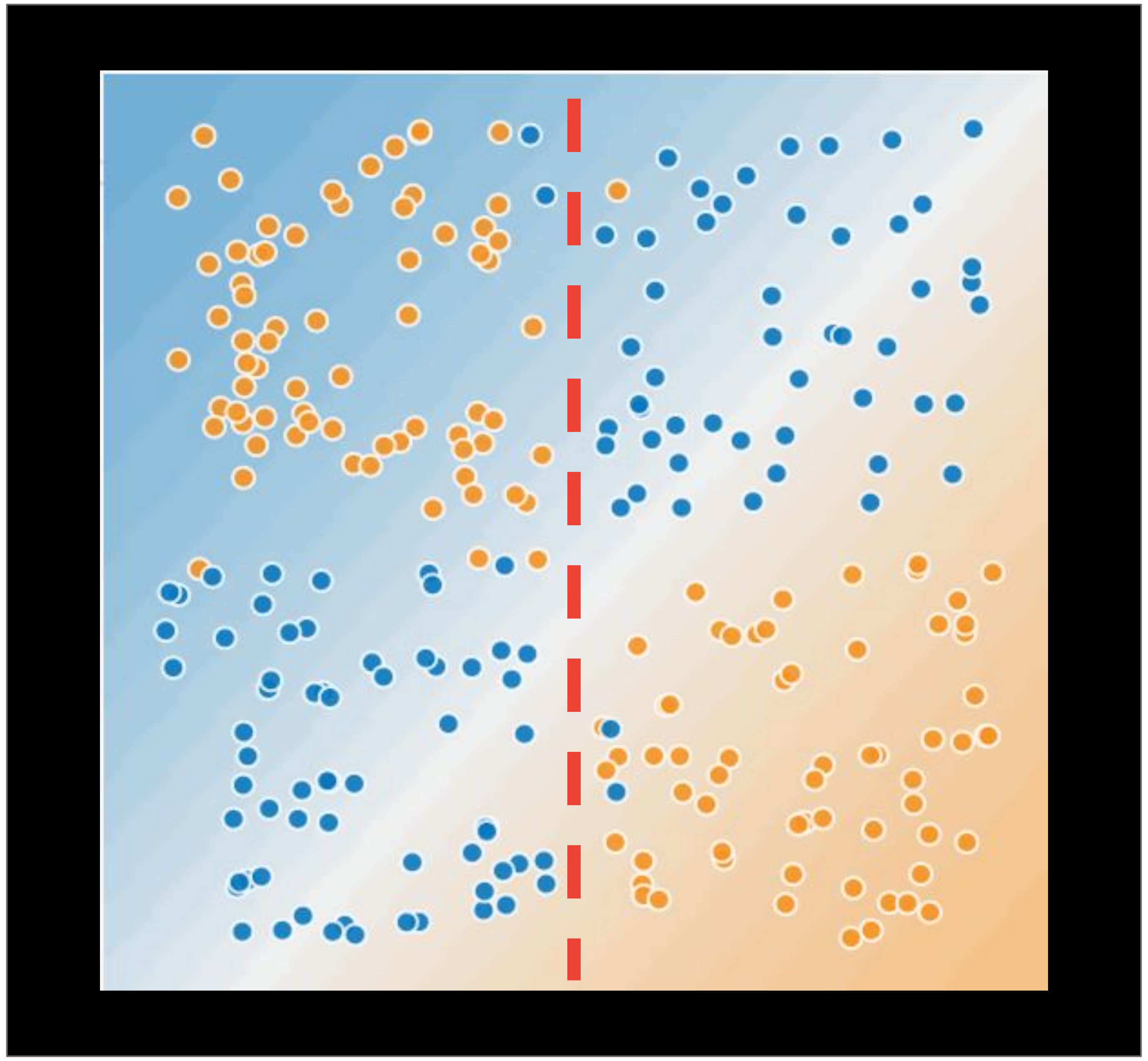
This is a linear problem



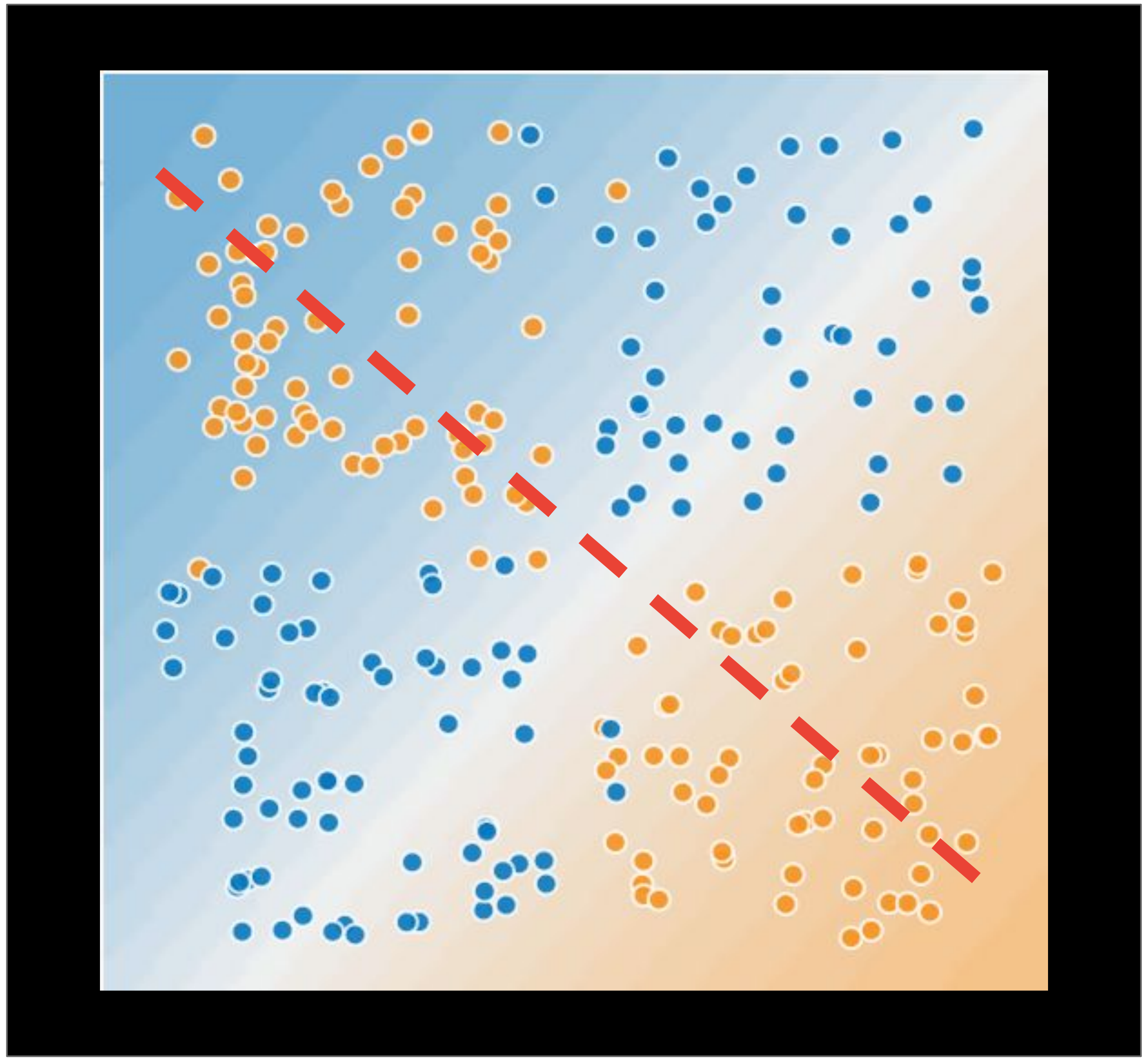
How about this?
Is it a linear problem?



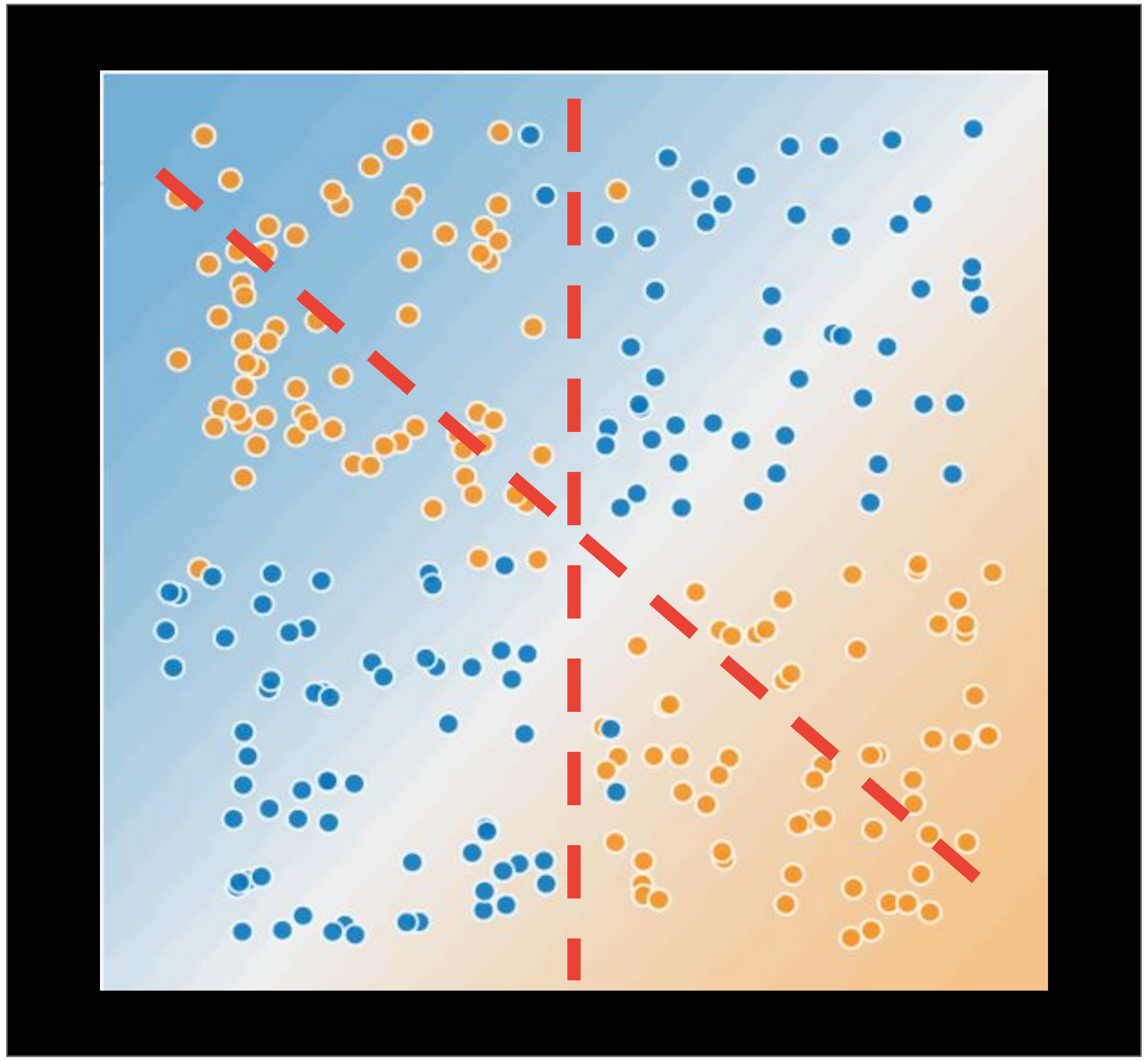
How about this?
Is it a linear problem?

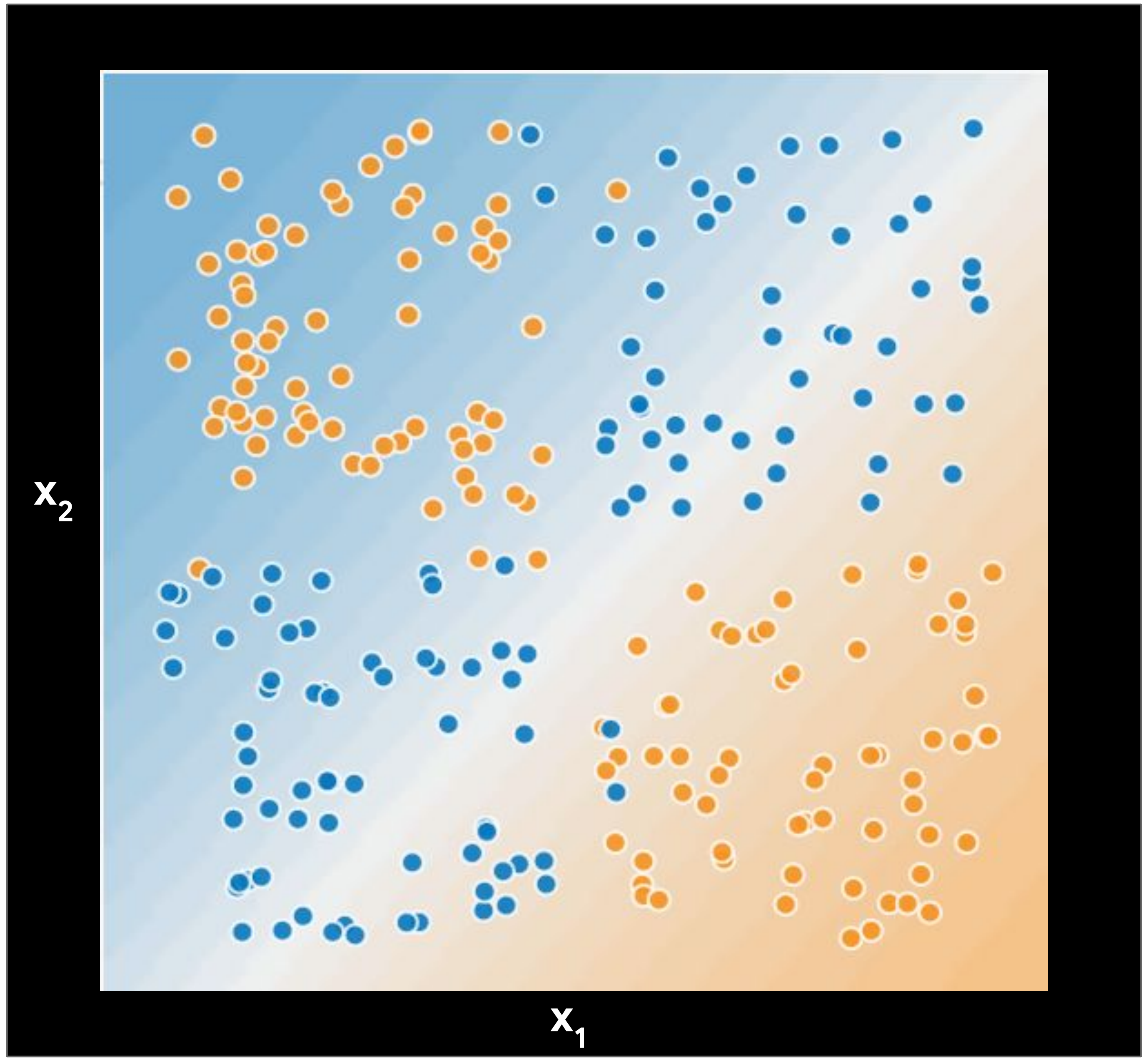


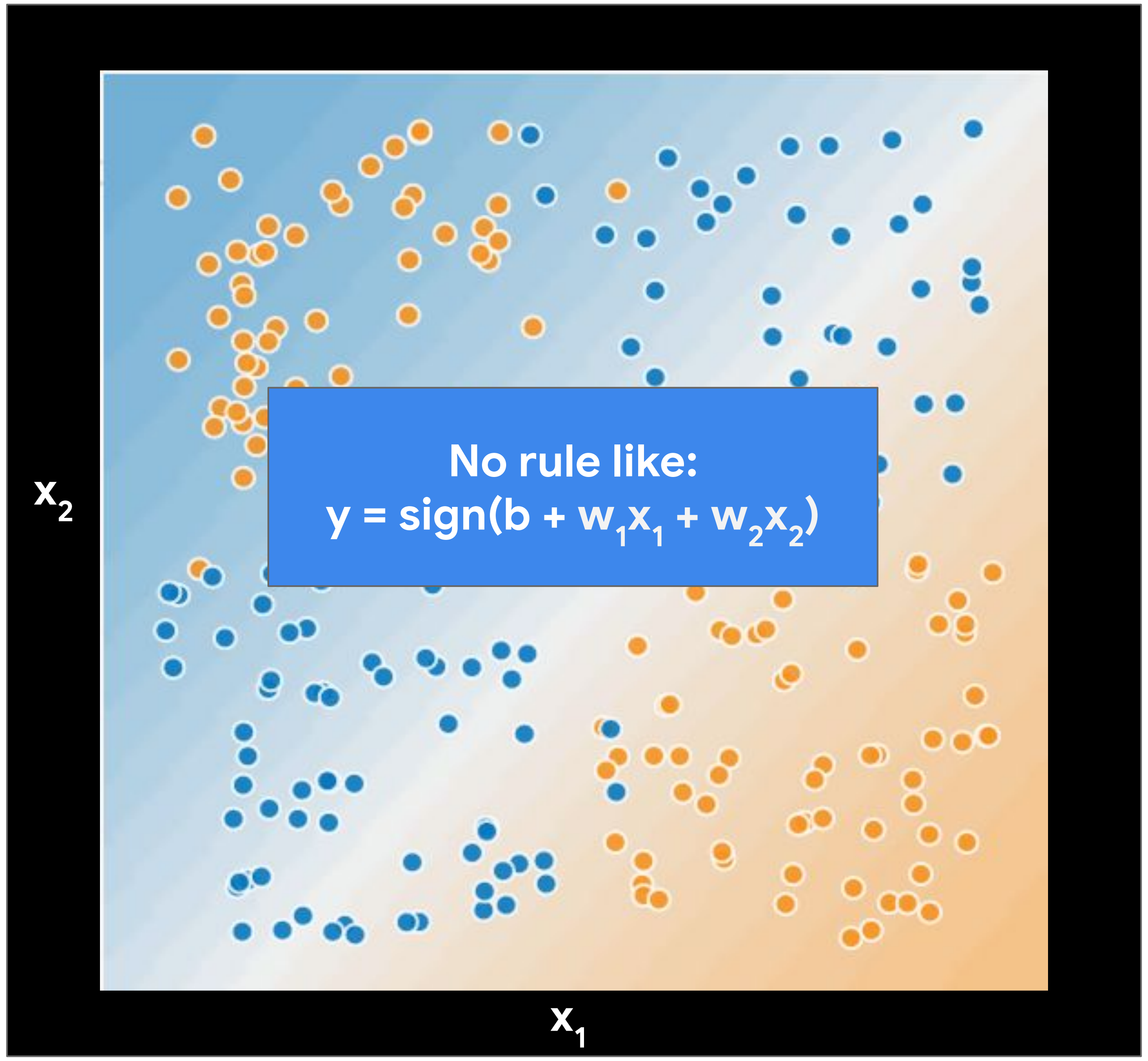
How about this?
Is it a linear problem?

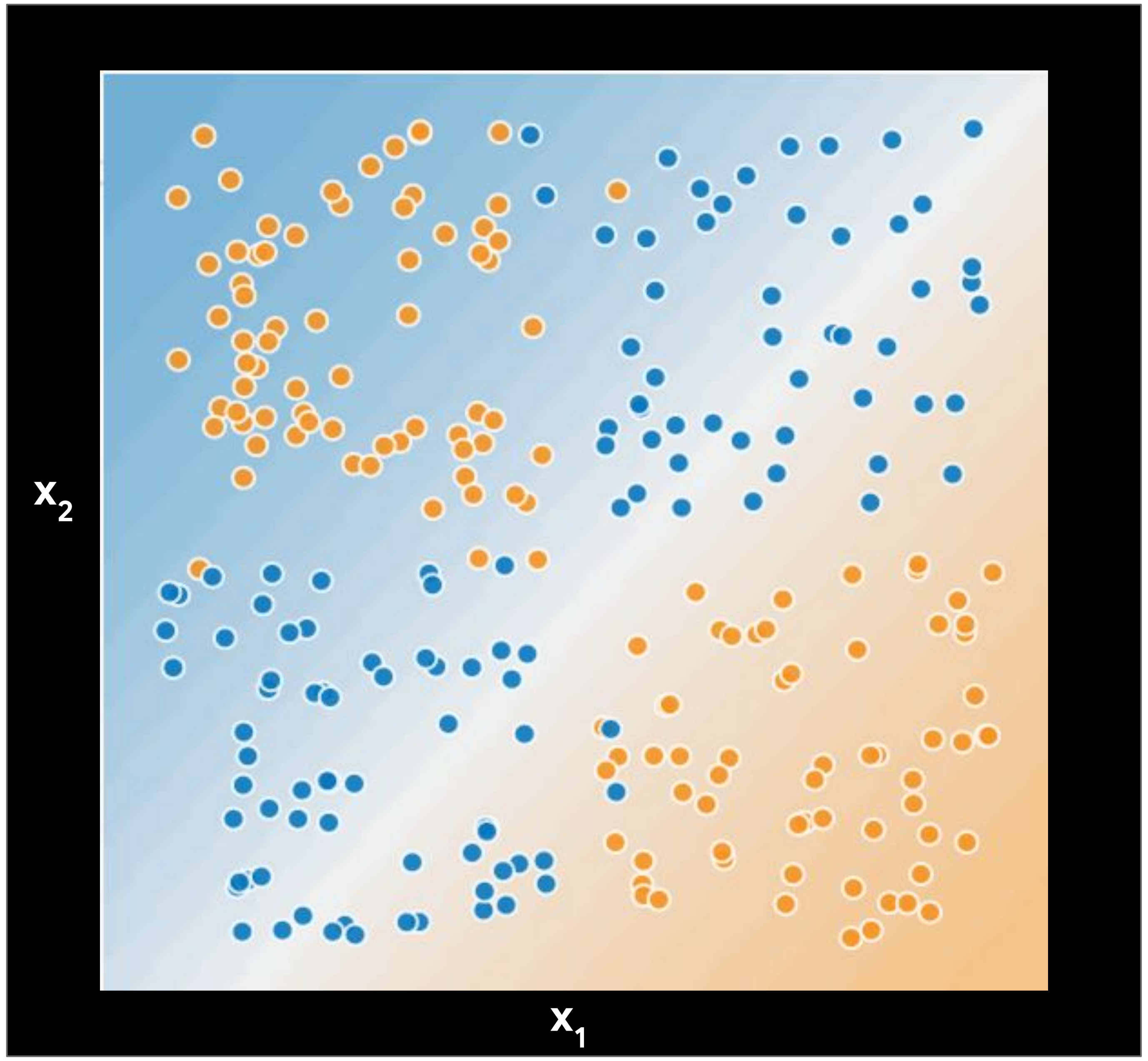


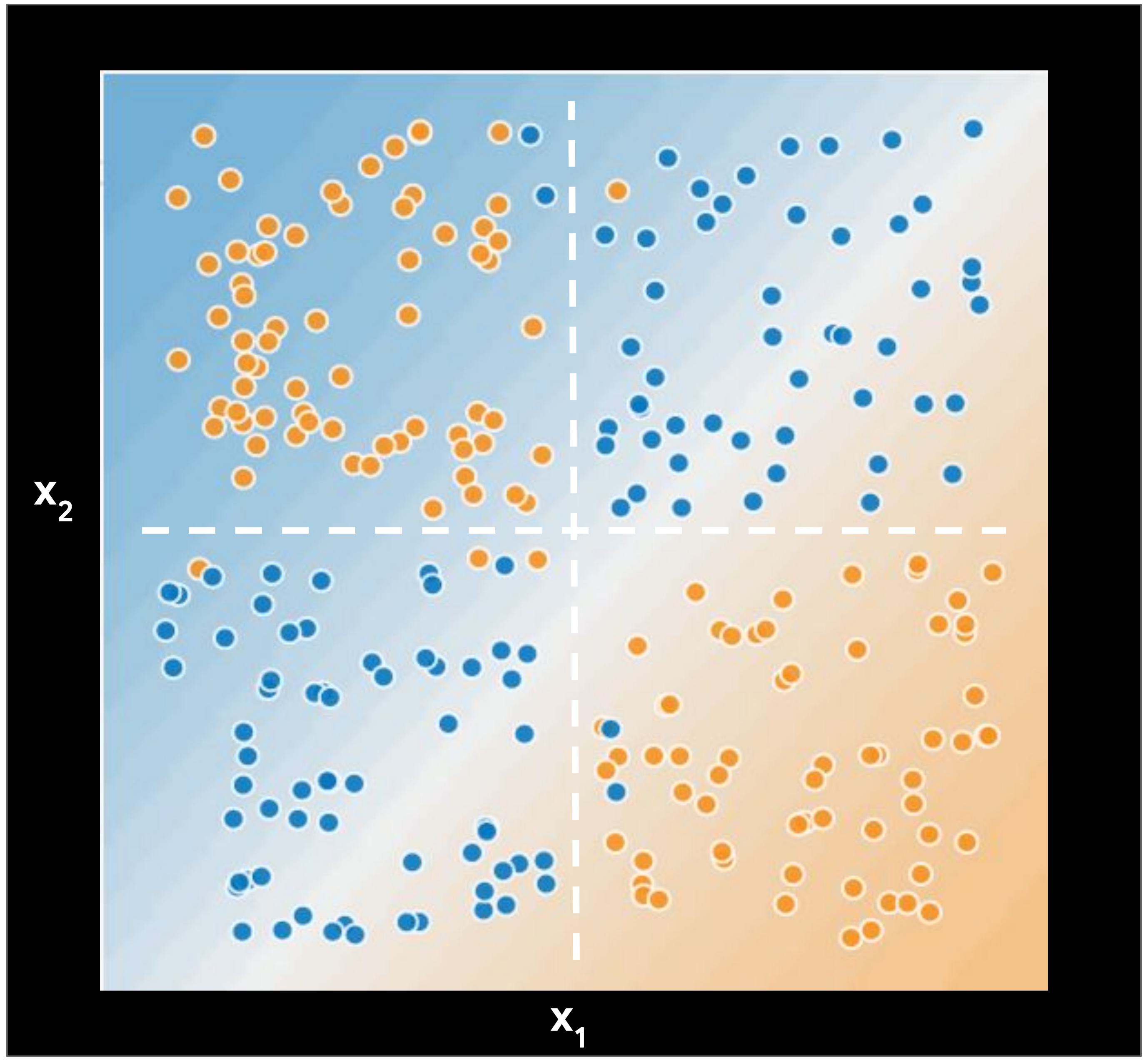
How about this?
Is it a linear problem?

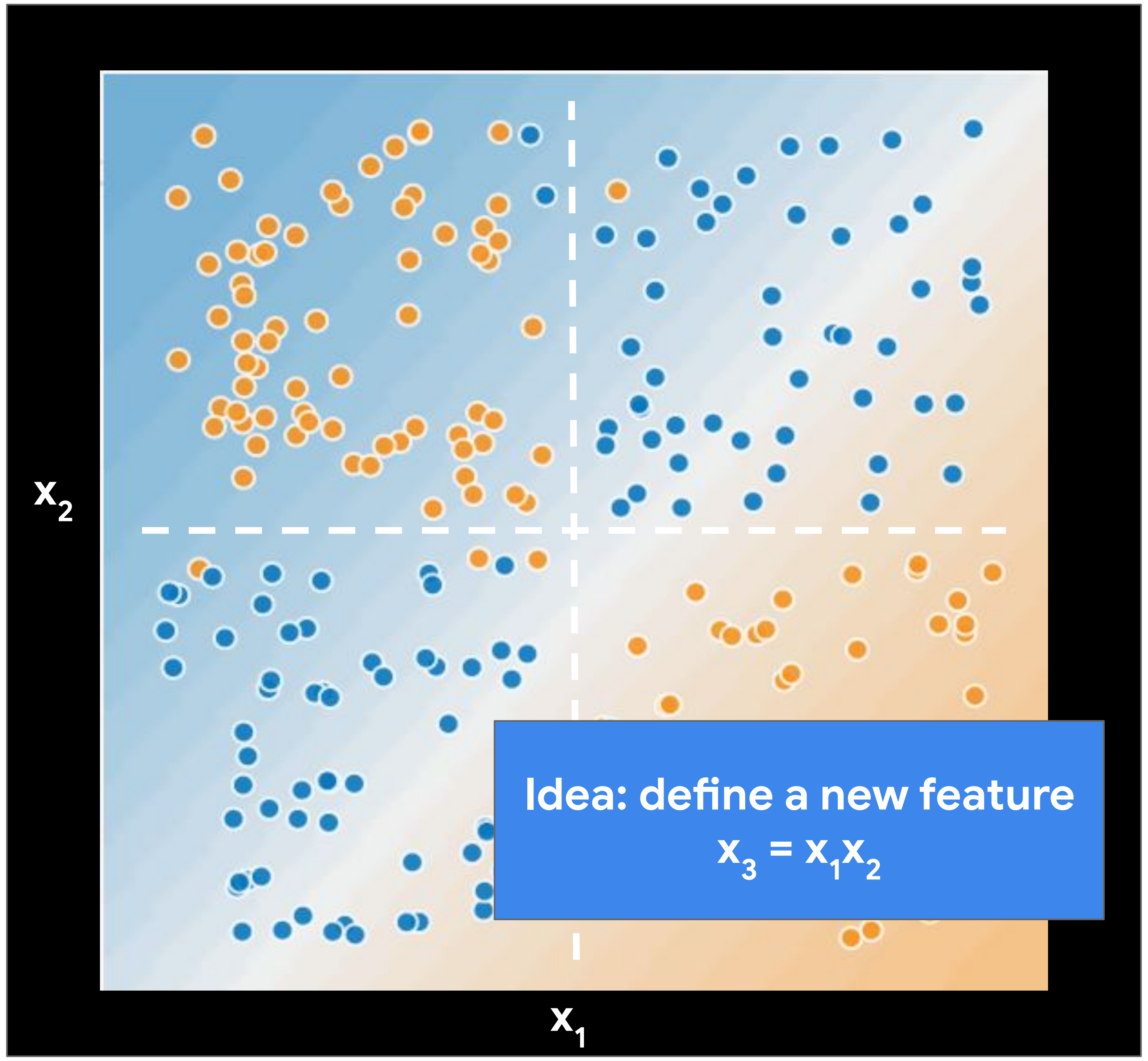



x_2
x_1

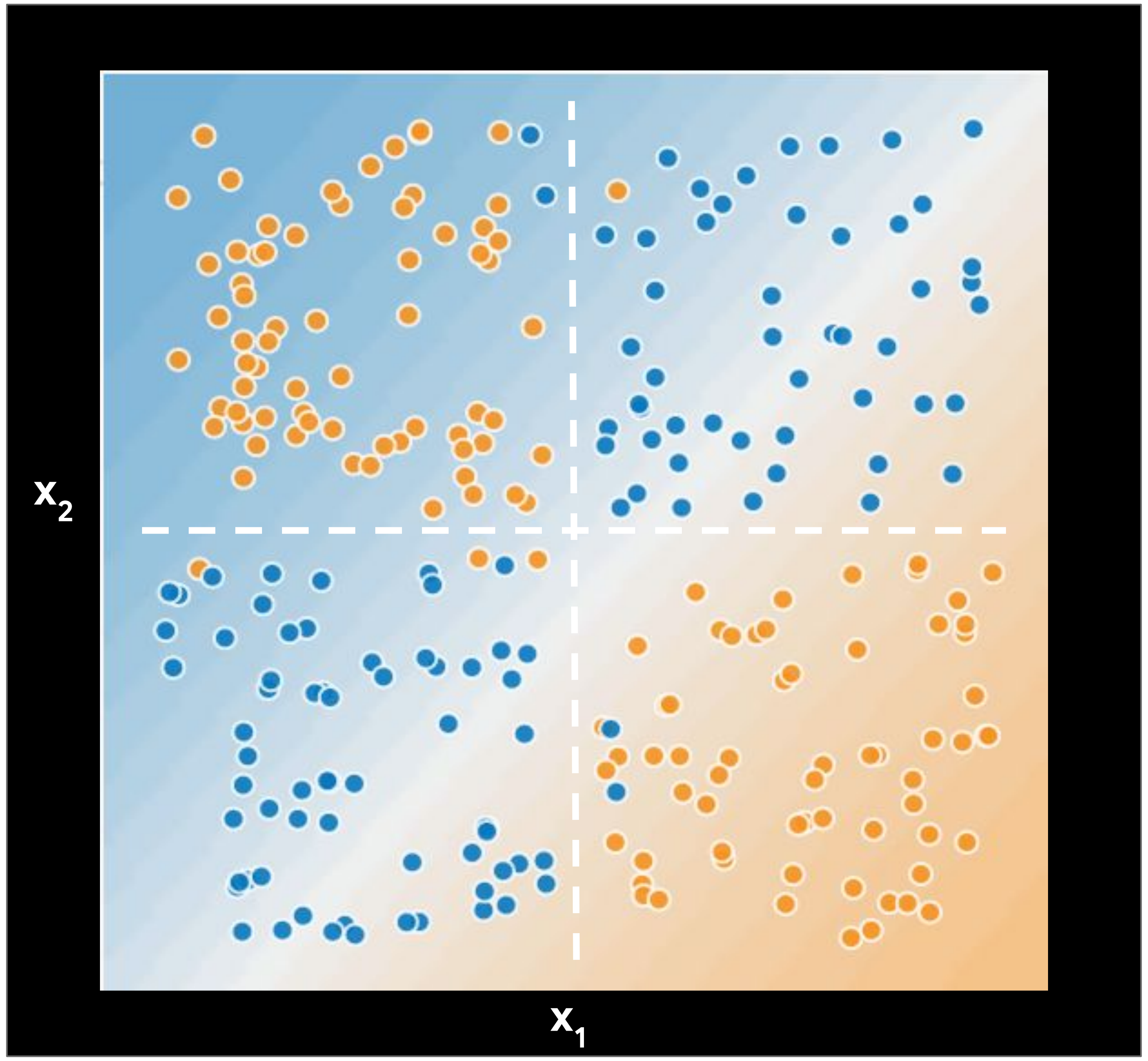

No rule like:
y = sign(b + w1x1 + w2x2)
x2
x1

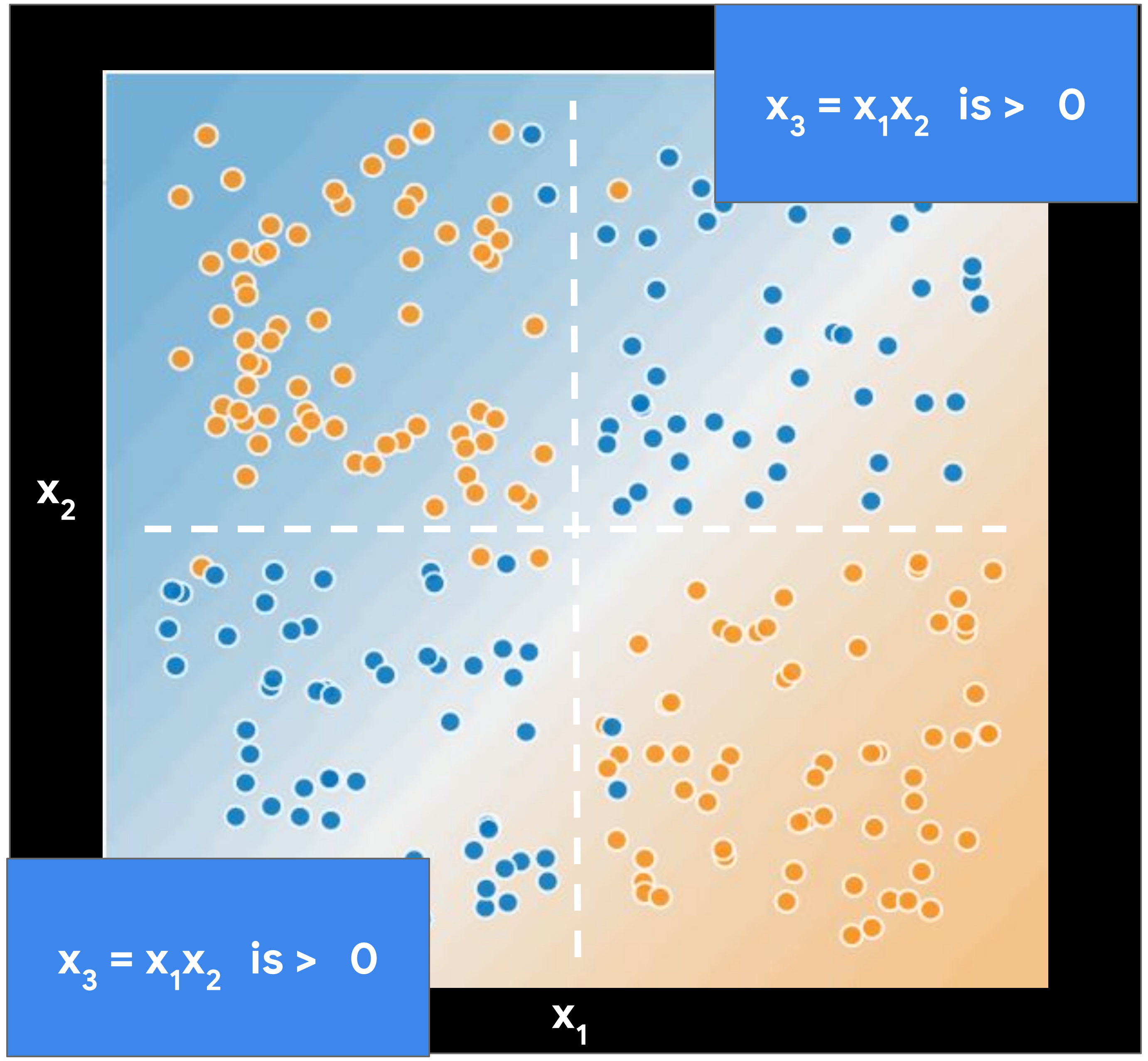

x_2
x_1

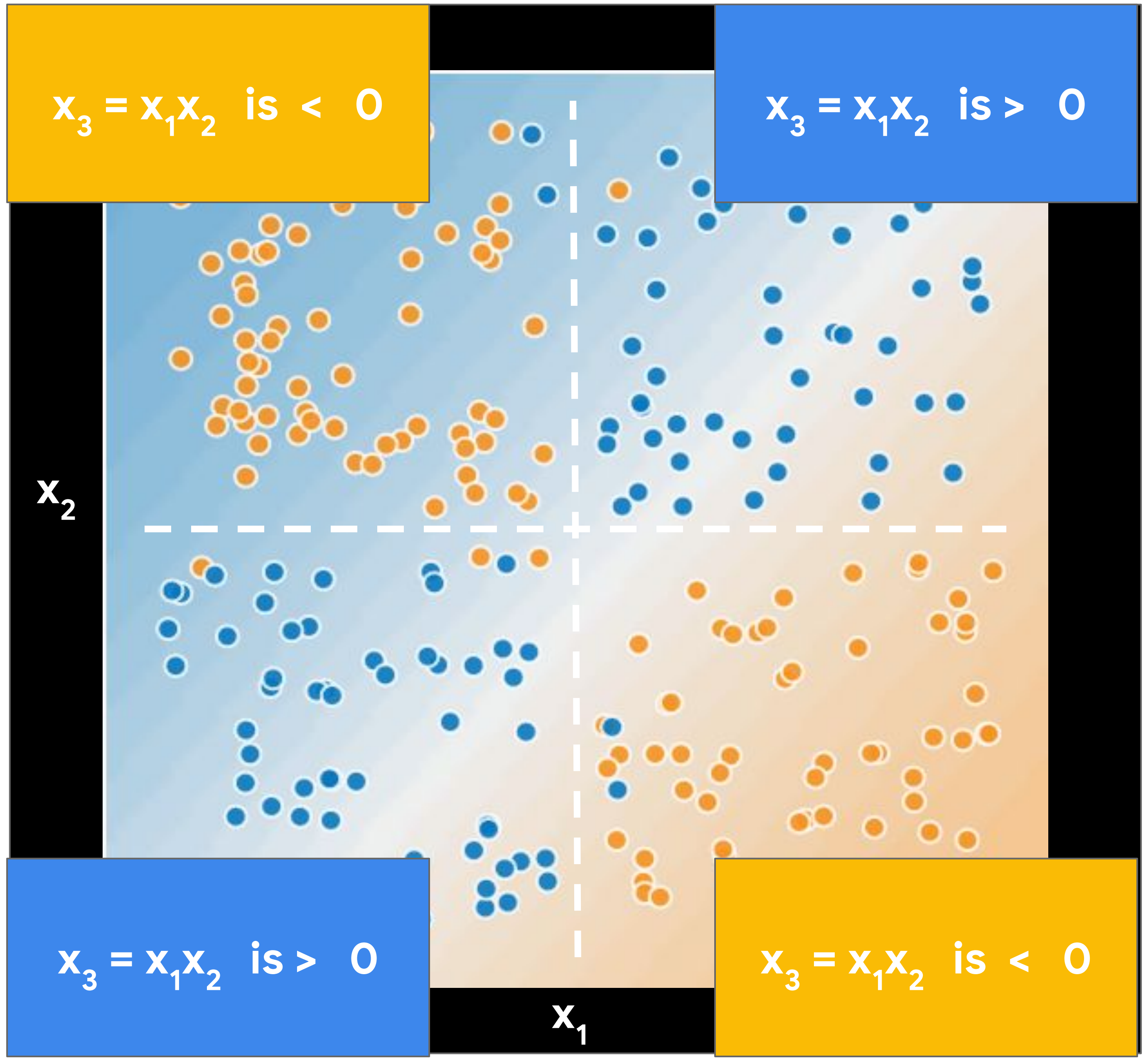

x_2
x_1



x_2
x_1
Idea: define a new feature
$x_3 = x_1 x_2$

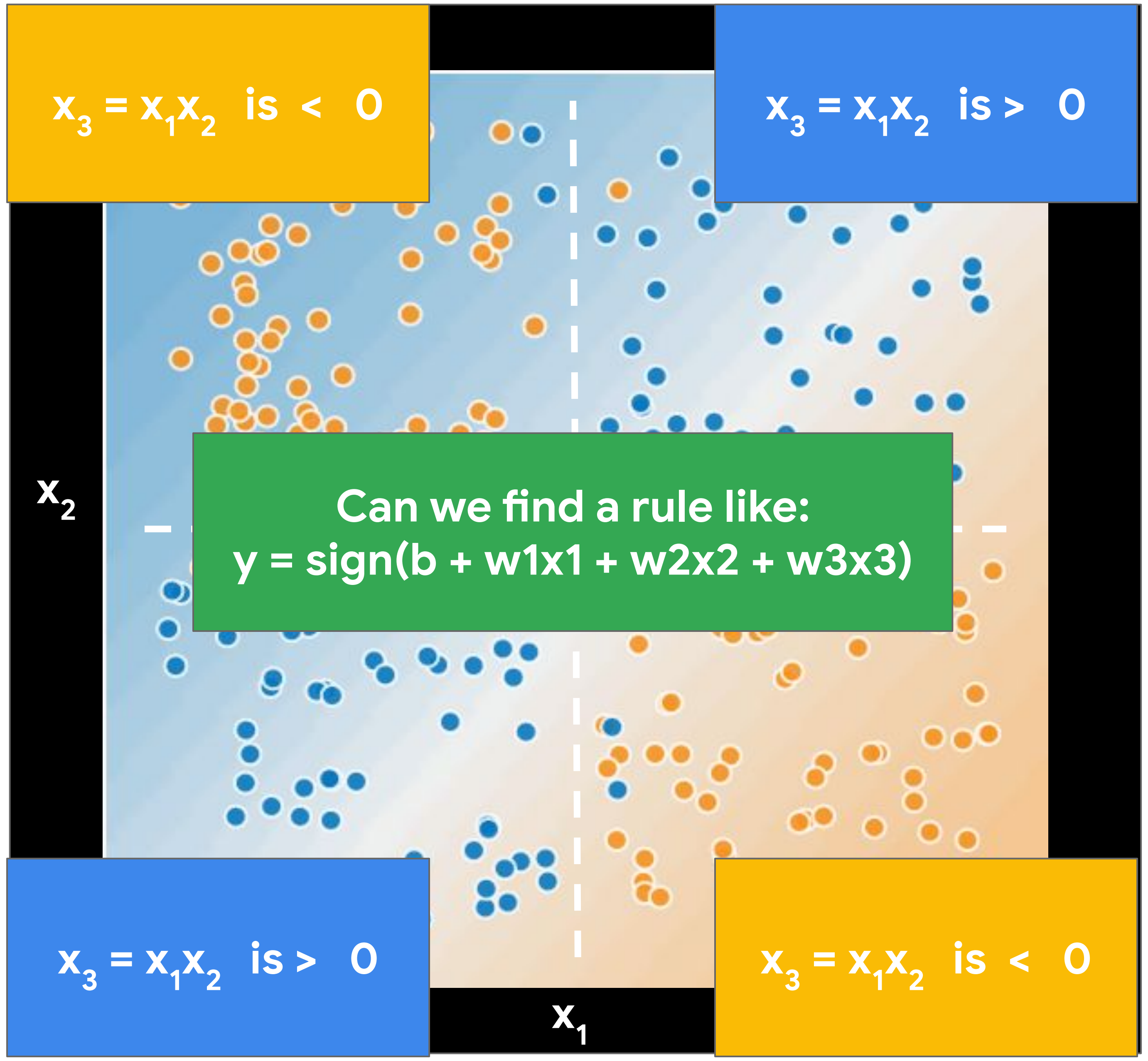

x_2
x_1


$x_3 = x_1x_2$ is > 0
x_2
$x_3 = x_1x_2$ is > 0
x_1

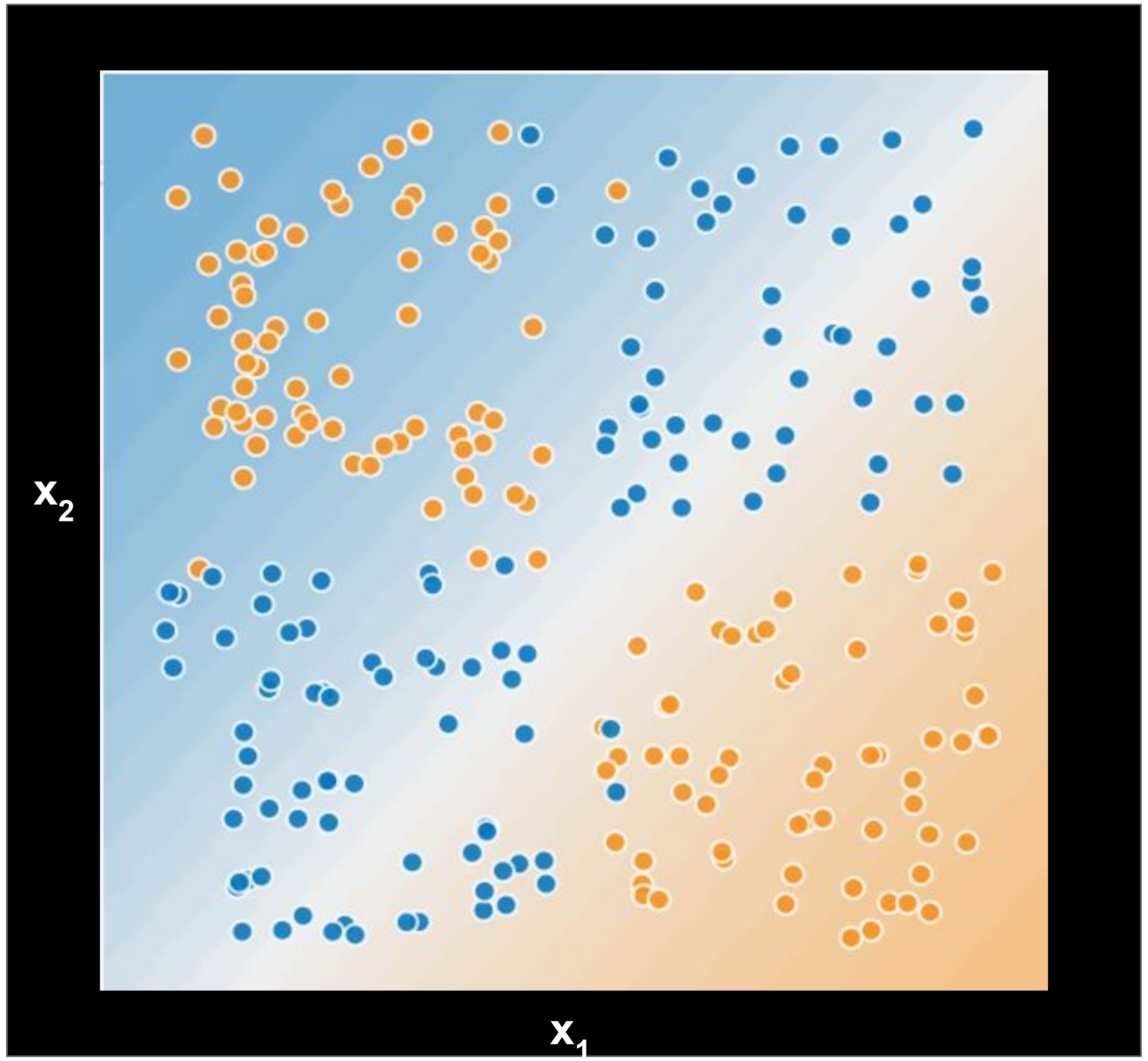


$x_3 = x_1x_2$ is < 0
$x_3 = x_1x_2$ is > 0
x_2
$x_3 = x_1x_2$ is > 0
x_1
$x_3 = x_1x_2$ is < 0

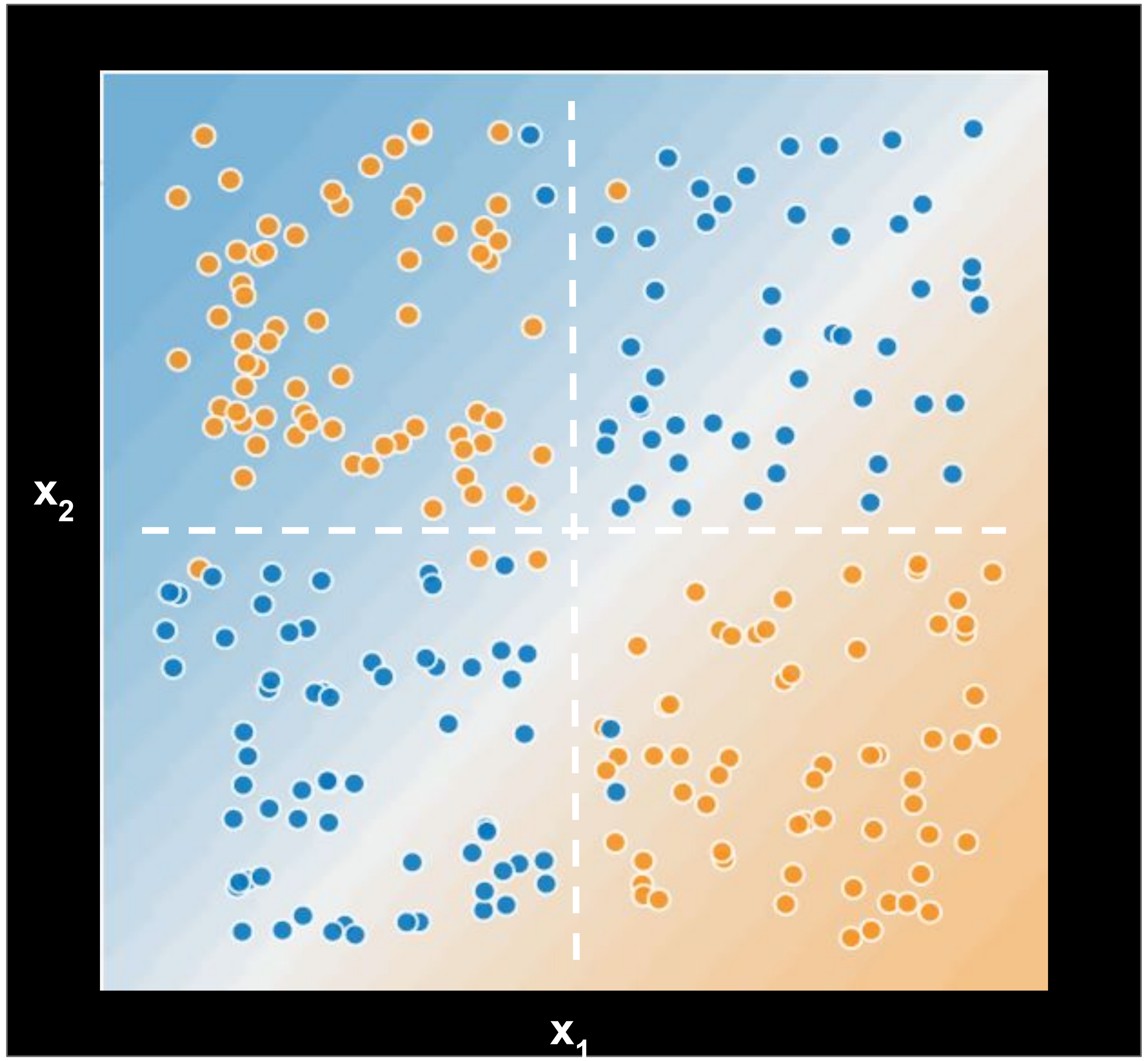
The feature cross provides a way to combine features in a linear model

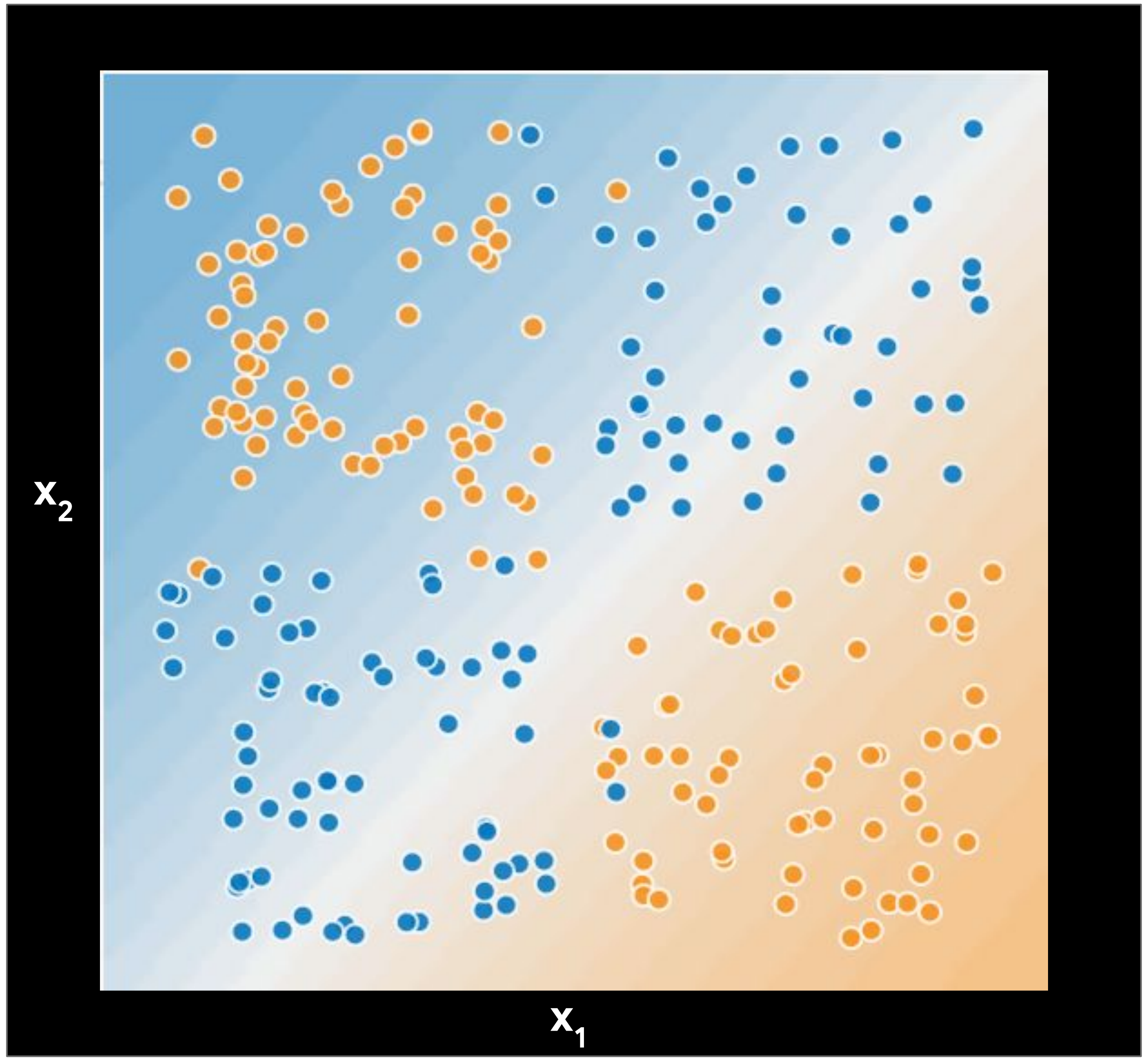


Using non-linear inputs in a linear learner

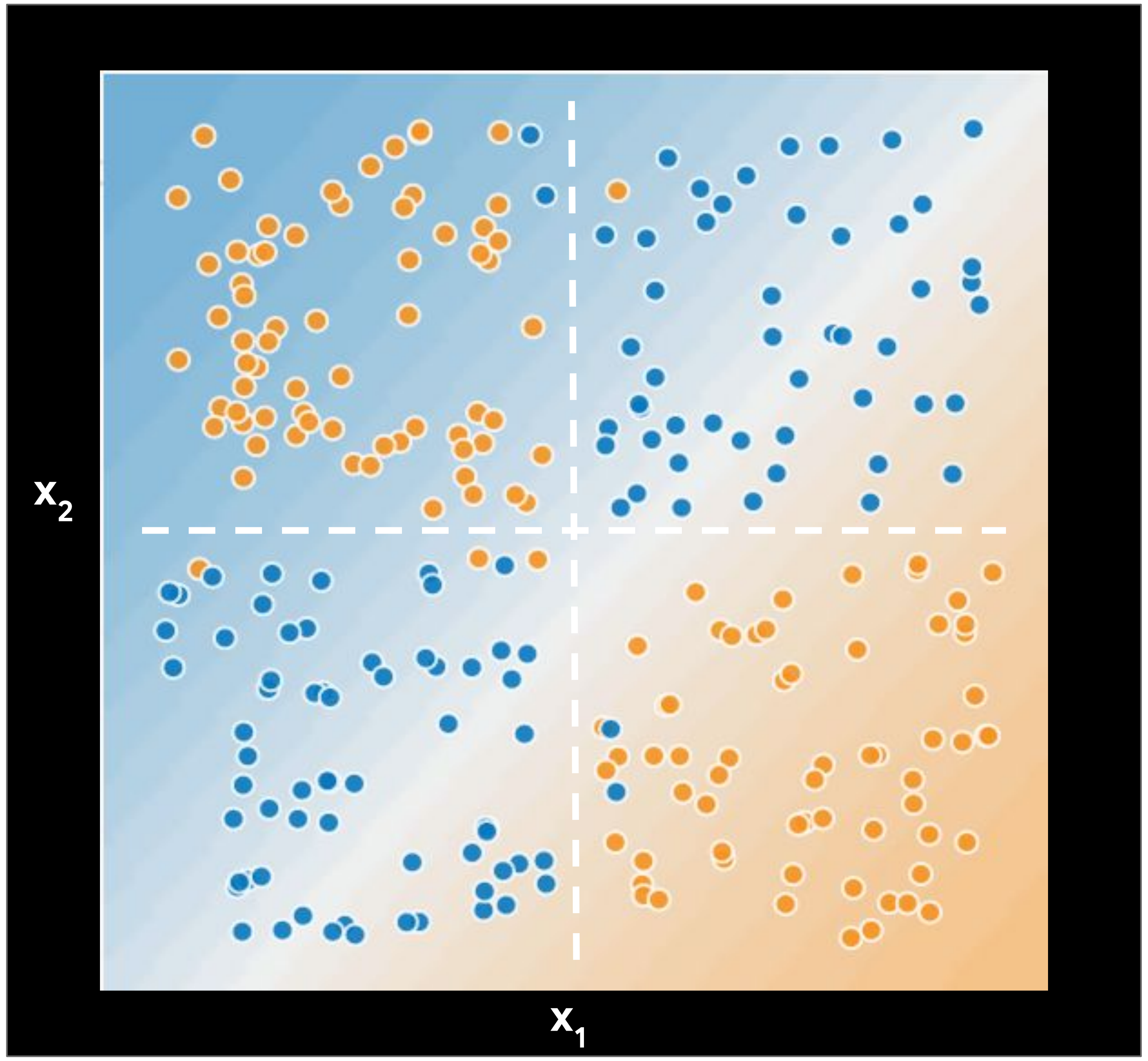


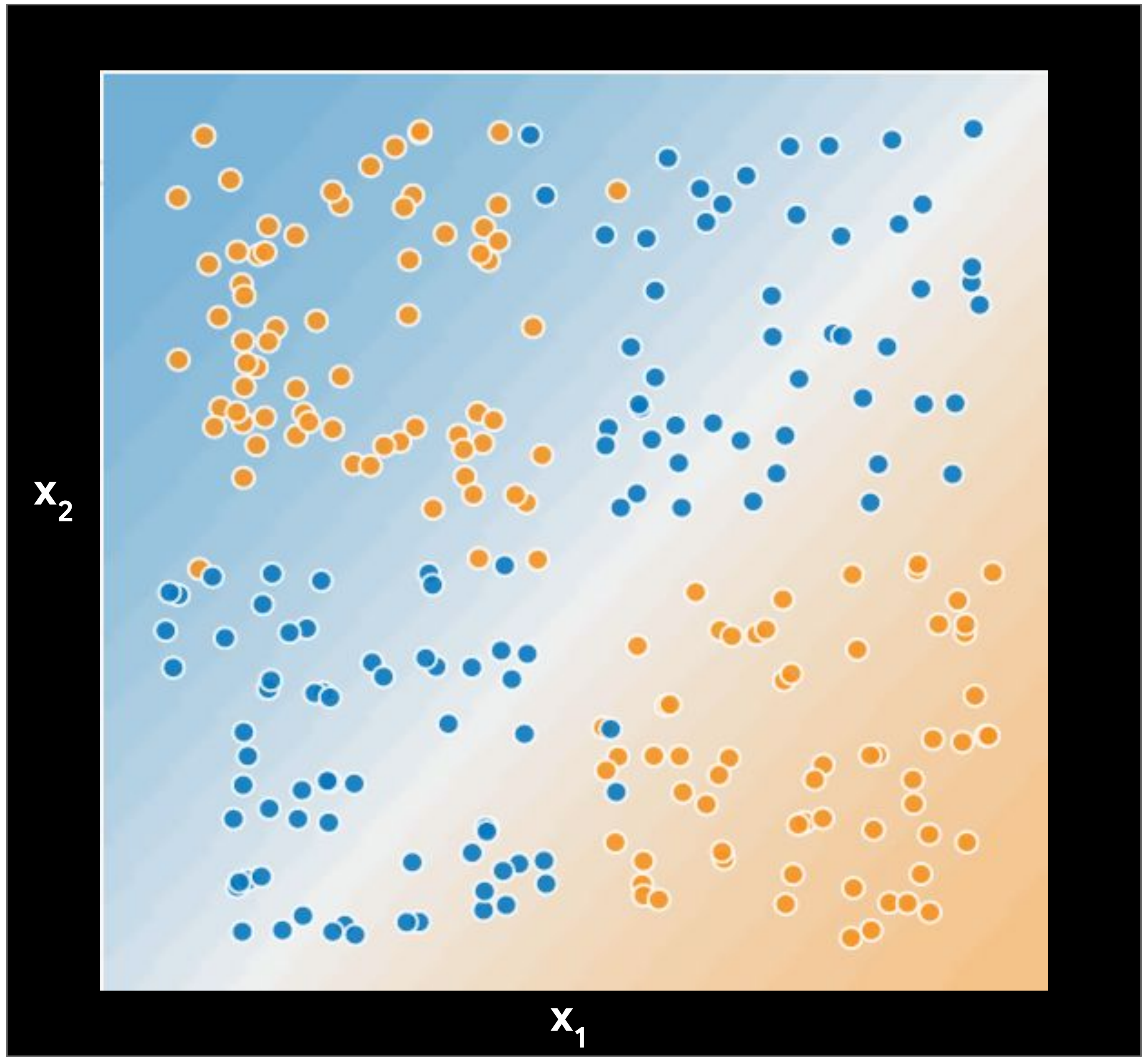
Using non-linear inputs in a linear learner



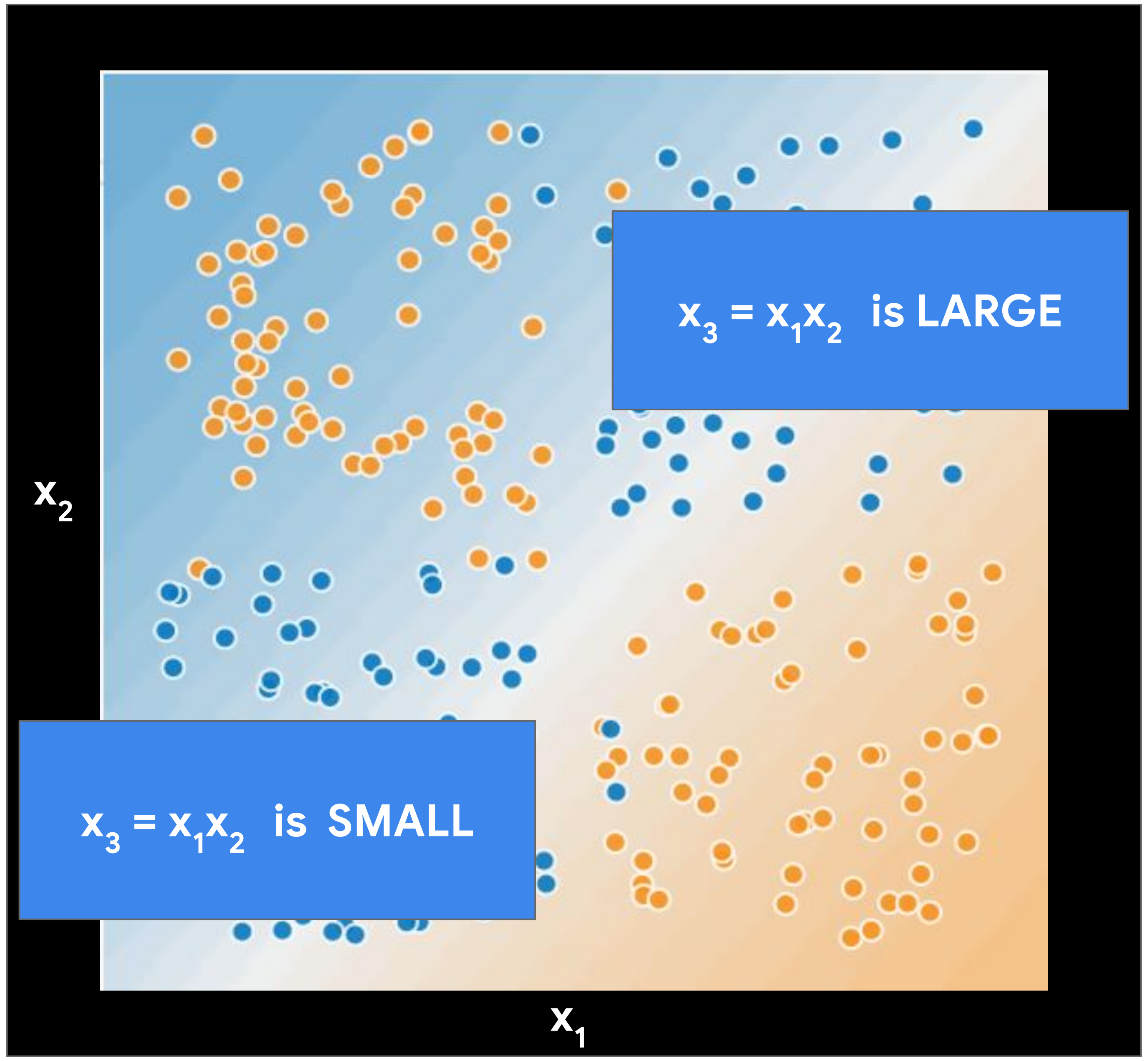


x_2
x_1

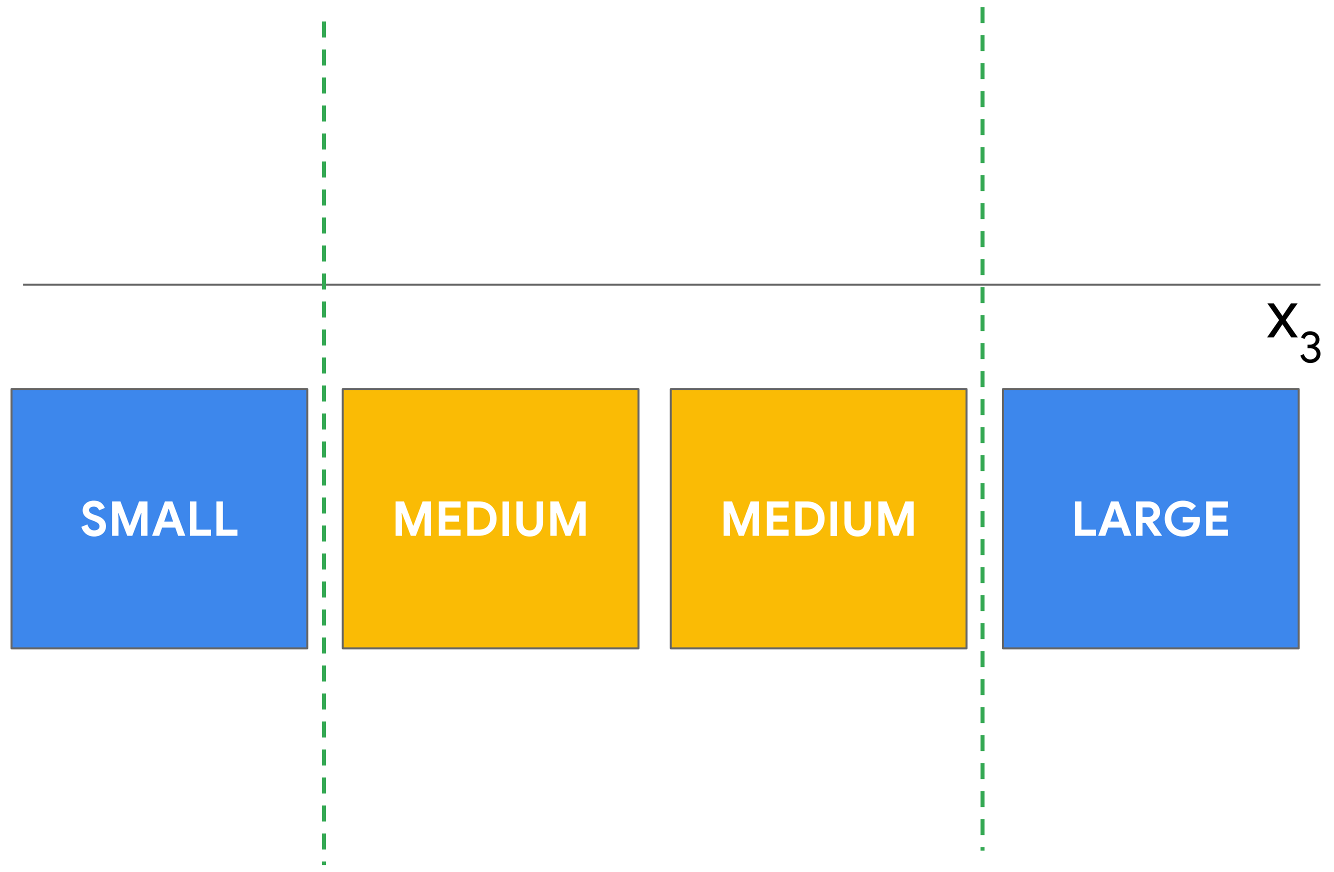

x2
x1



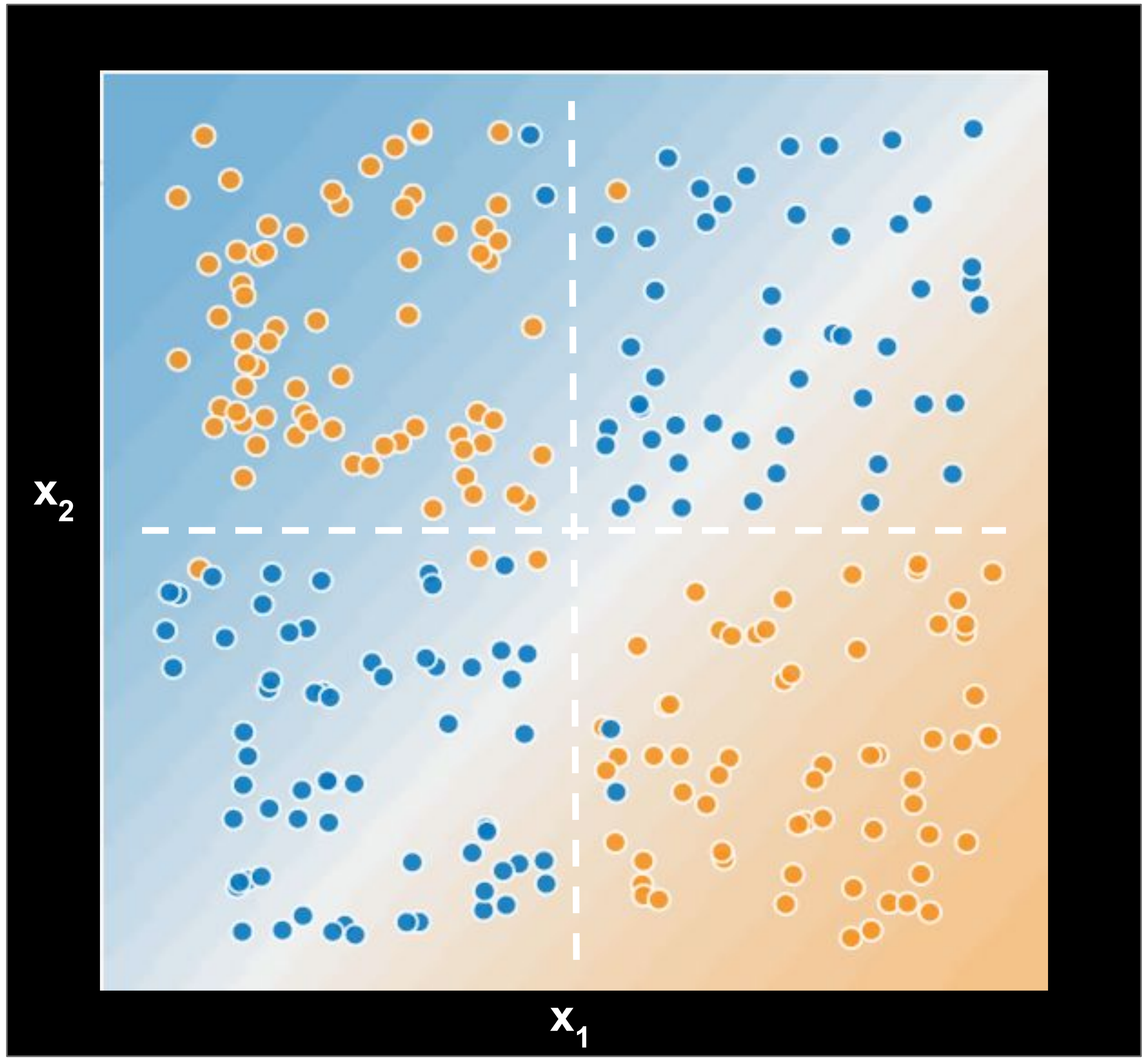
x_2
x_1


$x_3 = x_1x_2$ is LARGE
x_2
$x_3 = x_1x_2$ is SMALL
x_1

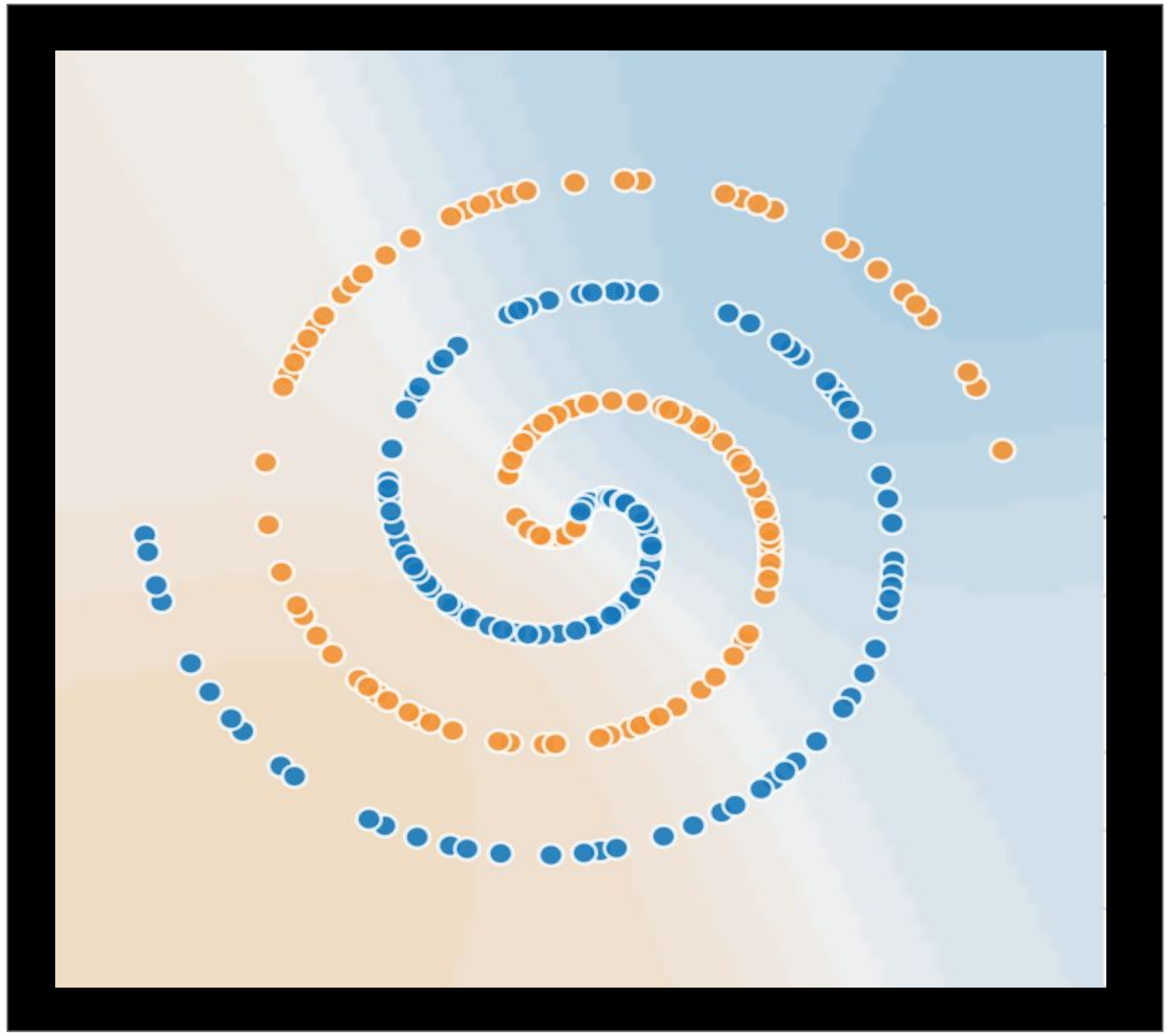
You now need two lines or more free parameters



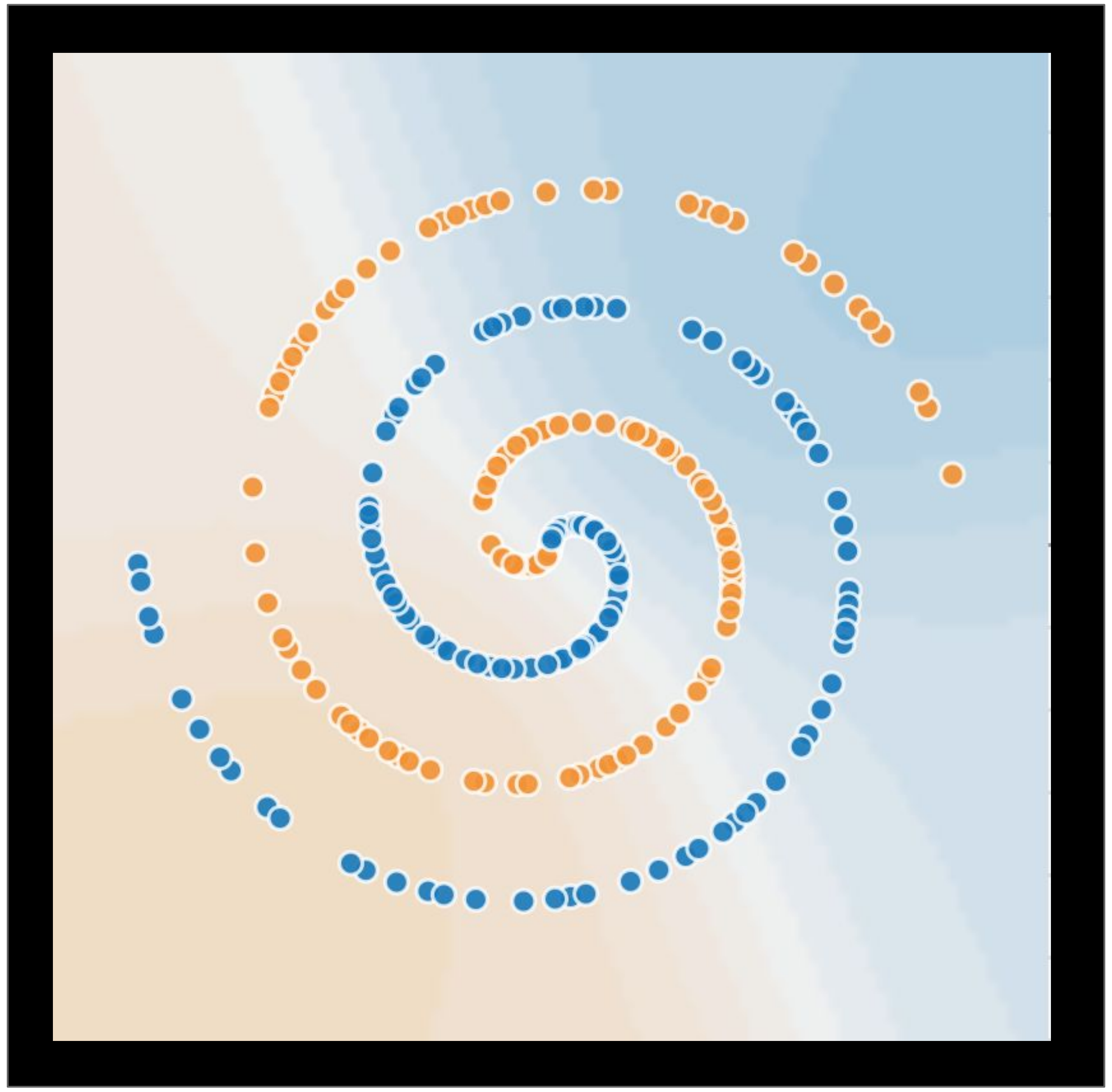
The white lines discretize the space



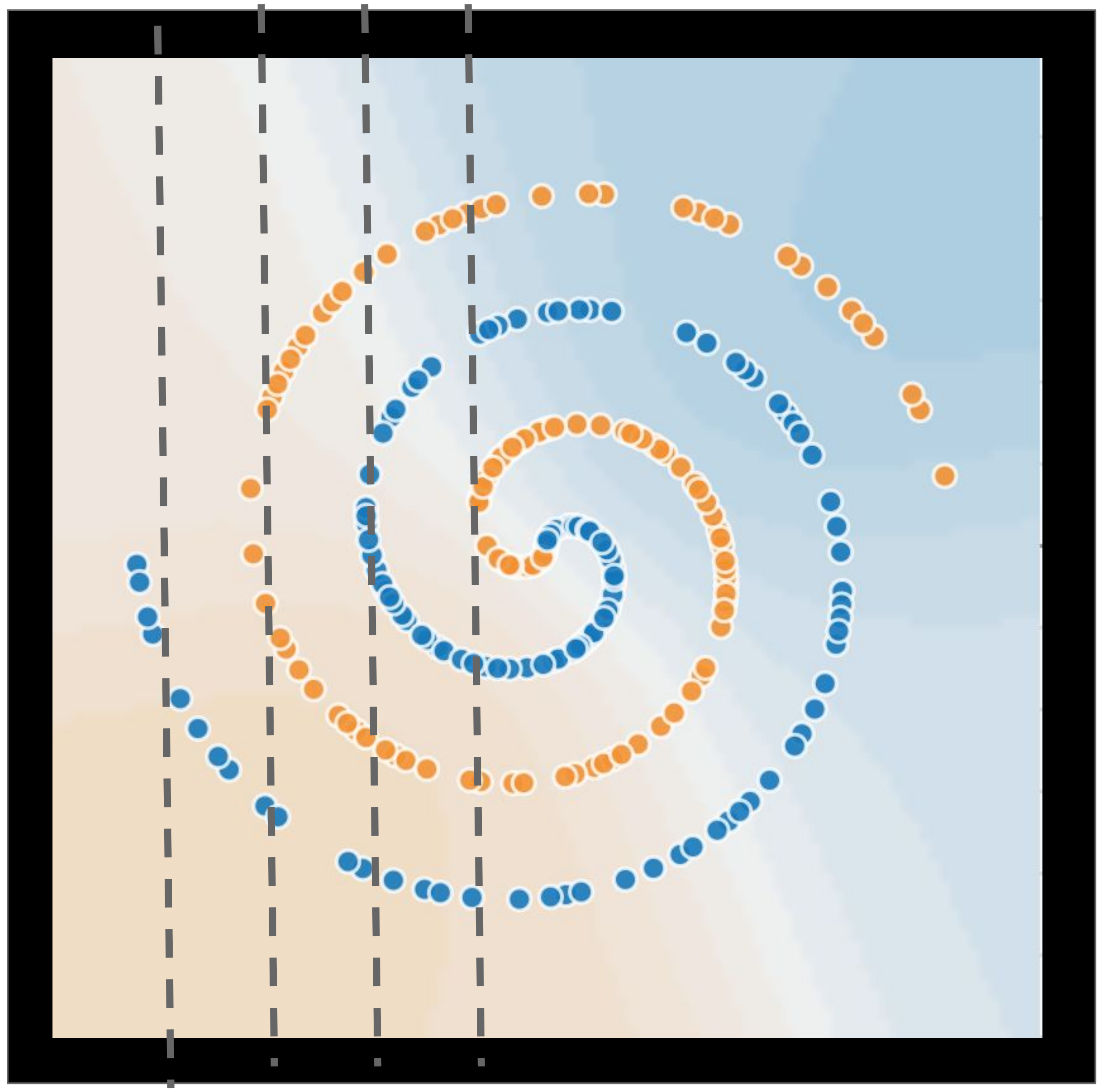
Can a linear model work for this problem?



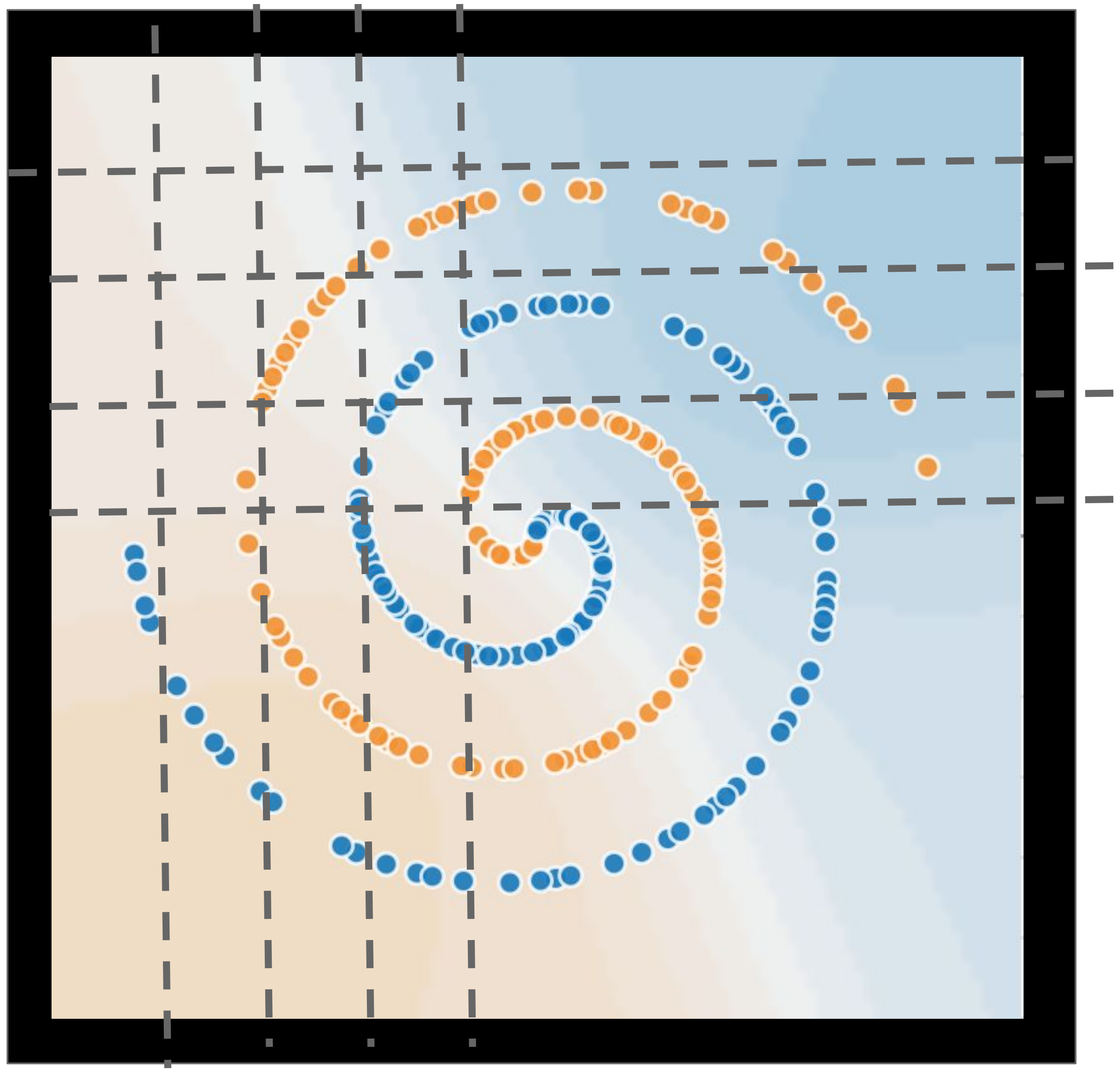
What if we discretize x1 and x2 and then multiply?



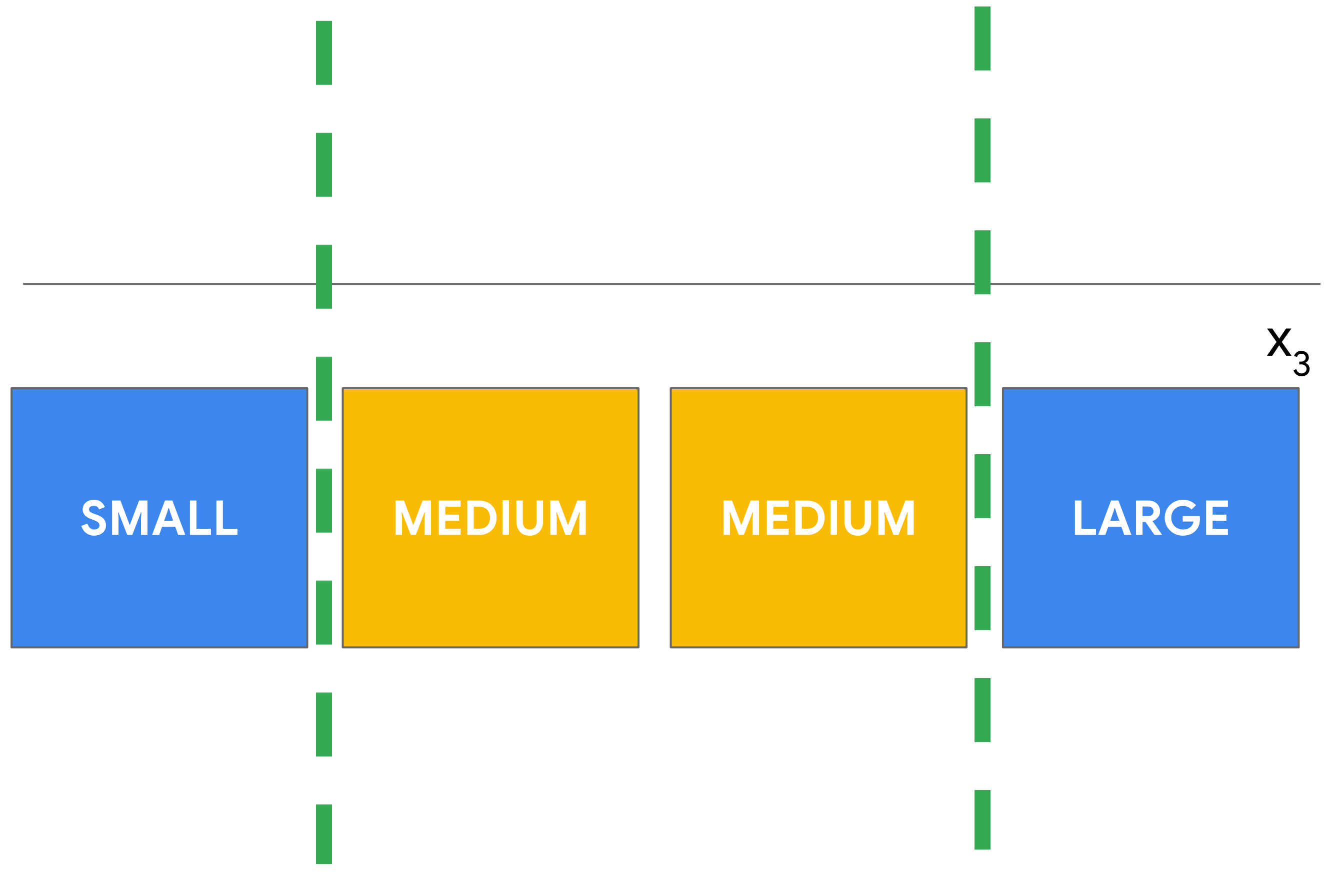
What if we discretize x1 and x2 and then multiply?



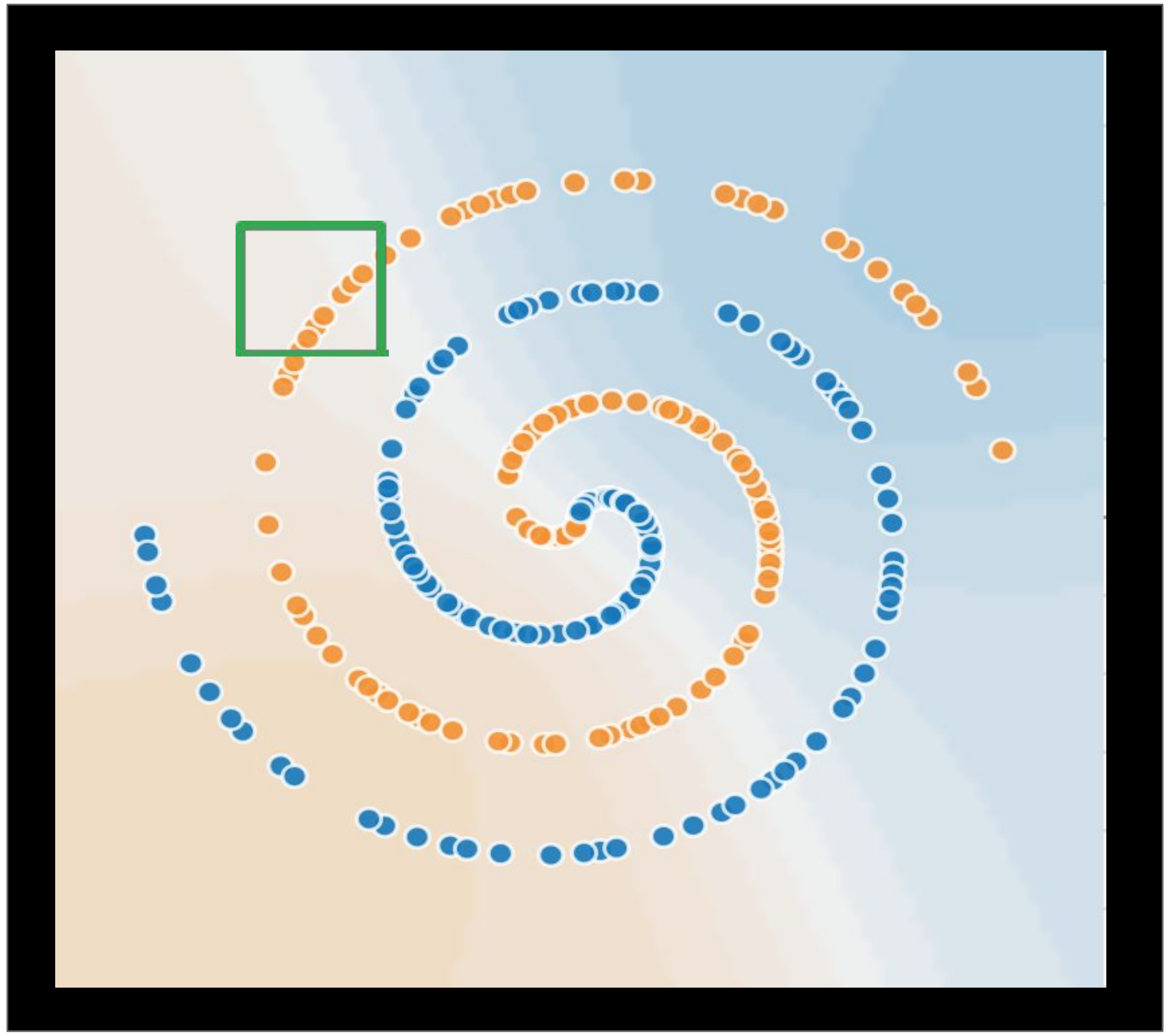
What if we discretize x1 and x2 and then multiply?



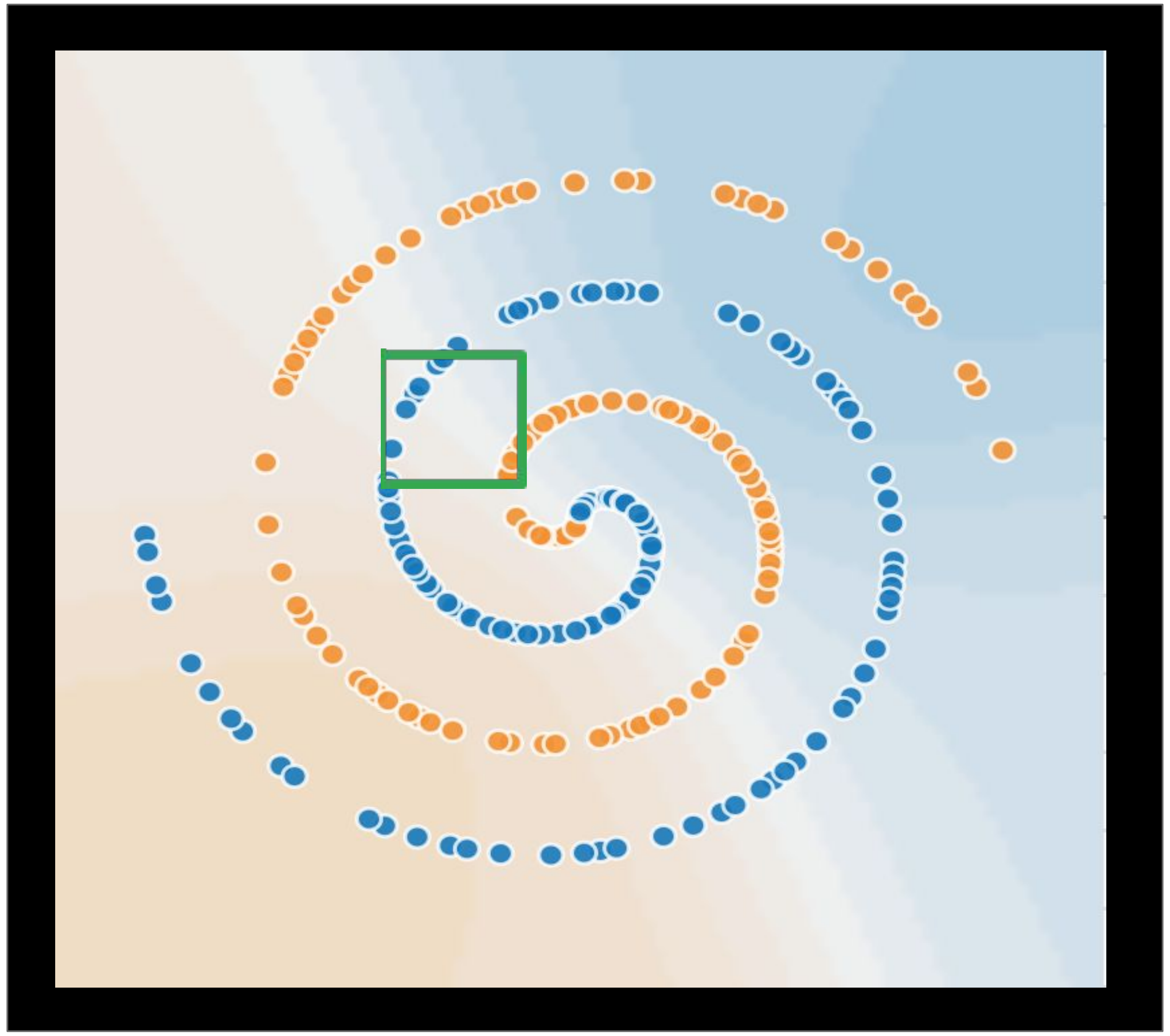
Dividing the input space with two lines yields four quadrants



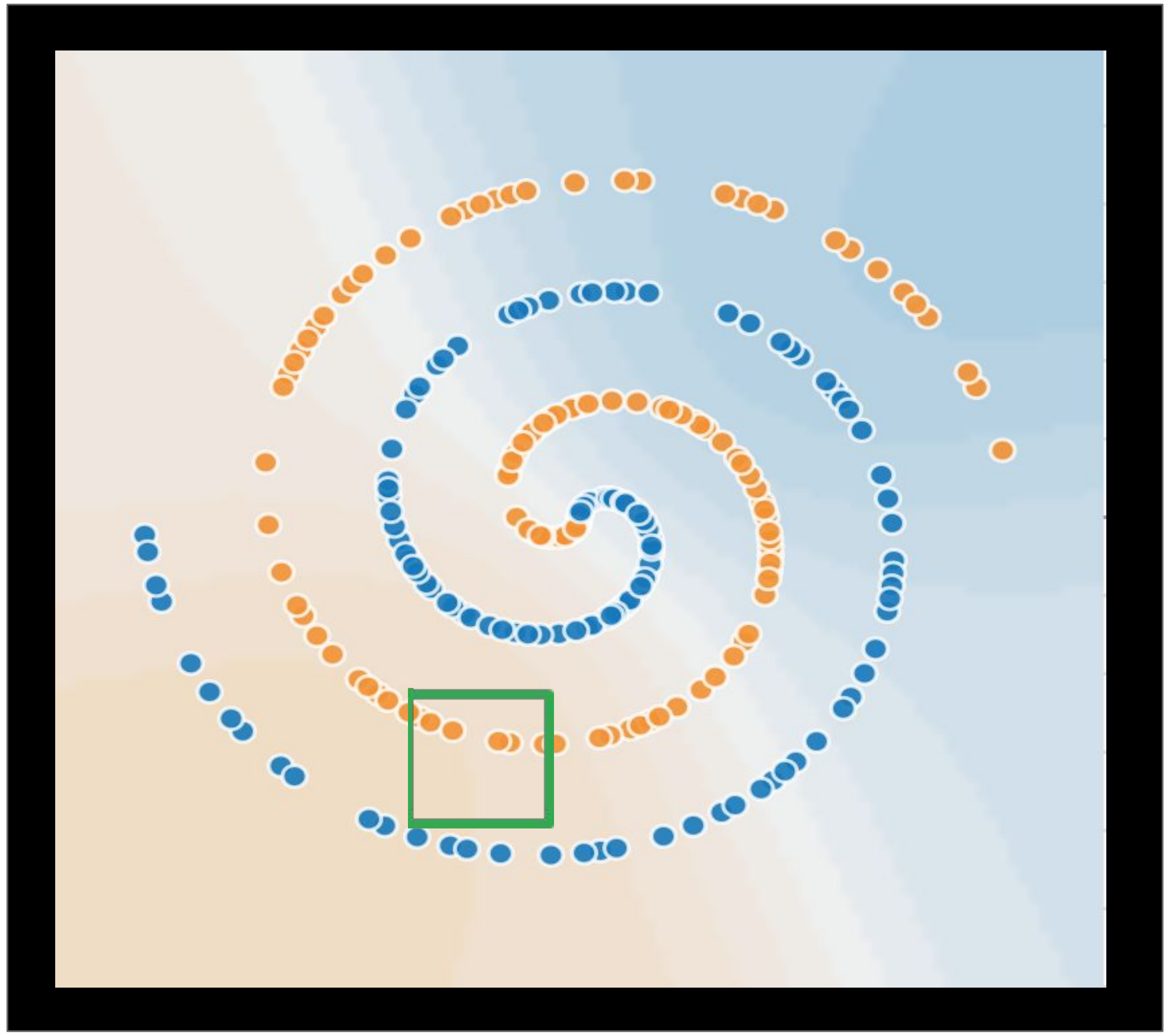
Separate prediction per grid cell



Separate prediction per grid cell

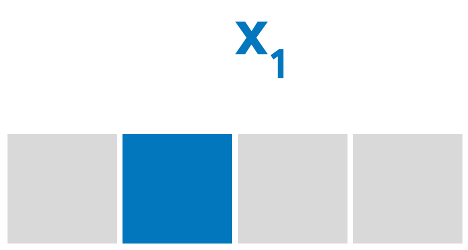


Separate prediction per grid cell



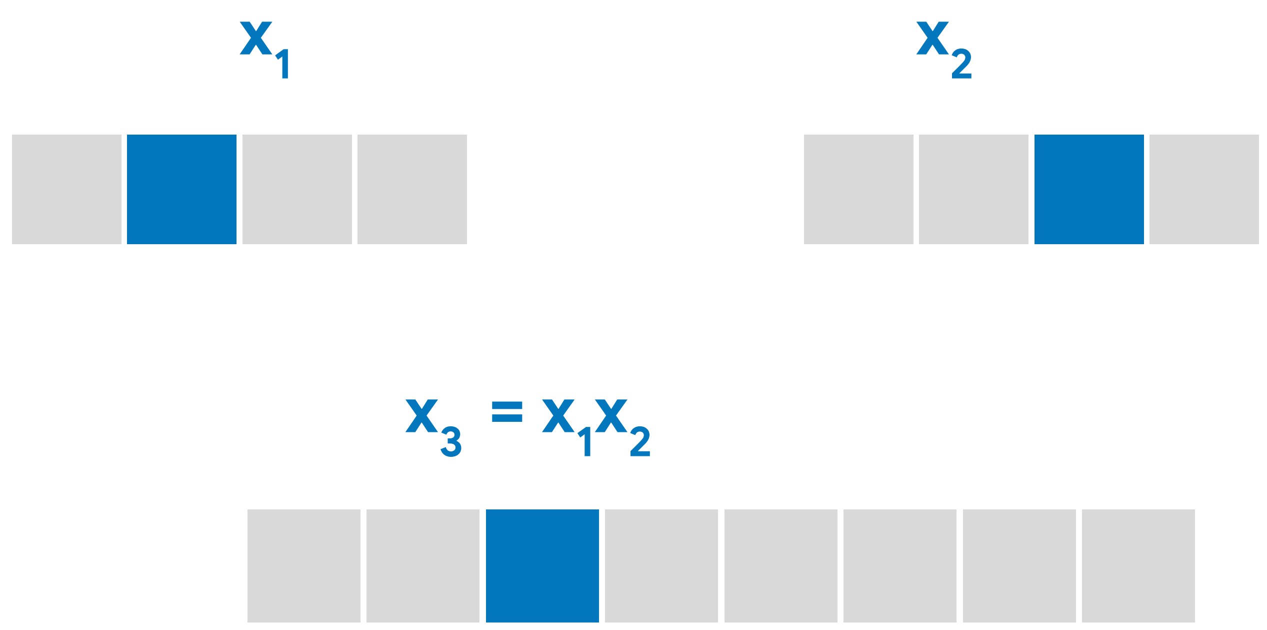
The weight of a cell is essentially the prediction for that cell

The weight of a cell is essentially the prediction for that cell

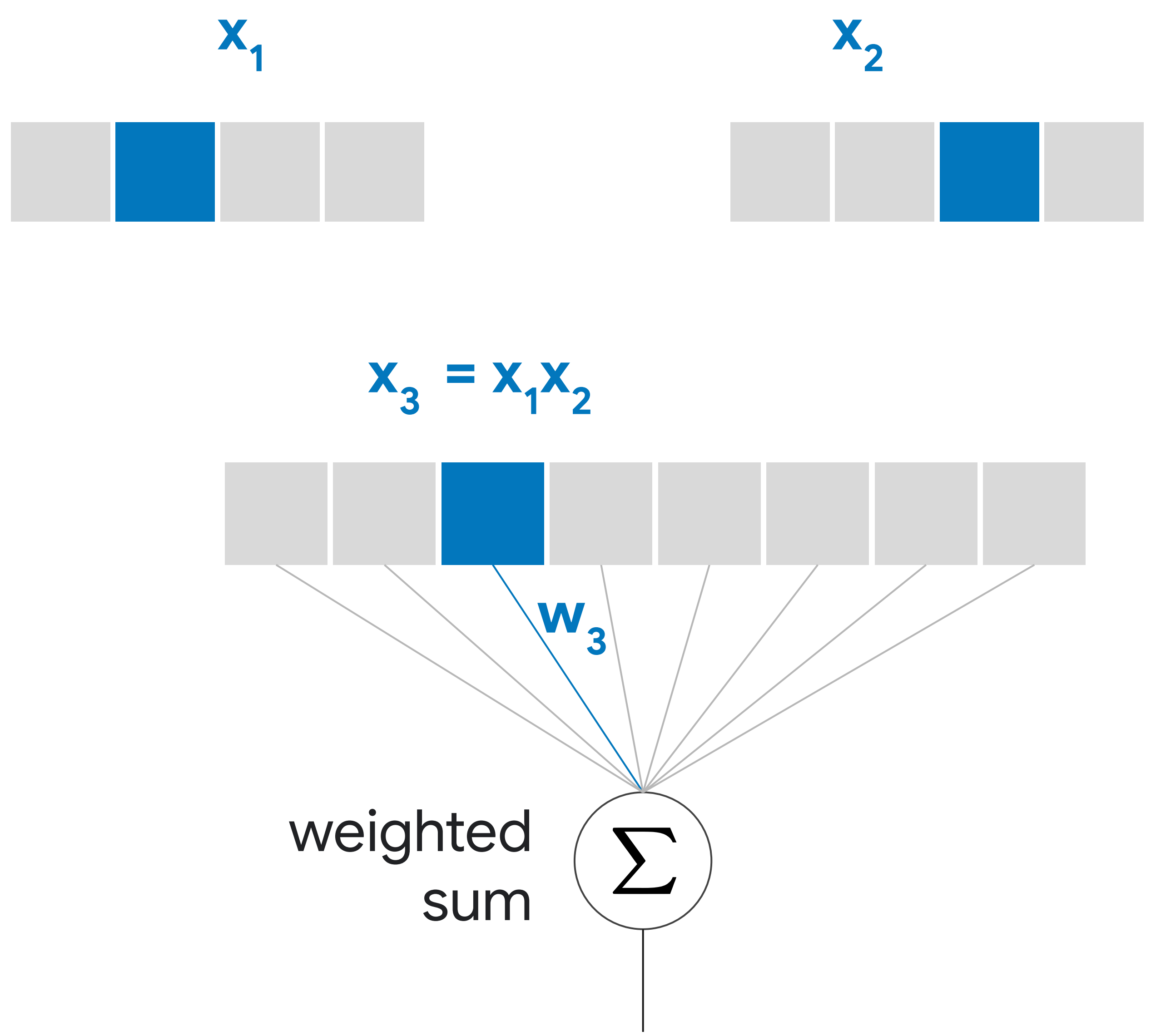


The weight of a cell is essentially the prediction for that cell

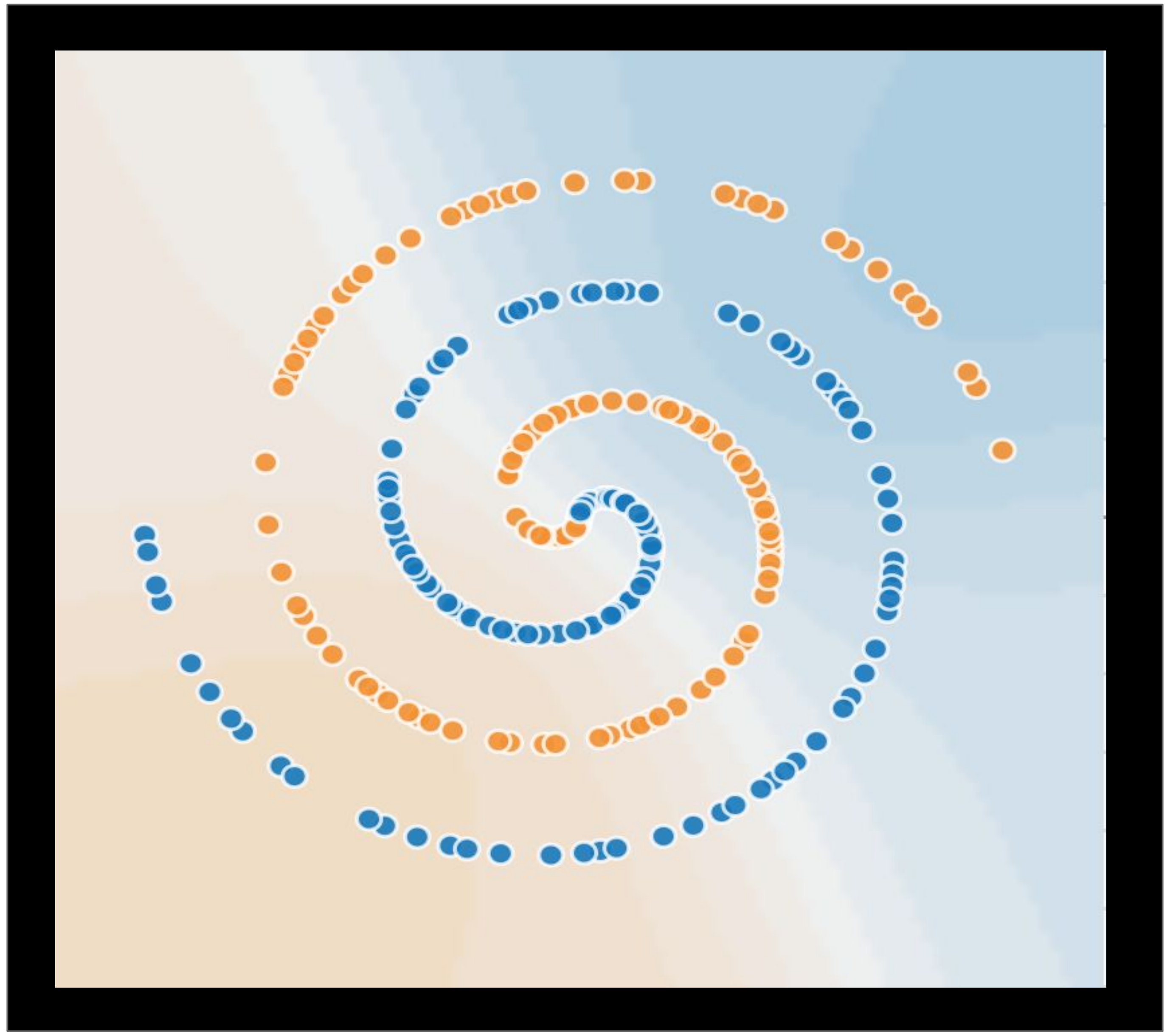
The weight of a cell is essentially the prediction for that cell



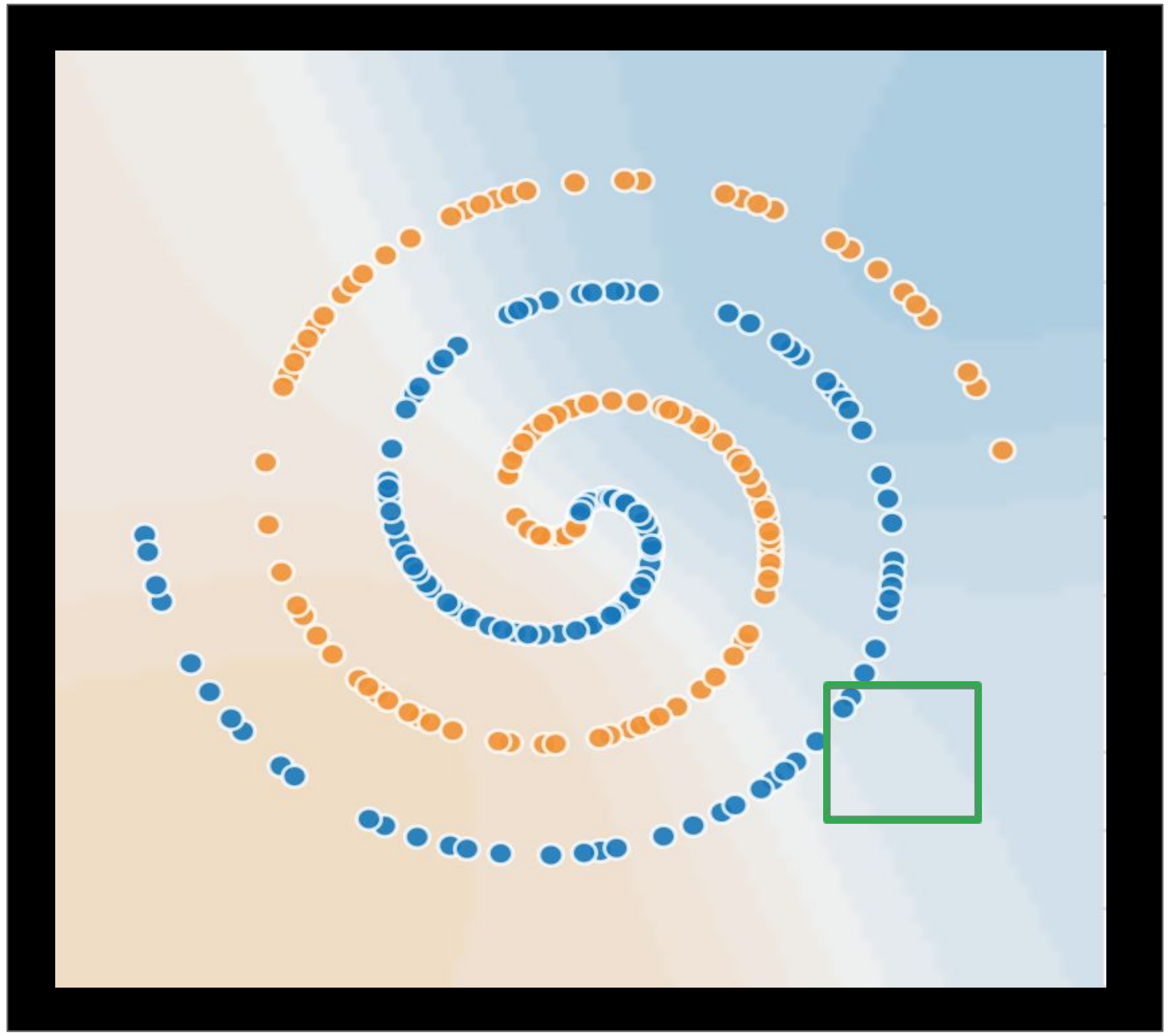
The weight of a cell is essentially the prediction for that cell



A feature cross memorizes the input space



A feature cross memorizes the input space



Feature crosses memorize!

Feature crosses memorize!

Goal of ML is generalization

Feature crosses memorize!

Goal of ML is generalization

Memorization works when you have lots of data

Feature crosses memorize!

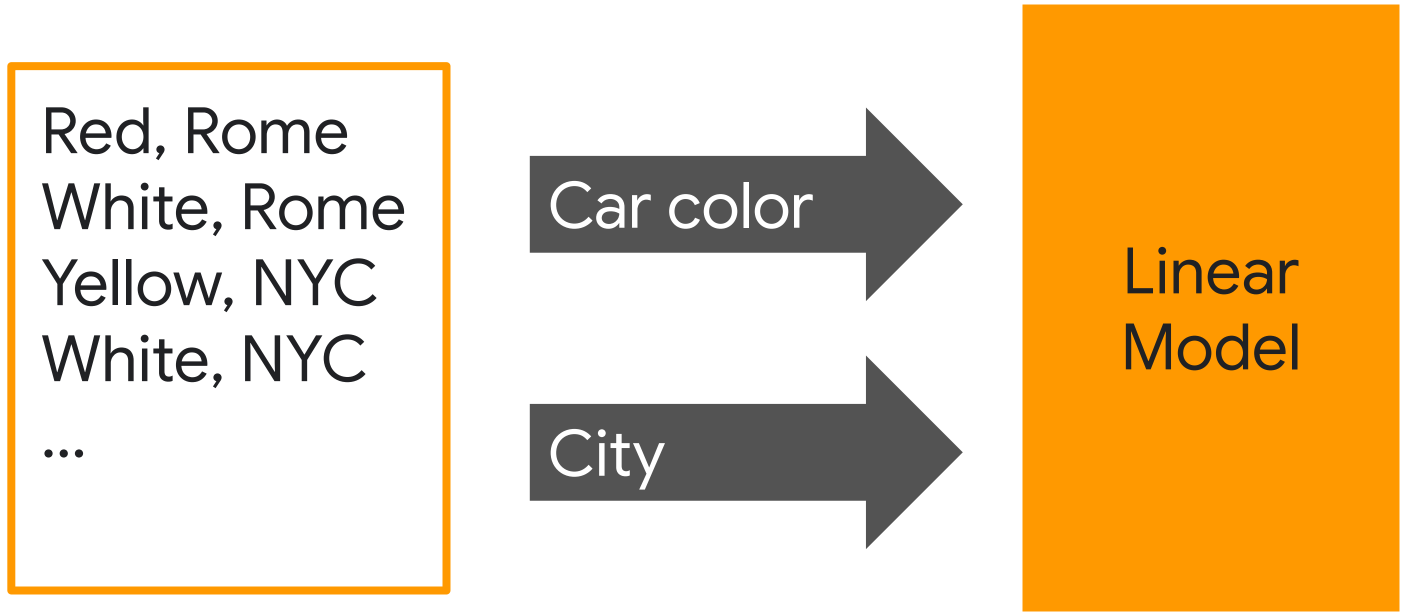
Goal of ML is generalization

Memorization works when you have lots of data

Feature crosses are powerful

Which of these cars is a taxi?

Assume that your input data looks like this

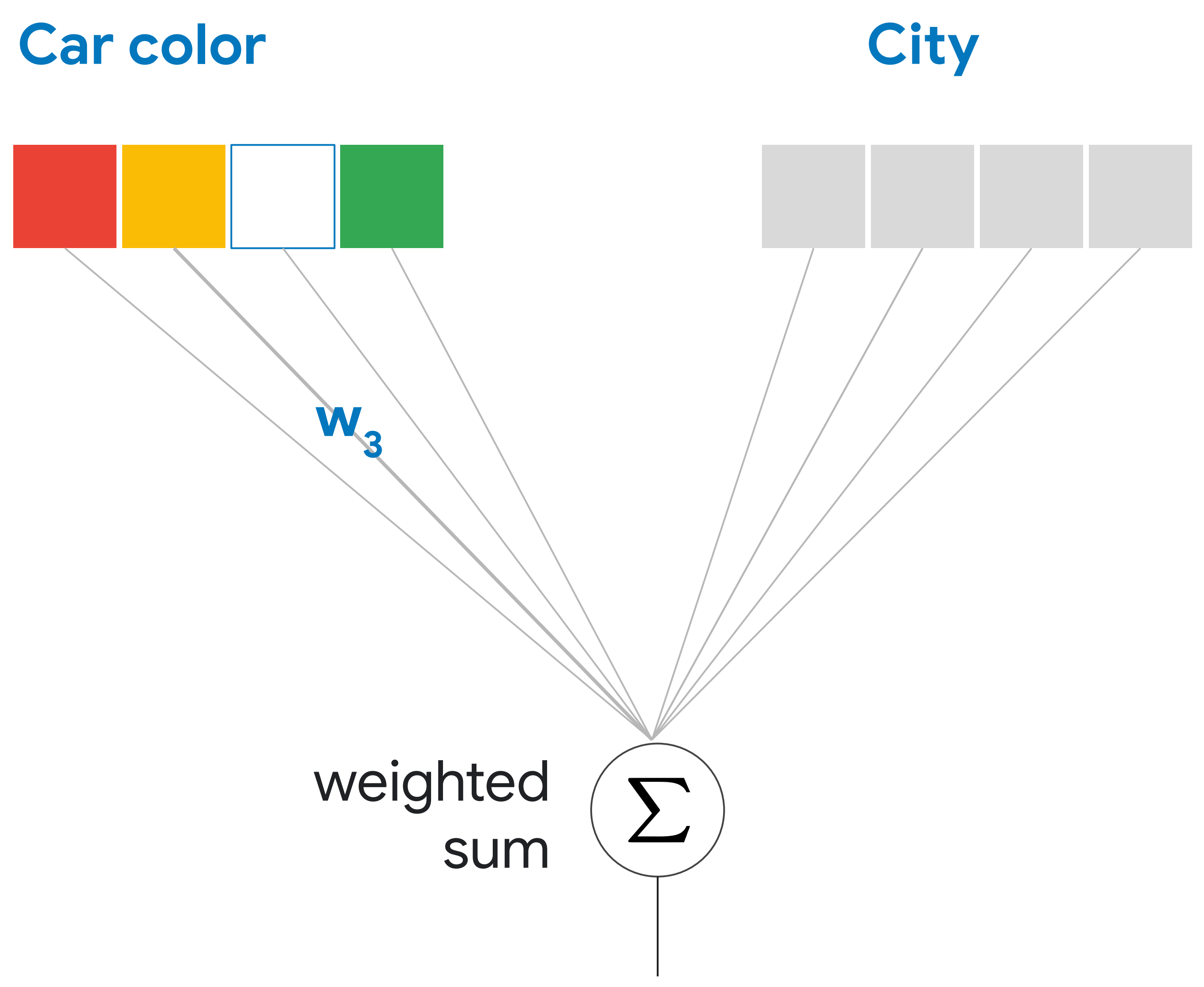


The linear model has problems ...

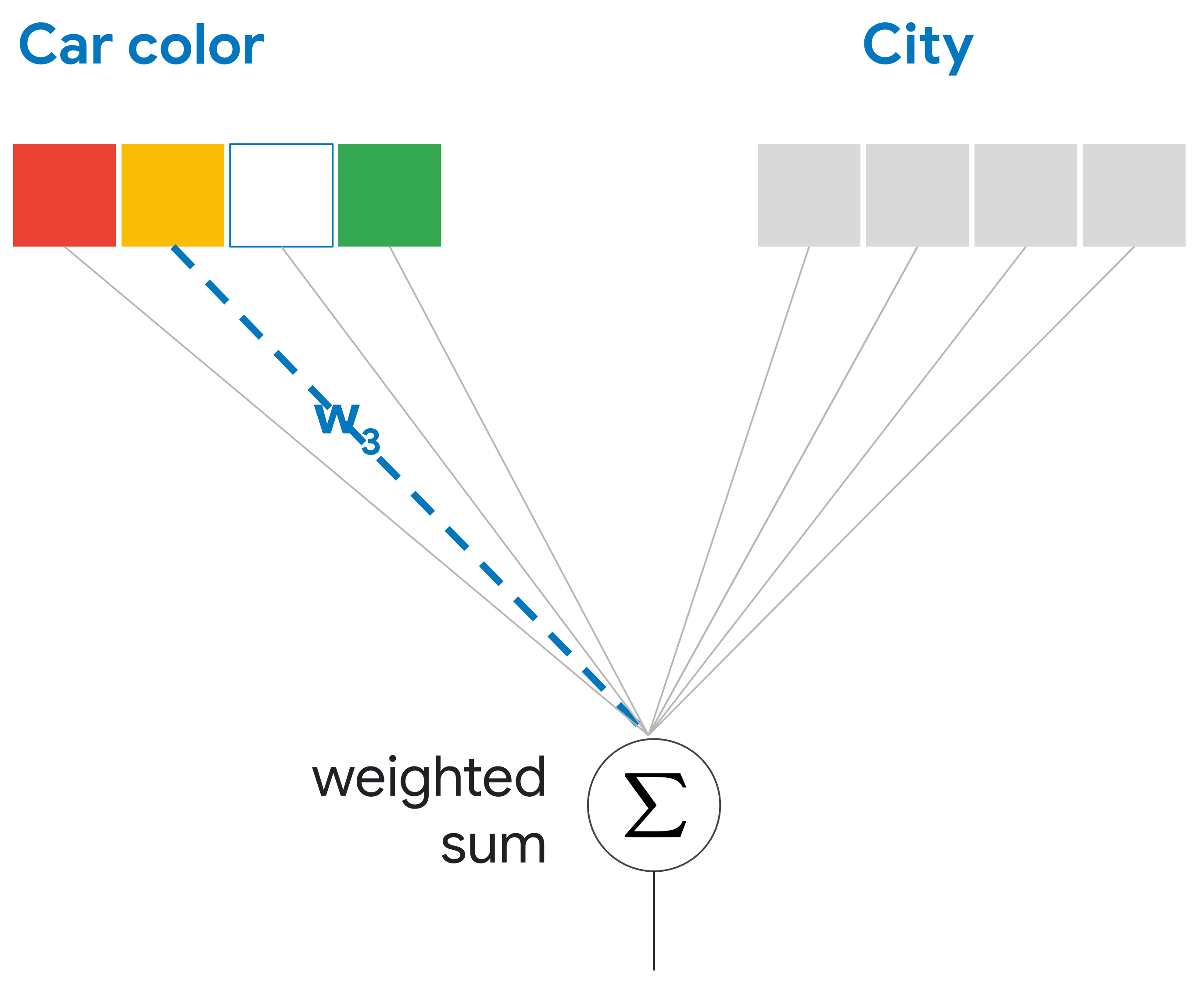
Car color

The linear model has problems …

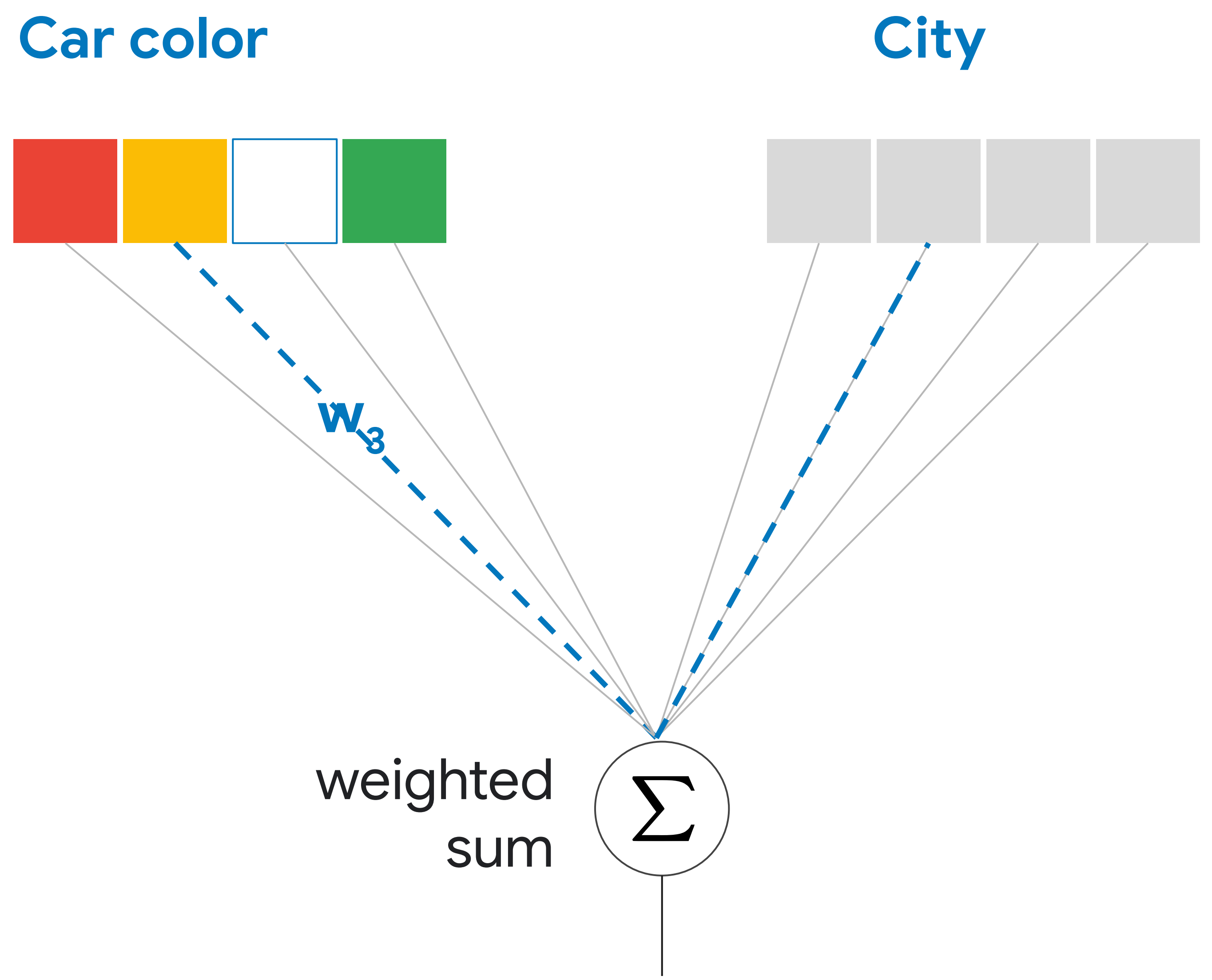
The linear model has problems ...



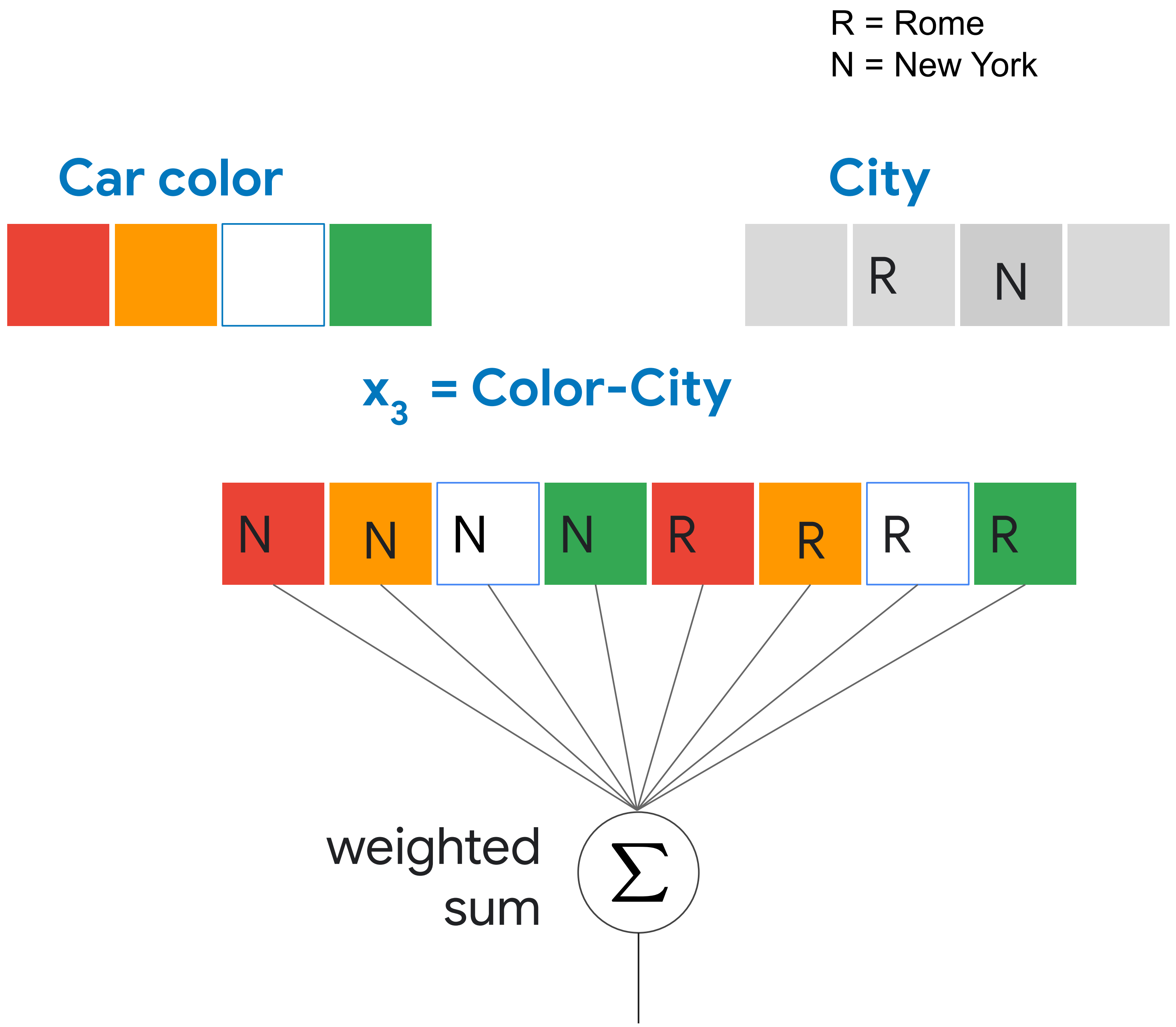
The linear model has problems ...



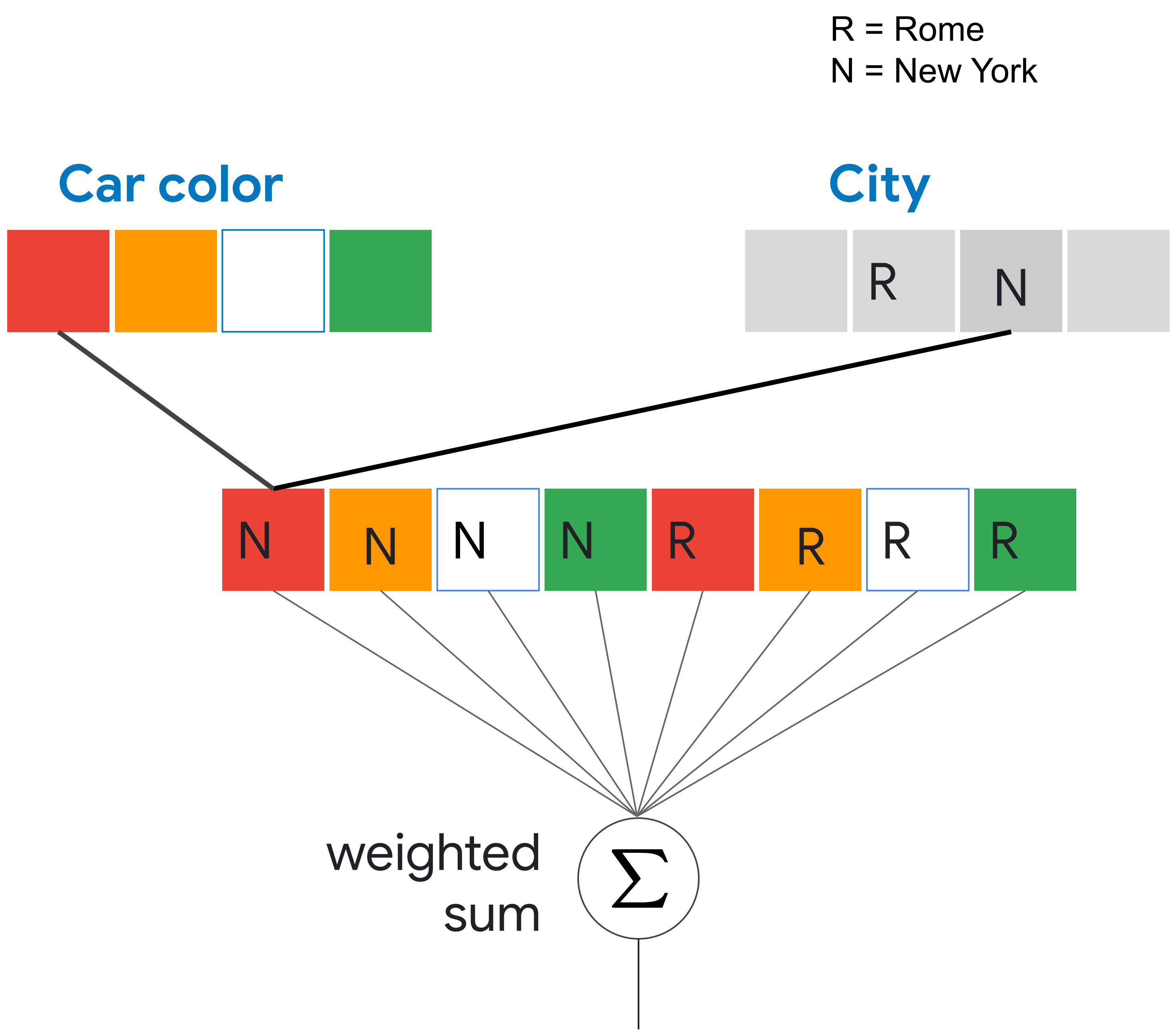
The linear model has problems ...



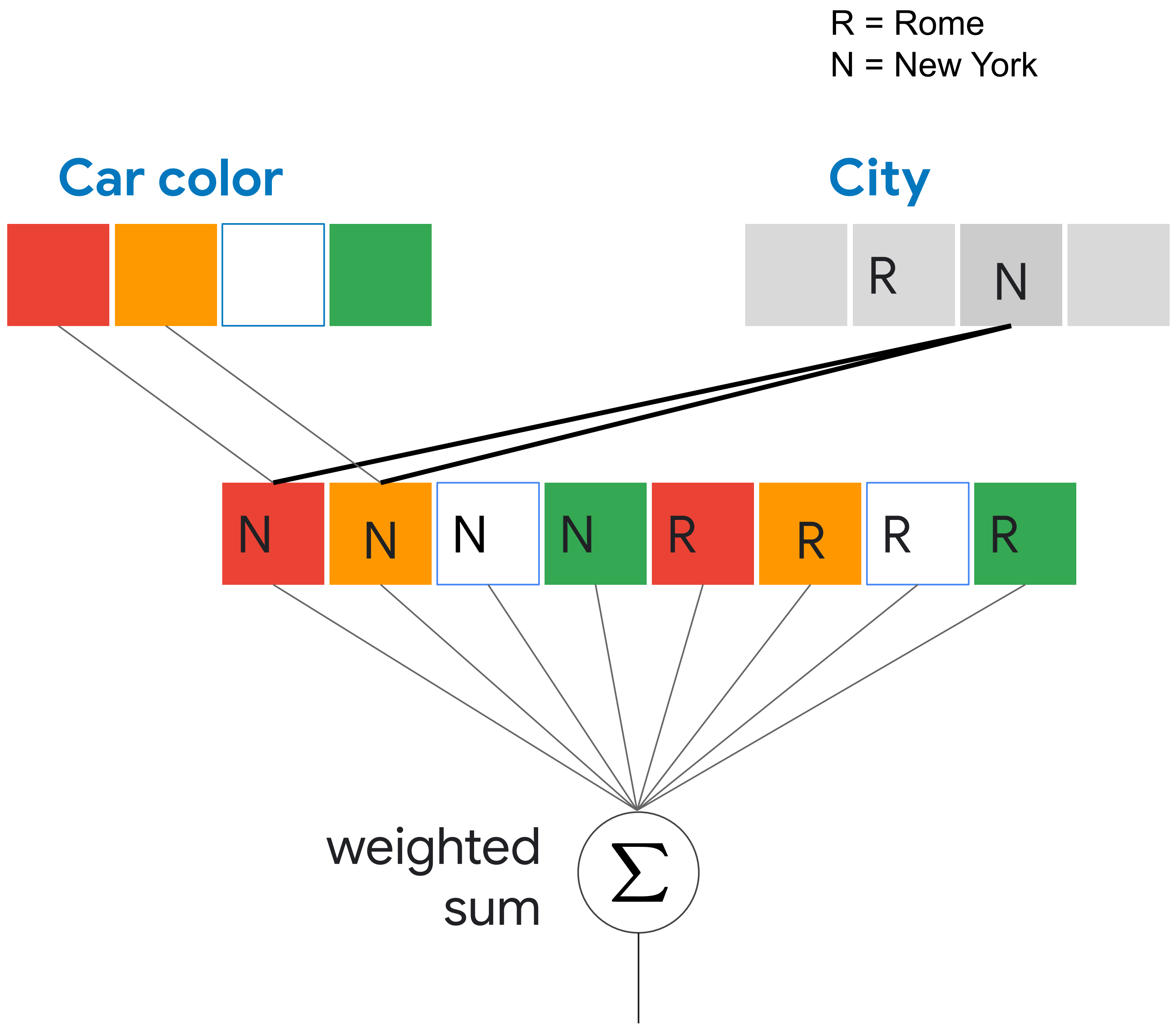
The feature cross has no problem



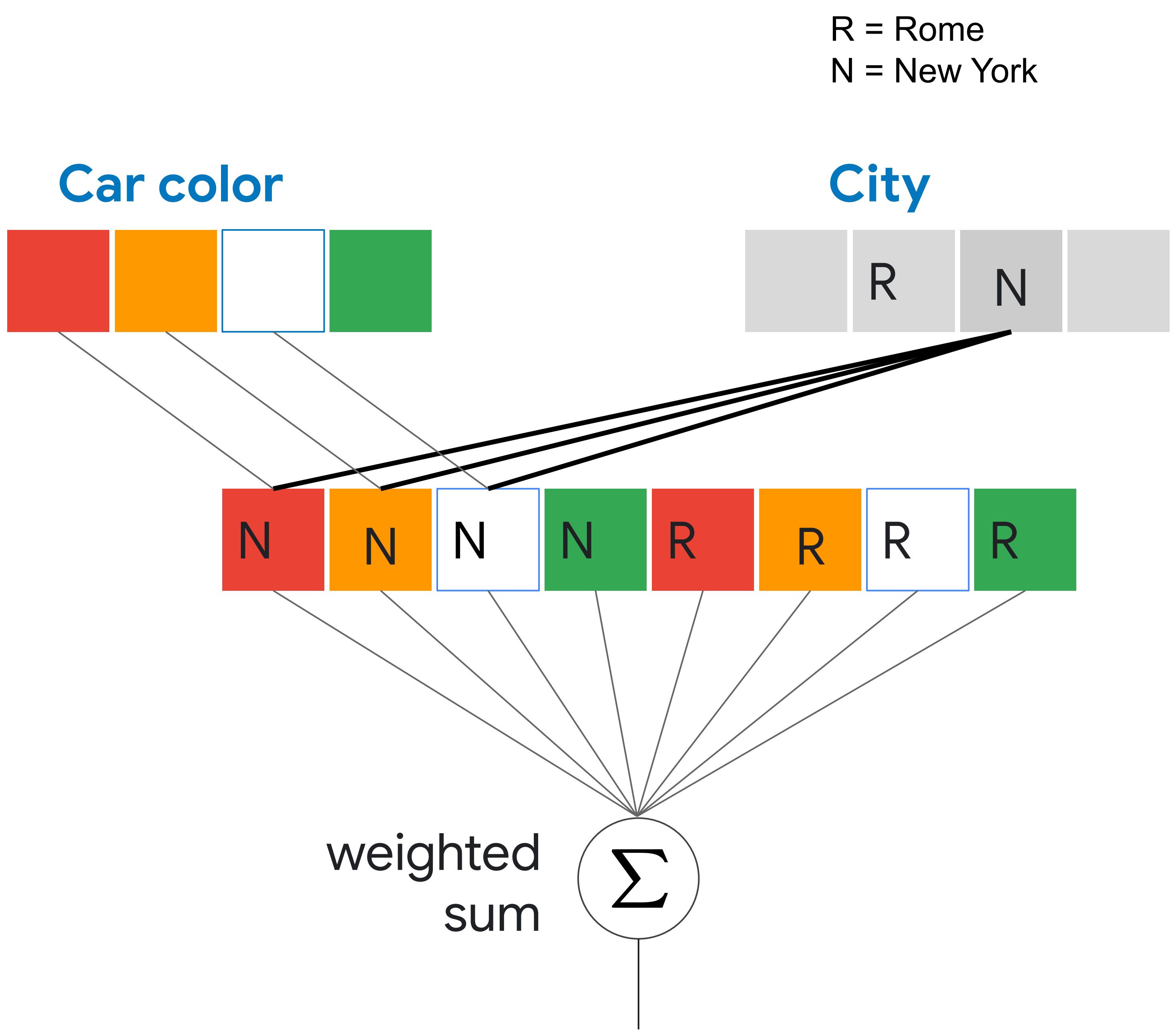
The feature cross has no problem



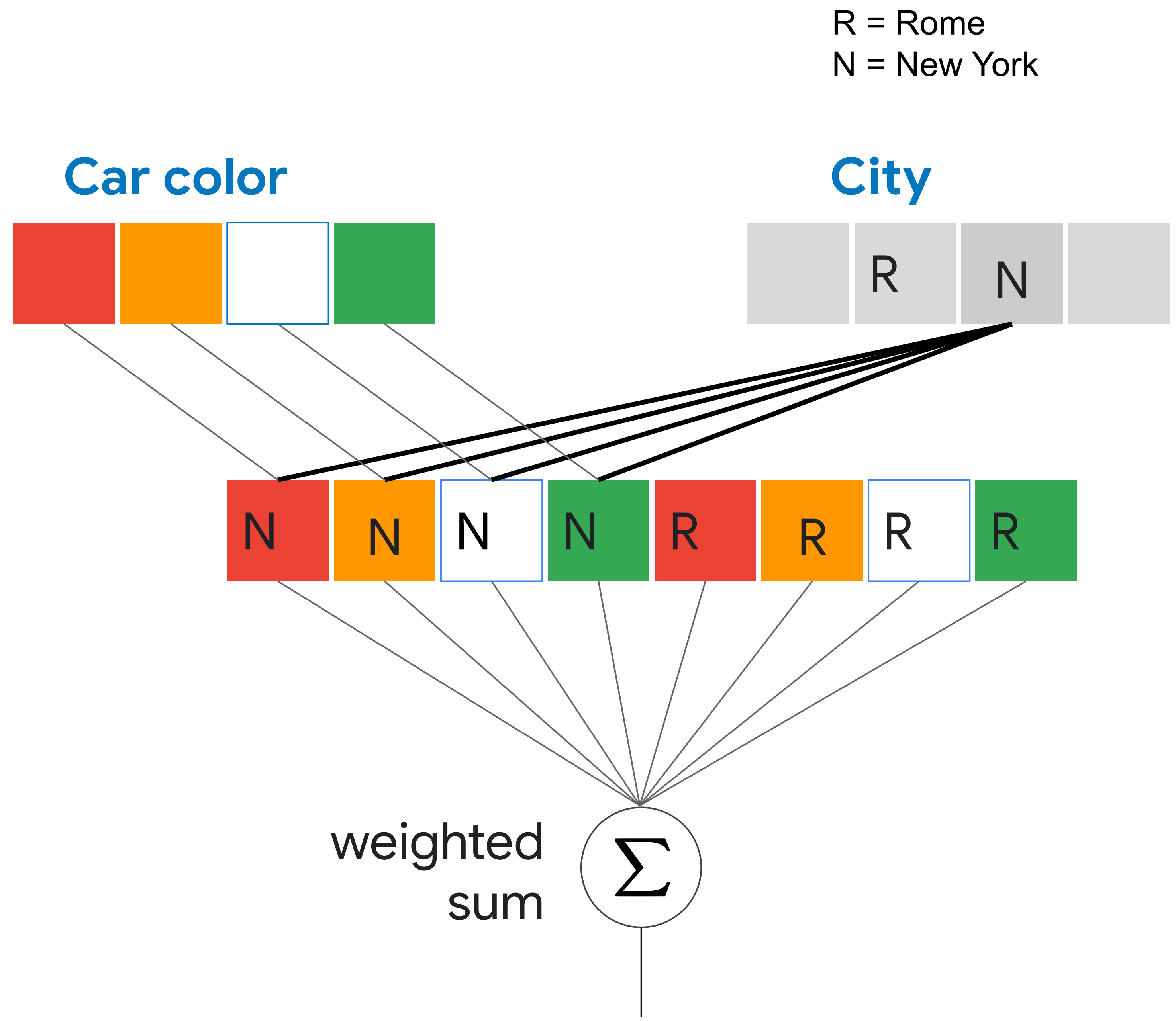
The feature cross has no problem



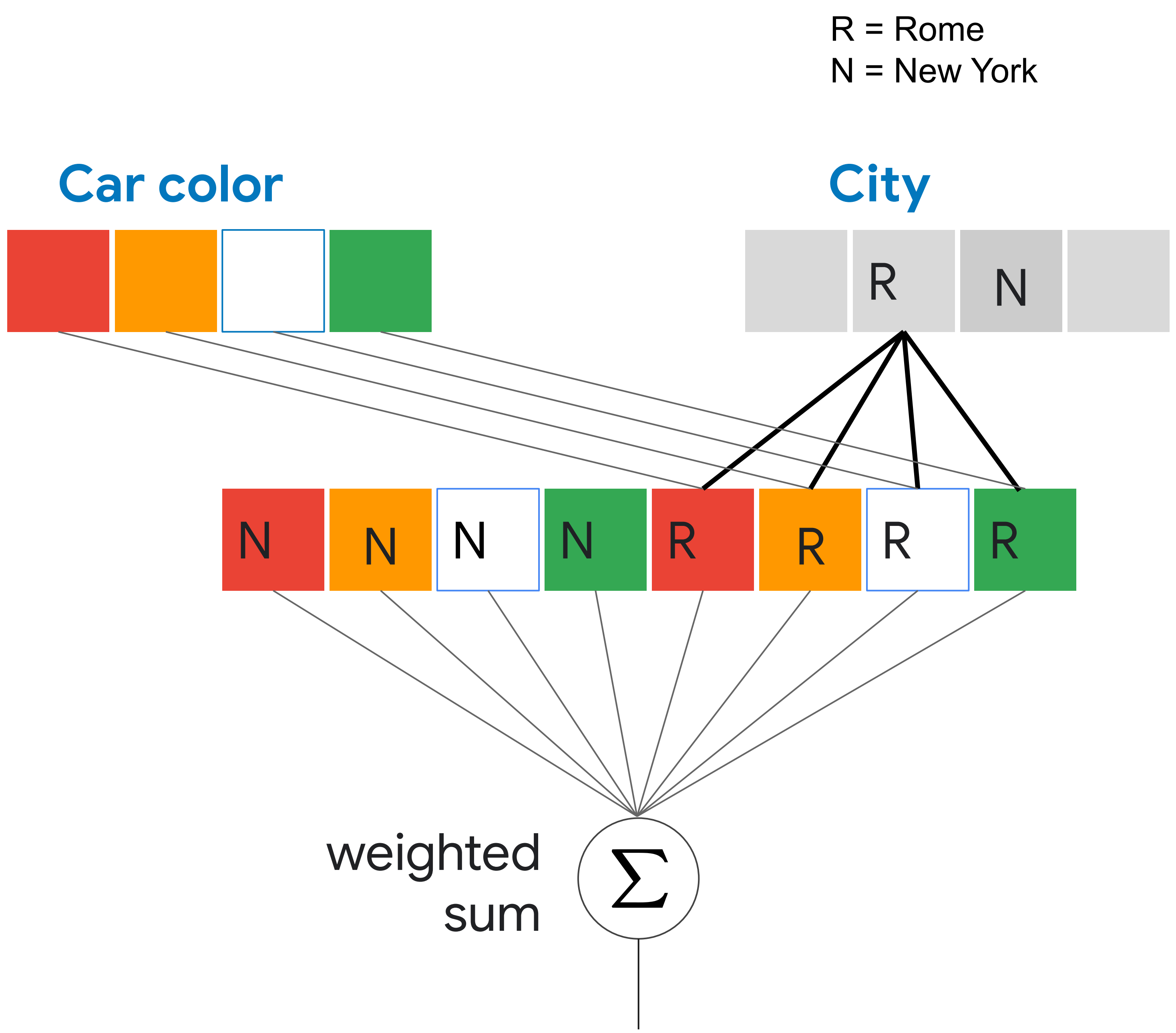
The feature cross has no problem



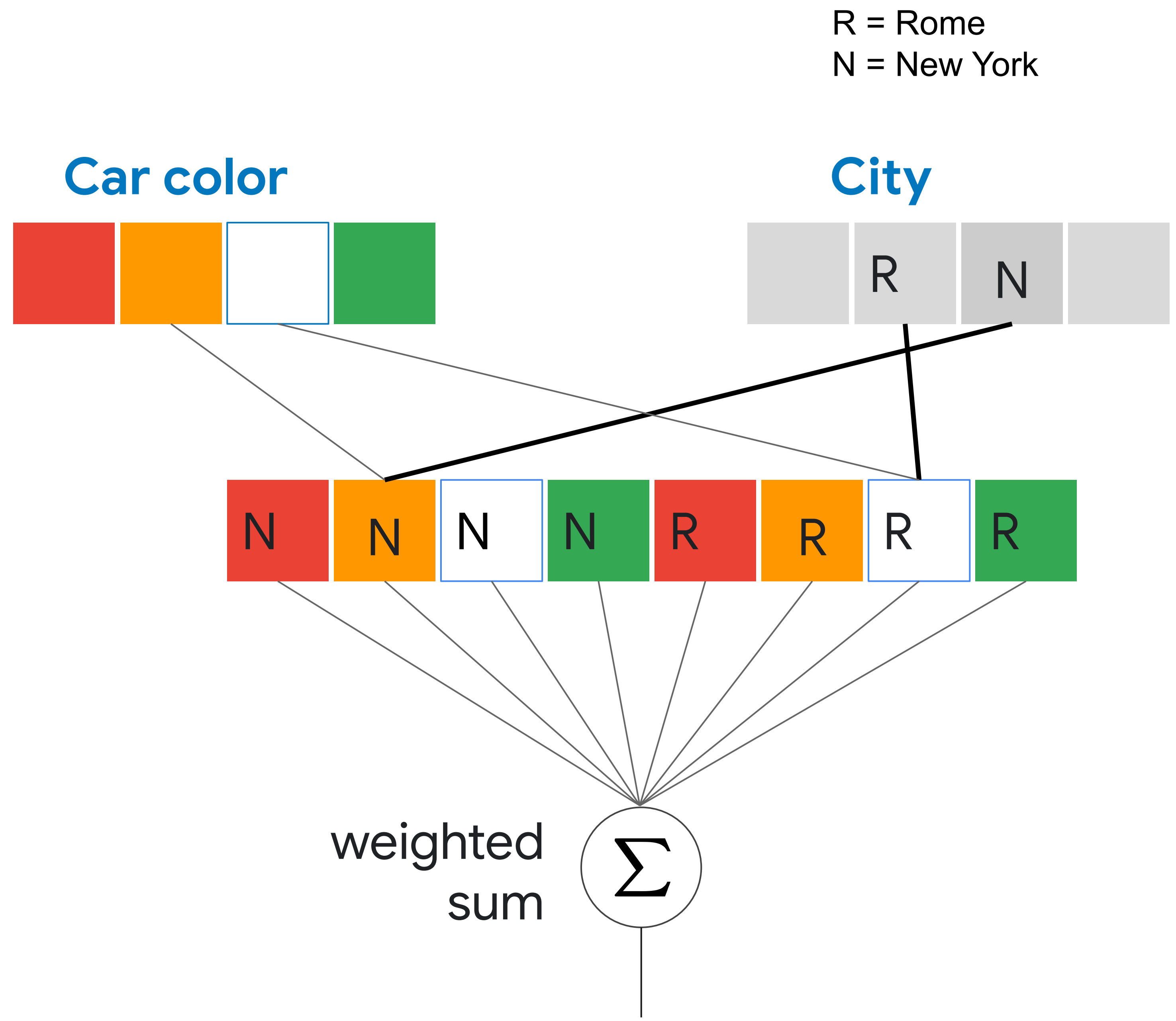
The feature cross has no problem



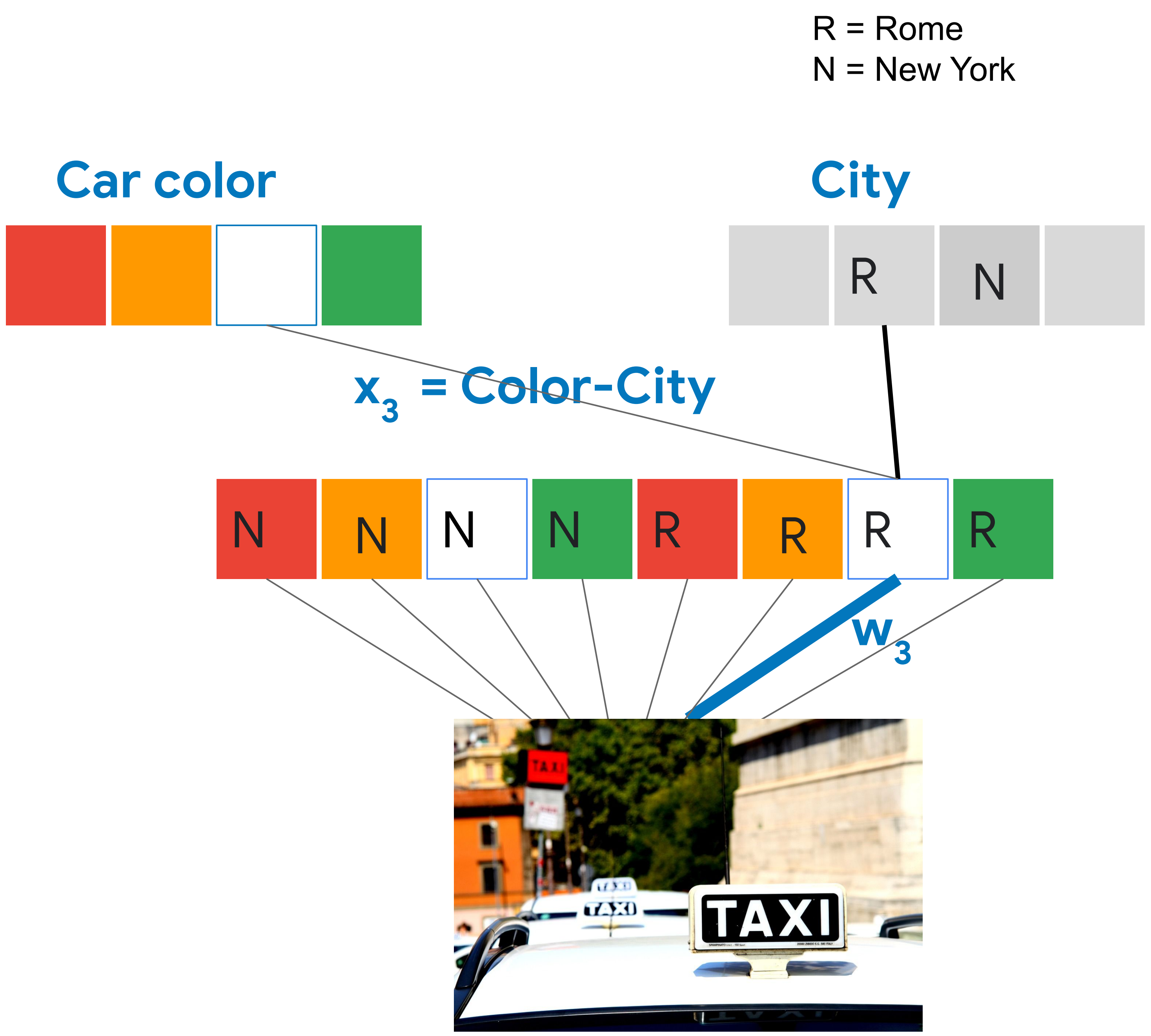
The feature cross has no problem



The feature cross has no problem



The feature cross has no problem



Feature Crosses bring a lot of power to linear models

Feature crosses + massive data is an efficient way for learning highly complex spaces

Feature Crosses bring a lot of power to linear models

Feature crosses + massive data is an efficient way for learning highly complex spaces

Feature crosses allow a linear model to memorize large datasets

Feature Crosses bring a lot of power to linear models

Feature crosses + massive data is an efficient way for learning highly complex spaces

Feature crosses allow a linear model to memorize large datasets

Optimizing linear models is a convex problem

Feature Crosses bring a lot of power to linear models

Feature crosses + massive data is an efficient way for learning highly complex spaces

Feature crosses allow a linear model to memorize large datasets

Optimizing linear models is a convex problem

Before TensorFlow, Google used massive scale learners

Feature Crosses bring a lot of power to linear models

Feature crosses + massive data is an efficient way for learning highly complex spaces

Feature crosses allow a linear model to memorize large datasets

Optimizing linear models is a convex problem

Before TensorFlow, Google used massive scale learners

Feature crosses, as a preprocessor, make neural networks converge a lot quicker

Lab

Use feature crosses to create
a good classifier

Lab: Use feature crosses to create a good classifier

https://goo.gl/2NUCAF

https://goo.gl/ivd4x4

What's the best performance you can get?

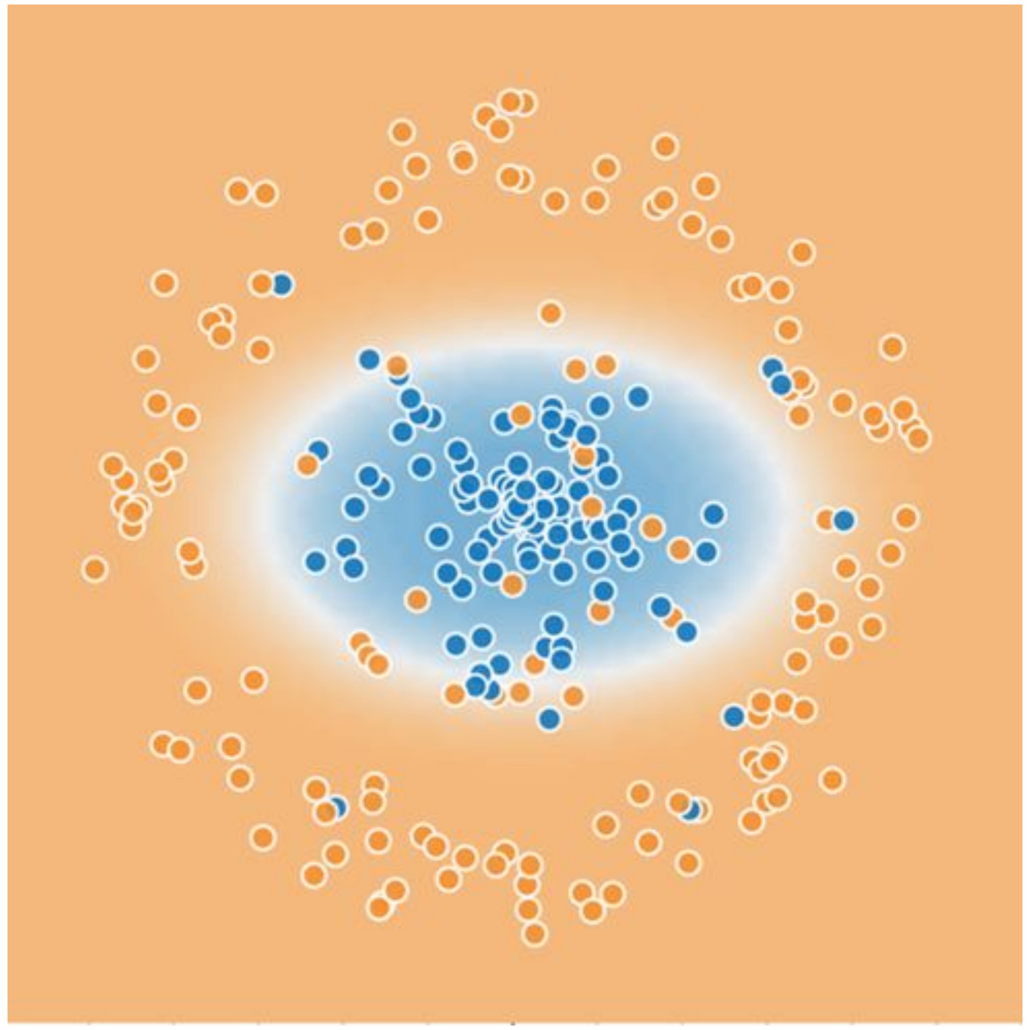
Which feature crosses help the most?

Does the model output surface look like a linear model?

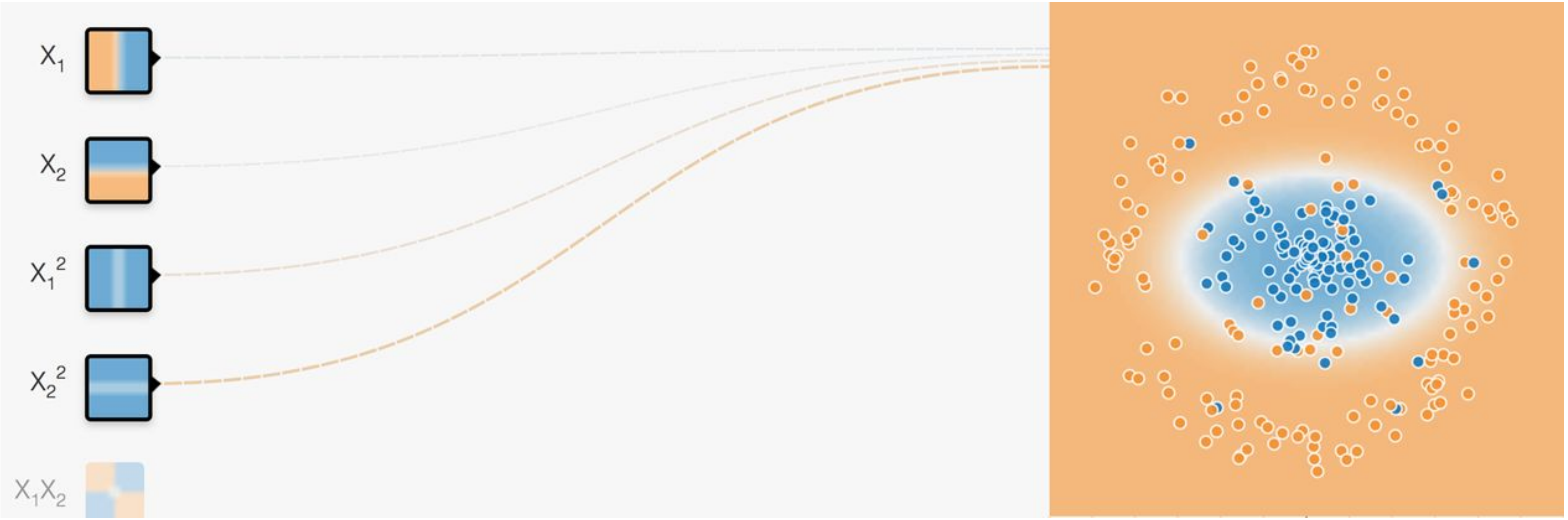
Lab

Screencast

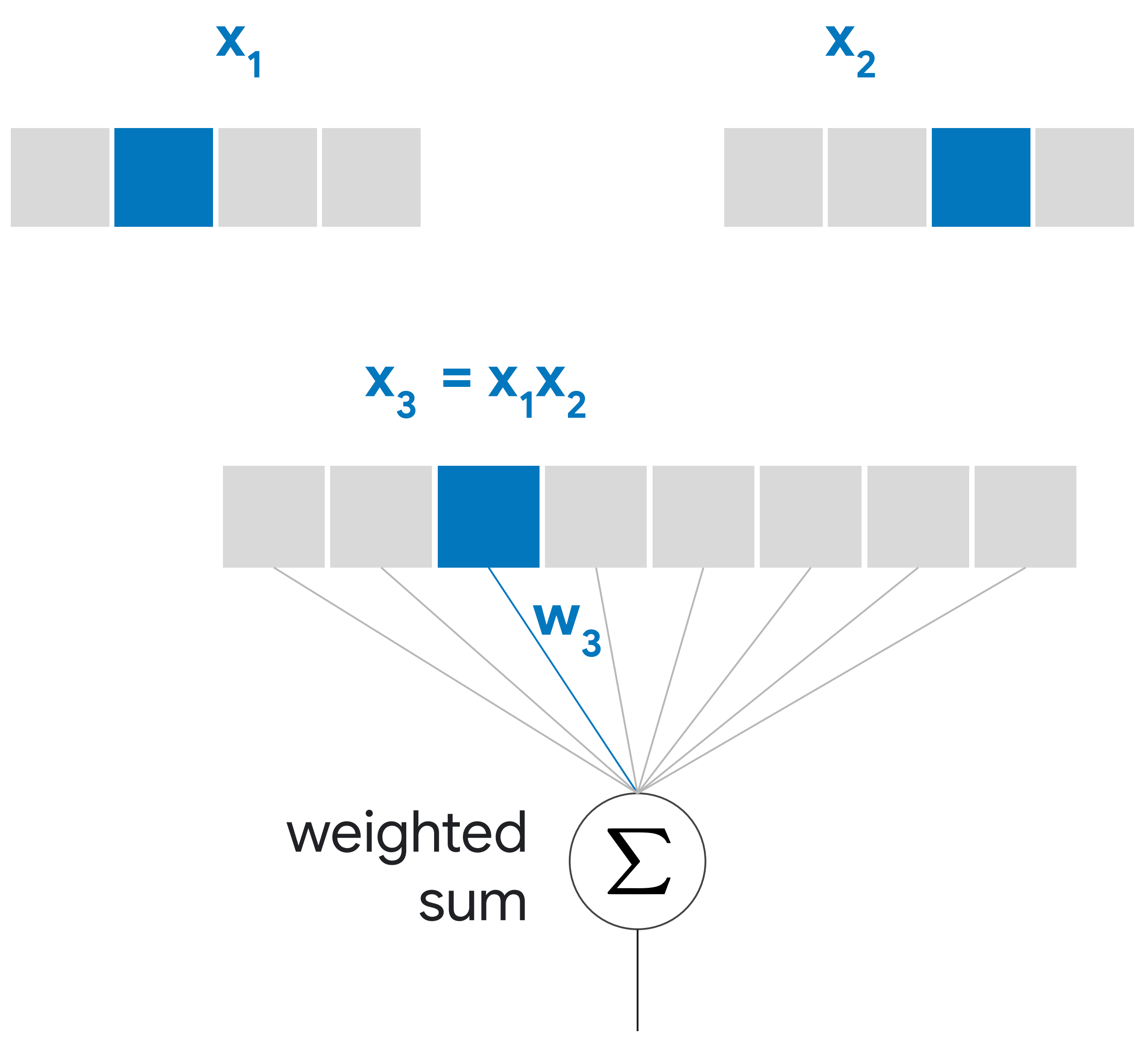
The model boundary doesn't look linear!



The linear decision boundary gets transformed into the curve in the original coordinate space



Feature crosses combine discrete/categorical features

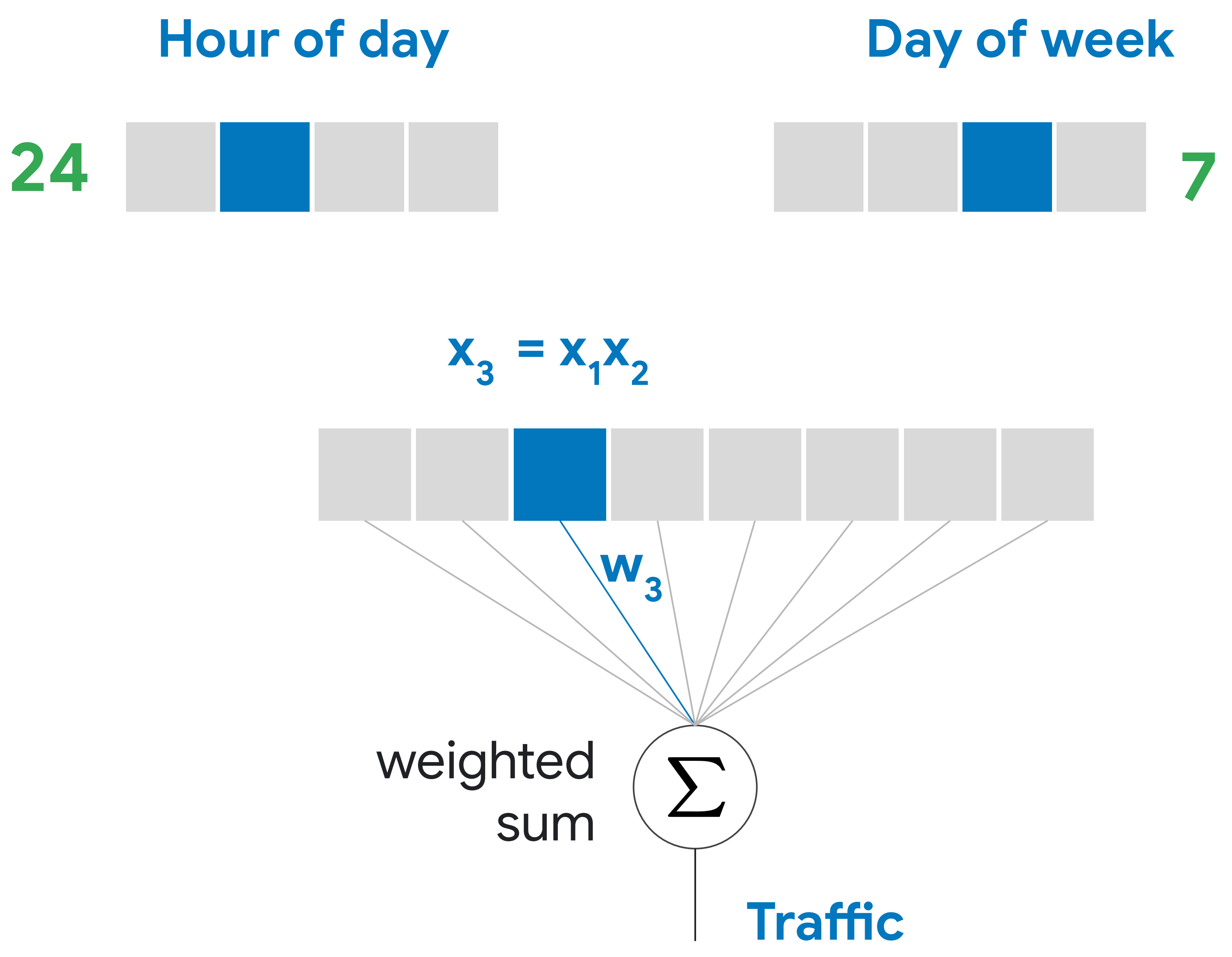


Cross hour_of_day with day_of_week to predict traffic

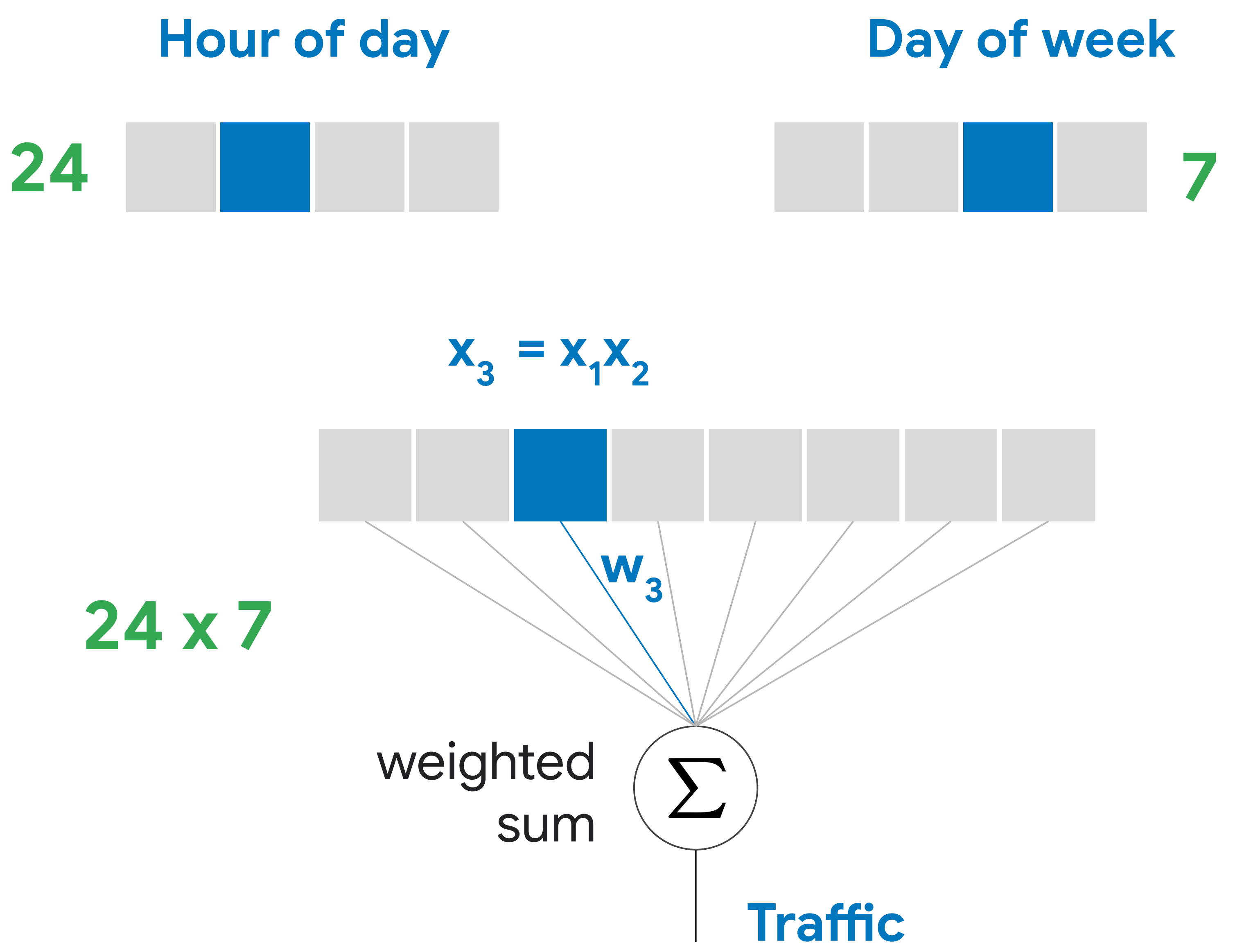
Cross hour_of_day with day_of_week to predict traffic

Cross hour_of_day with day_of_week to predict traffic

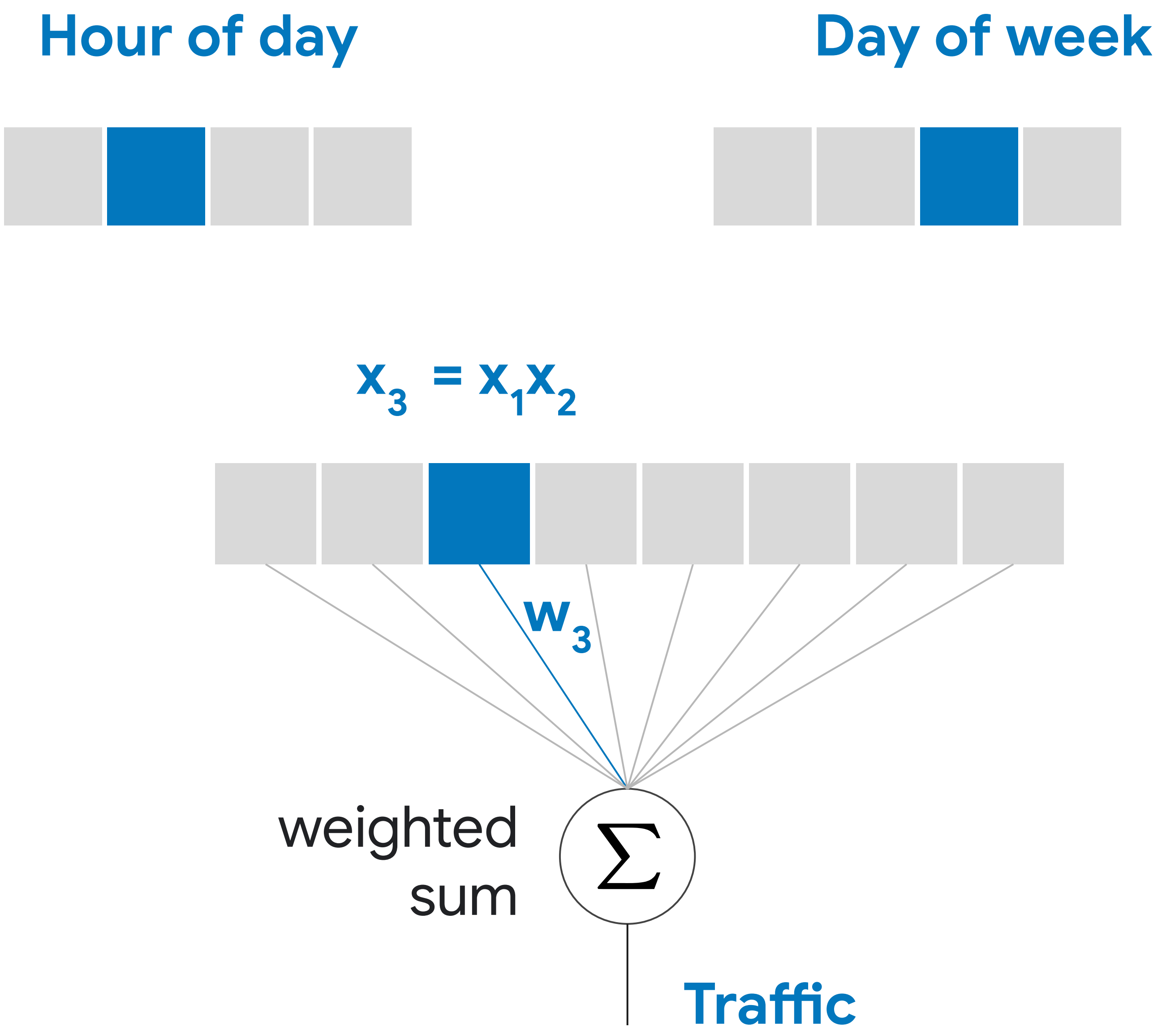
Cross hour_of_day with day_of_week to predict traffic



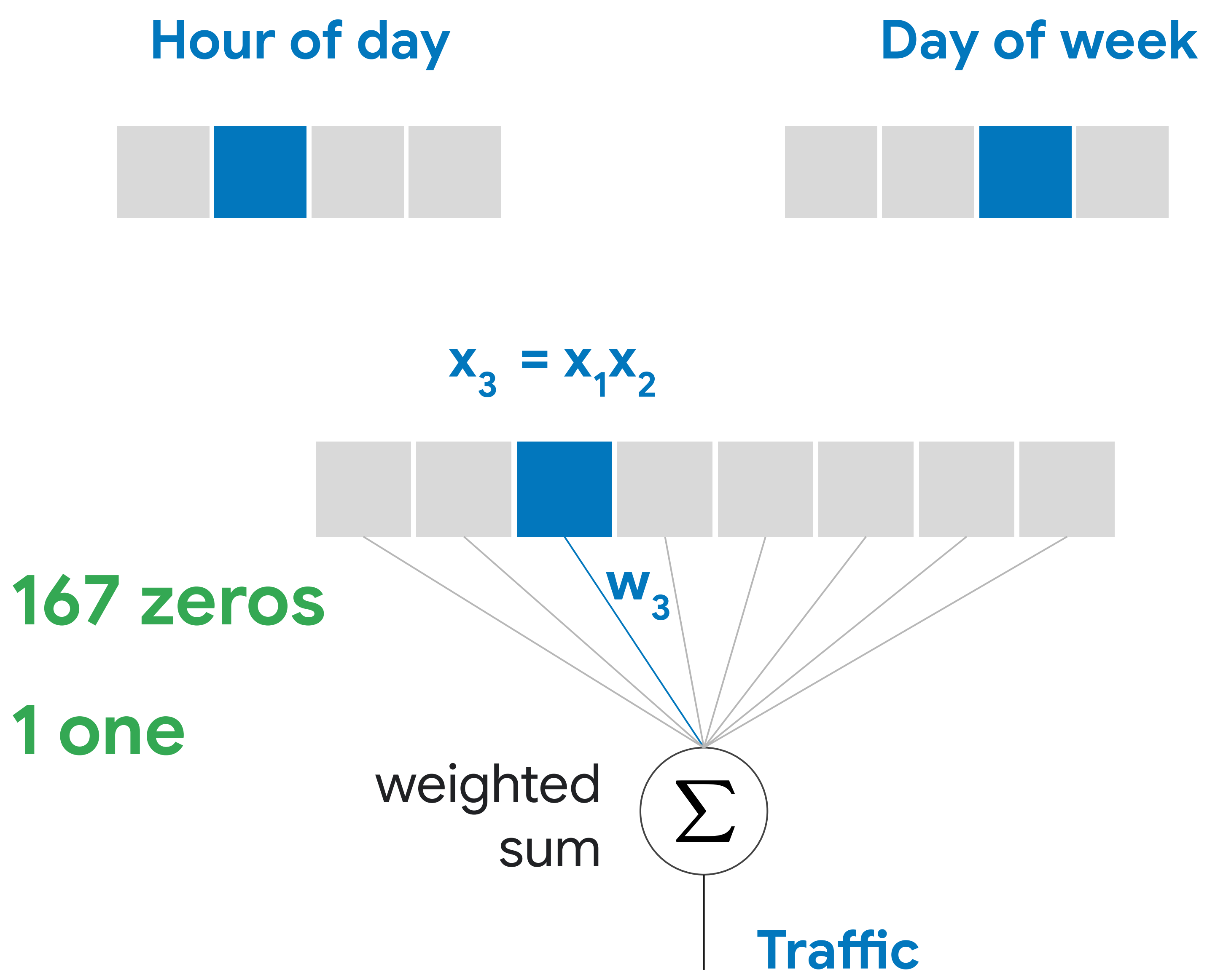
Cross hour_of_day with day_of_week to predict traffic



Feature Crosses lead to sparsity



Feature Crosses lead to sparsity



Quiz: Which of these is a good feature cross?

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 house characteristic and housing price?

a) Three separate binned features: [binned latitude], [binned longitude], [binned roomsPerPerson]
b) Two feature crosses: [binned latitude X binned roomsPerPerson] and [binned longitude X binned roomsPerPerson]
c) One feature cross: [binned latitude X binned longitude X binned roomsPerPerson]
d) One feature cross: [latitude X longitude X roomsPerPerson]

Lab

Too much of a good thing

Lab: Too much of a good thing

https://goo.gl/ofiHCT

Is the model behavior surprising?

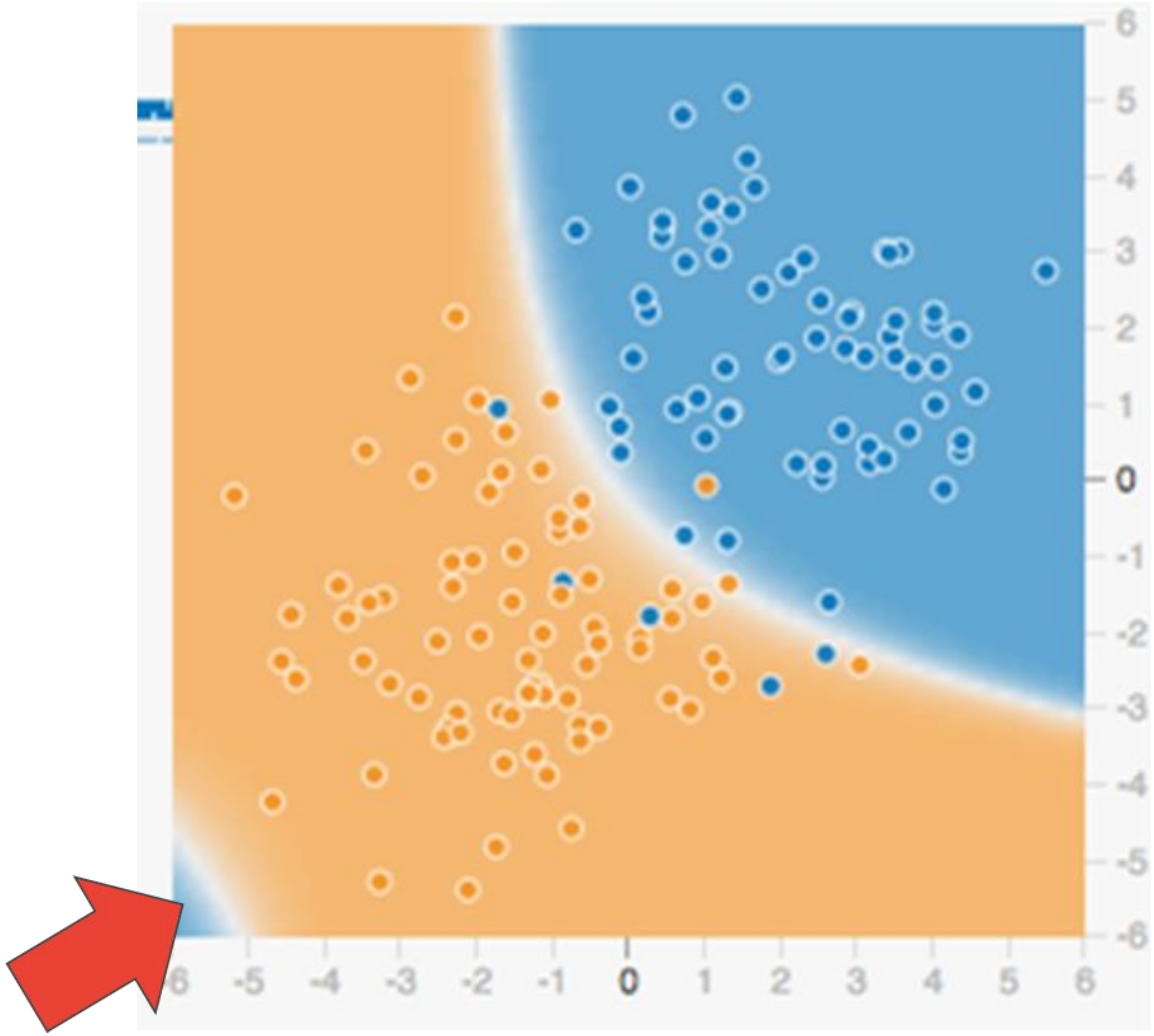
What's the issue?

Try removing cross-product features. Does performance improve?

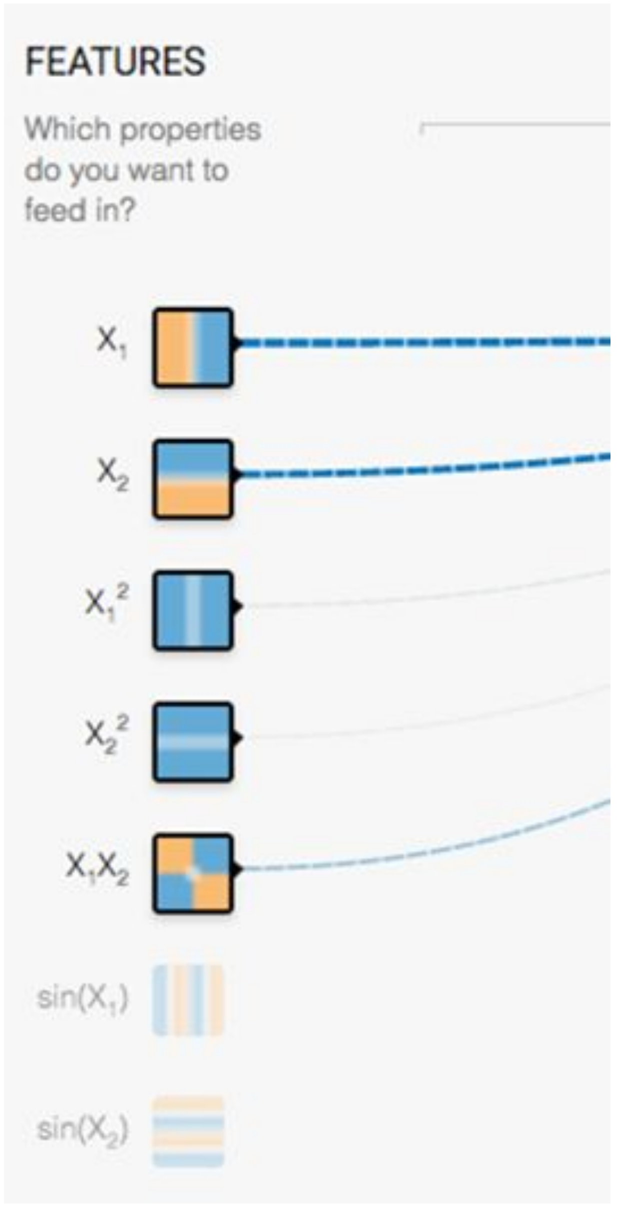
Lab

Screencast

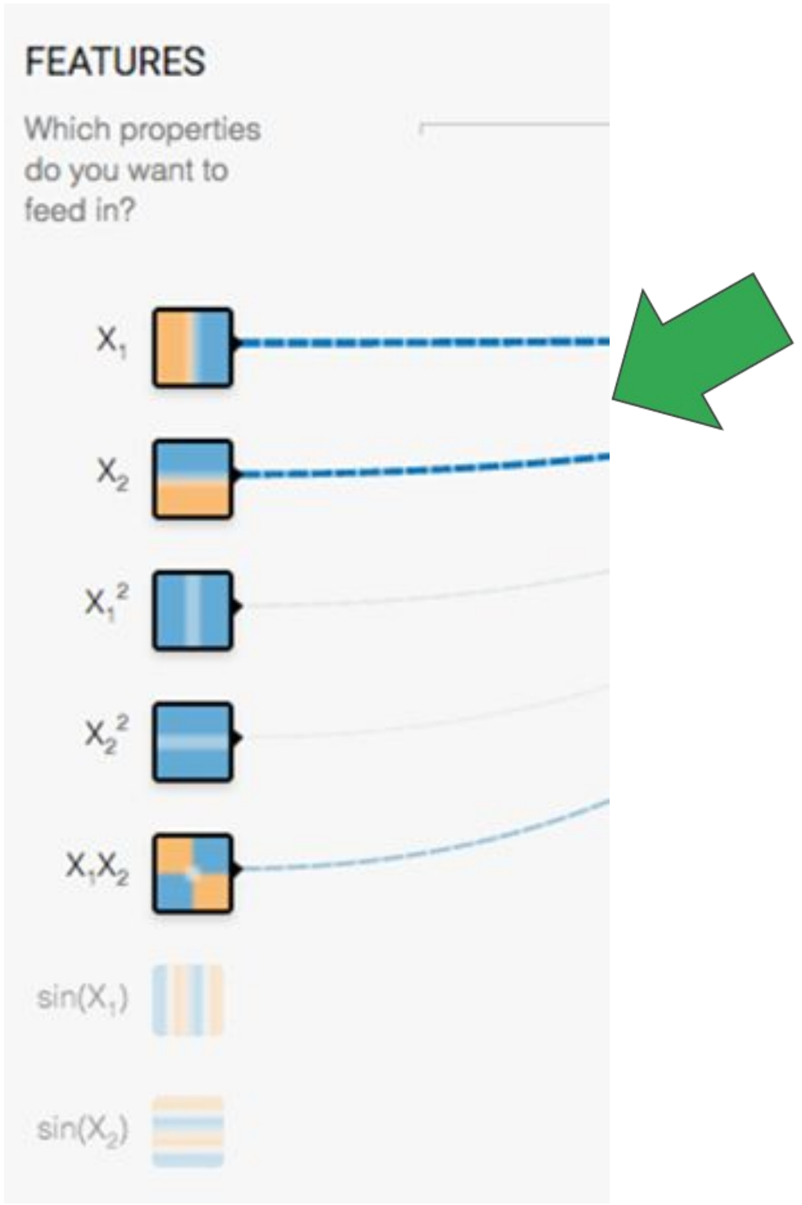
Lab: Too much of a good thing



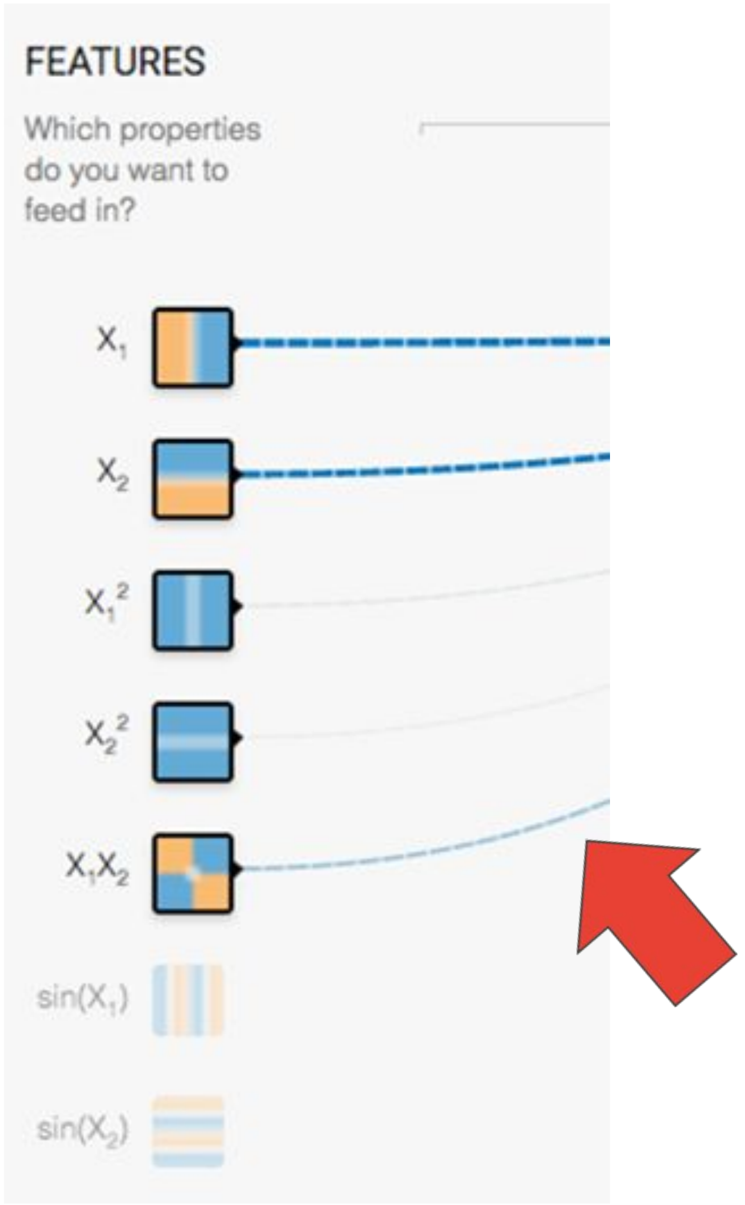
Lab: Too much of a good thing



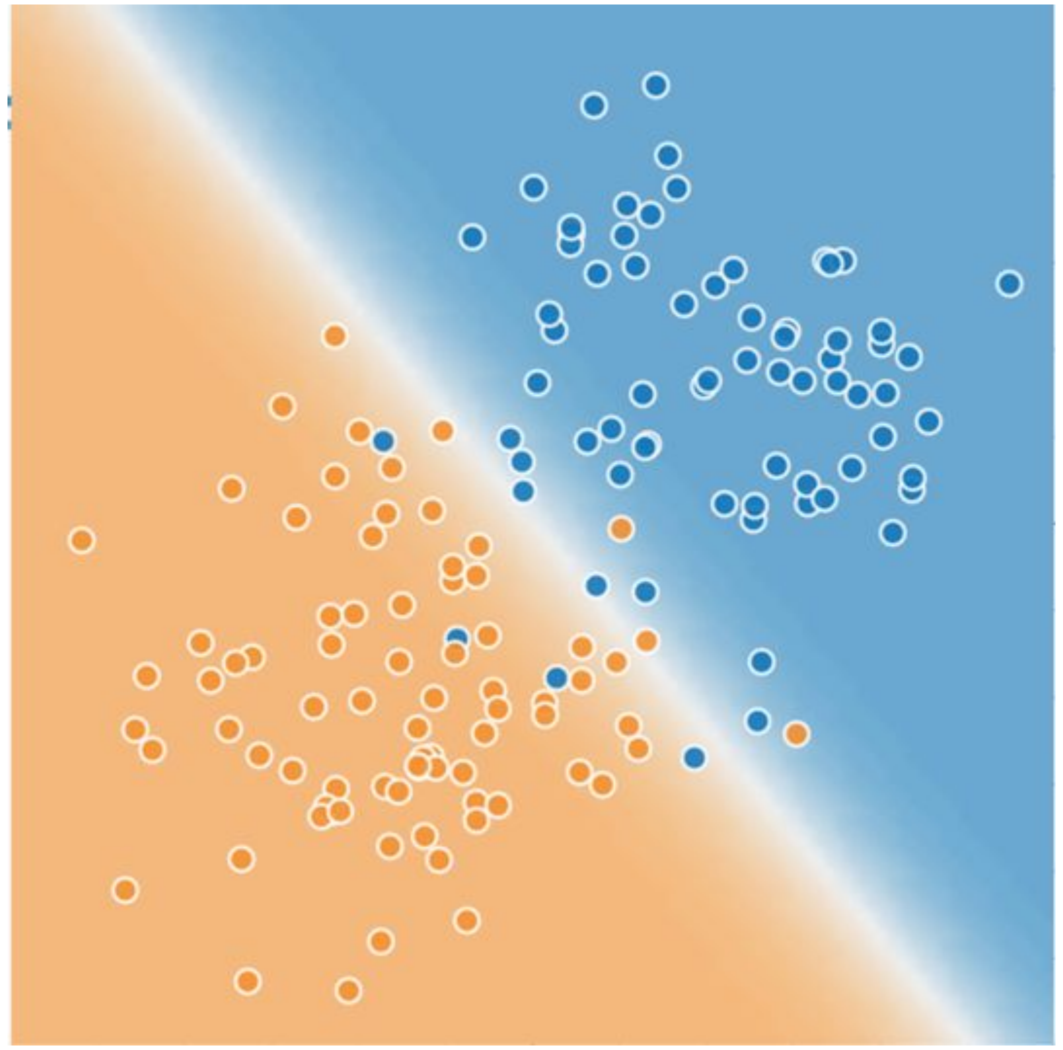
Lab: Too much of a good thing



Lab: Too much of a good thing



After removing the feature crosses ...



Implementing feature crosses

First Lastname

Creating feature crosses using TensorFlow

```
day_hr =
tf.feature_column.crossed_column(
   [dayofweek, hourofday],
   24 * 7)
```

Creating feature crosses using TensorFlow

```
day_hr =
tf.feature_column.crossed_column(
   [dayofweek, hourofday],
   24 * 7)
```

You can cross two or more categorical or bucketized columns

Creating feature crosses using TensorFlow

```
day_hr =
tf.feature_column.crossed_column(
    [dayofweek, hourofday],
    24 * 7)
```

This is the number
of hash buckets:
feature-cross %
hash_bucket_size

Choosing the number of hash buckets is an art, not a science

```
day_hr =
tf.feature_column.crossed_column(
   [dayofweek, hourofday],
   24 * 7)
```

Choosing the number of hash buckets is an art, not a science

```
day_hr =
tf.feature_column.crossed_column(
   [dayofweek, hourofday],
     6
```

Choosing the number of hash buckets is an art, not a science

```
day_hr =
tf.feature_column.crossed_column(
   [dayofweek, hourofday],
     6
```

feature-cross % hash_bucket_size

3pm->15   Wed->3

3pm X Wed -> 15 + 3*24 = 87

Choosing the number of hash buckets is an art, not a science

```
day_hr =
tf.feature_column.crossed_column(
   [dayofweek, hourofday],
     6
```

feature-cross % hash_bucket_size

hash(87) % 6 = 3

3pm->15   Wed->3

3pm X Wed -> 15 + 3*24 = 87

The number of hash buckets controls sparsity and collisions

Small hash_buckets -> lots of collisions

The number of hash buckets controls sparsity and collisions

Small hash_buckets -> lots of collisions

High hash_buckets -> very sparse

The number of hash buckets controls sparsity and collisions

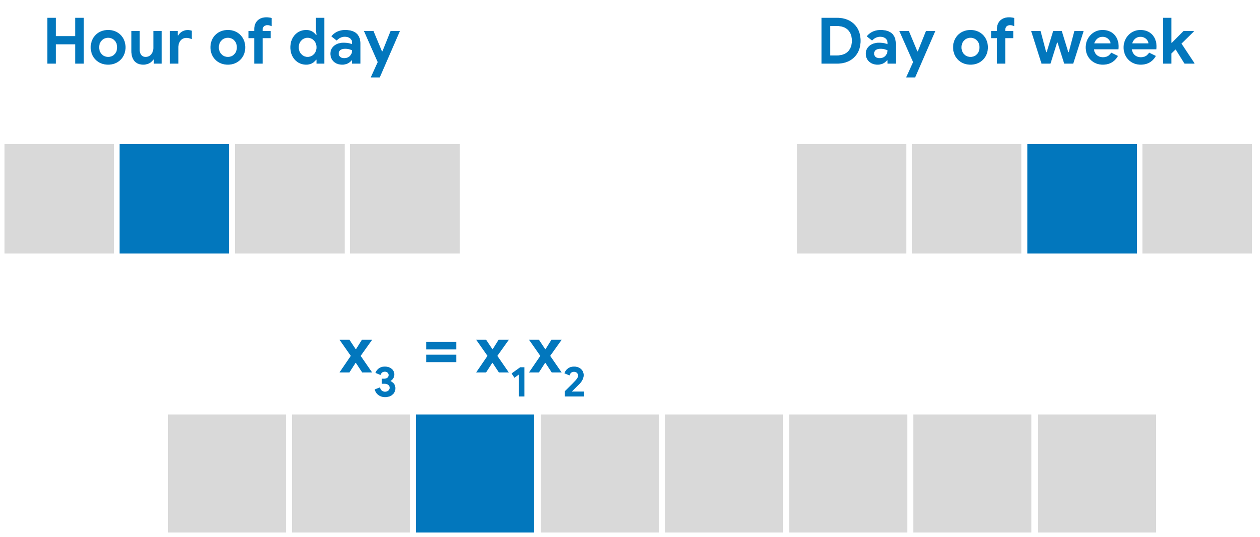
Small hash_buckets -> lots of collisions

½ sqrt(N)

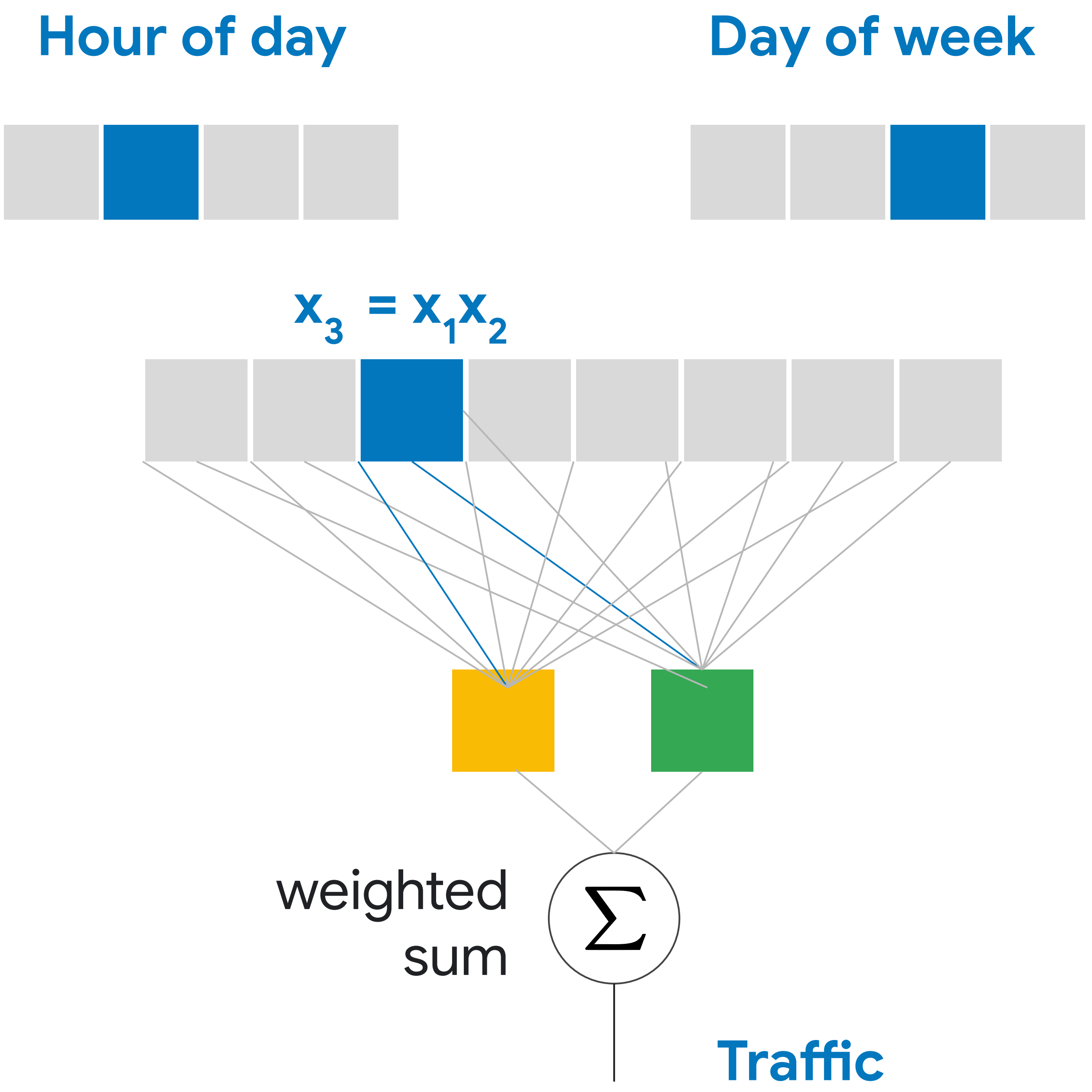
High hash_buckets -> very sparse

2N

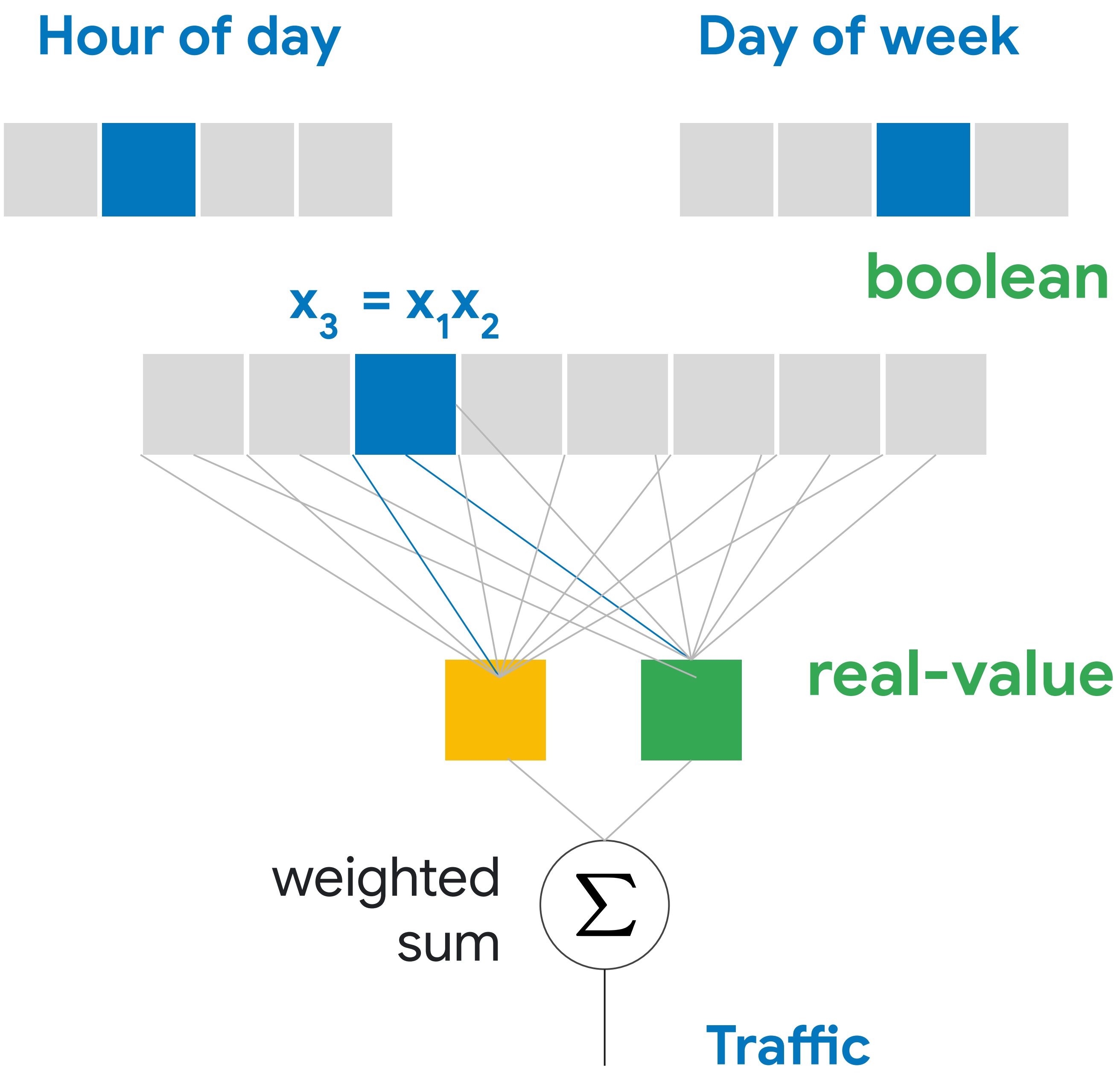
Creating an embedding column from a feature cross



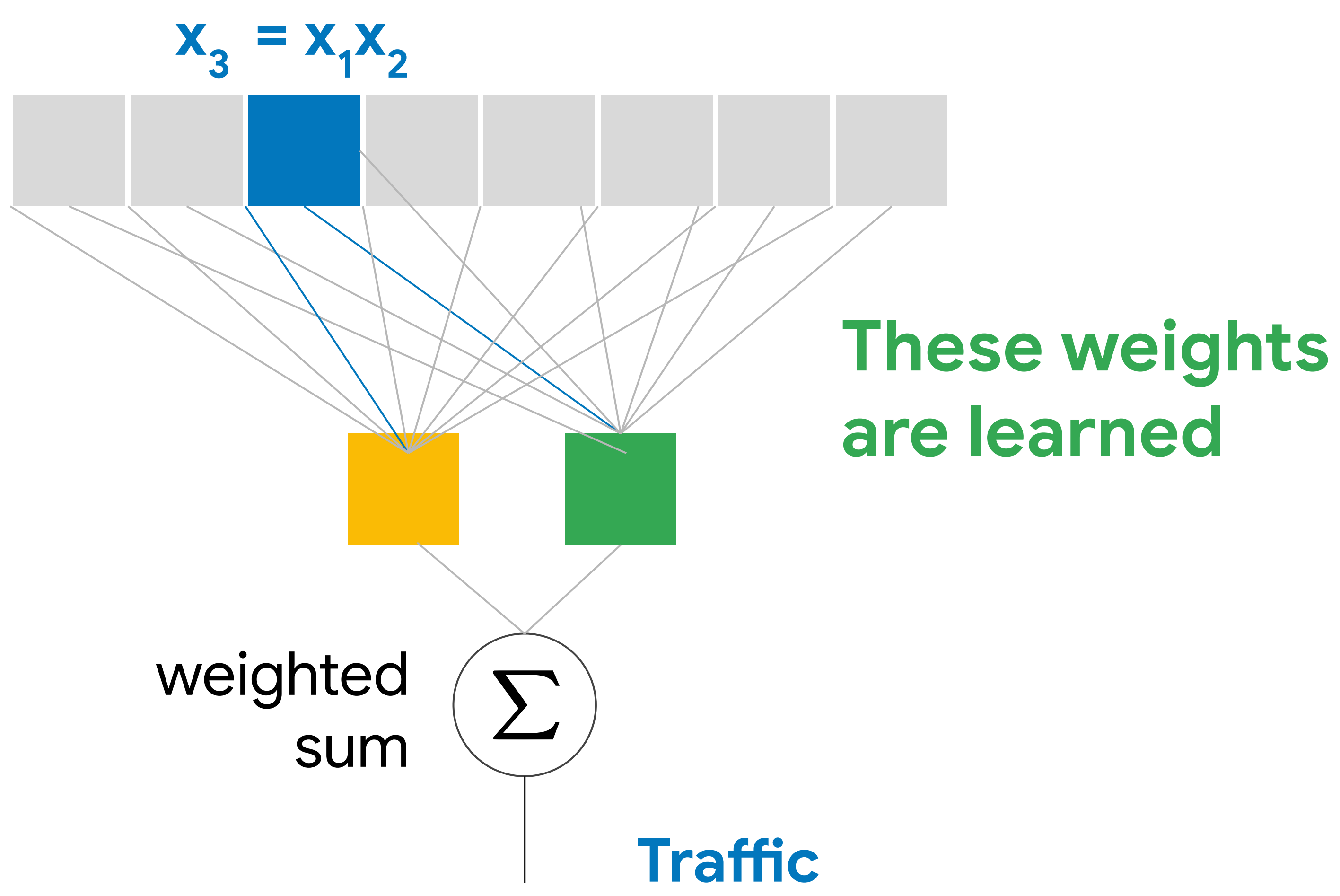
Creating an embedding column from a feature cross



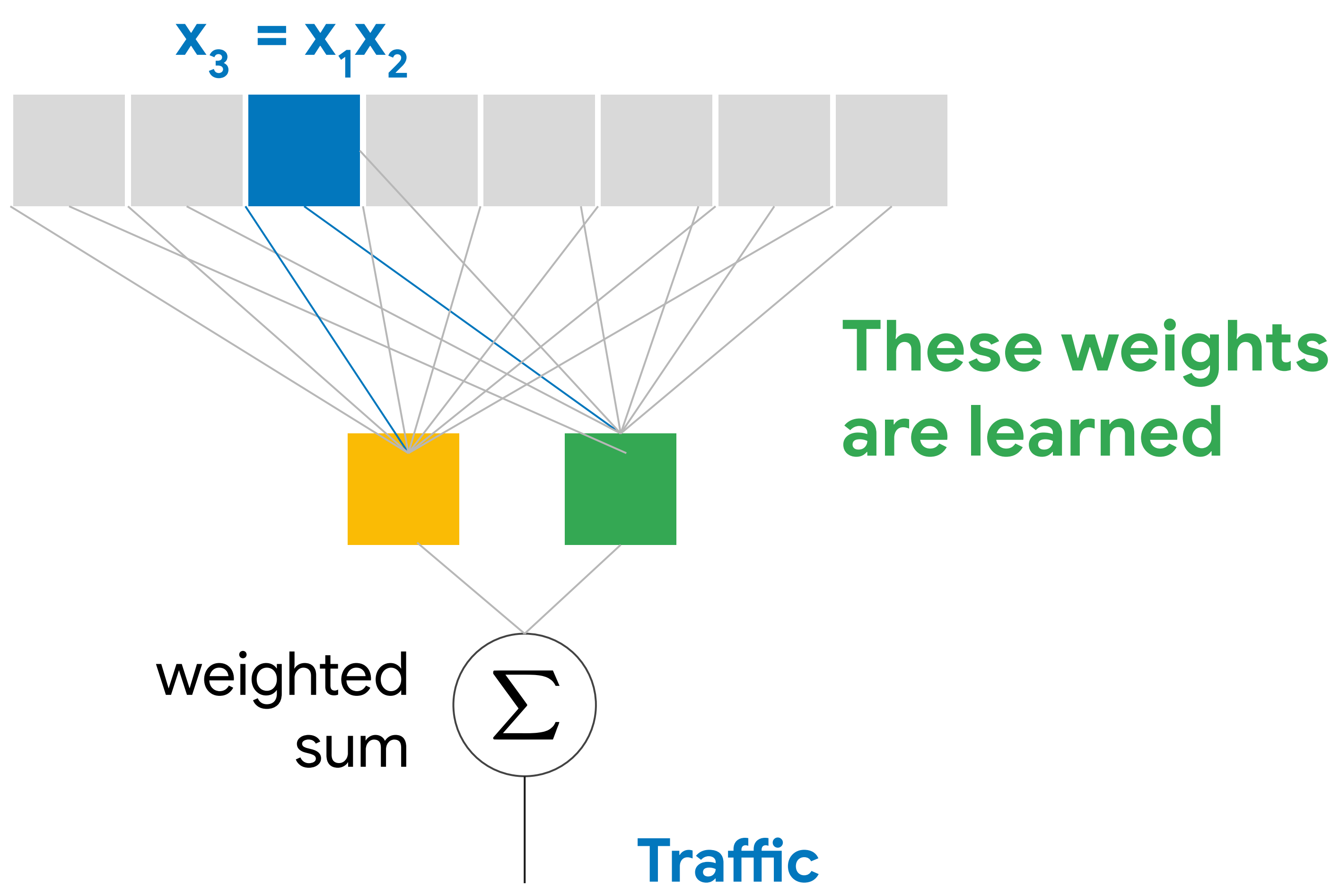
Creating an embedding column from a feature cross



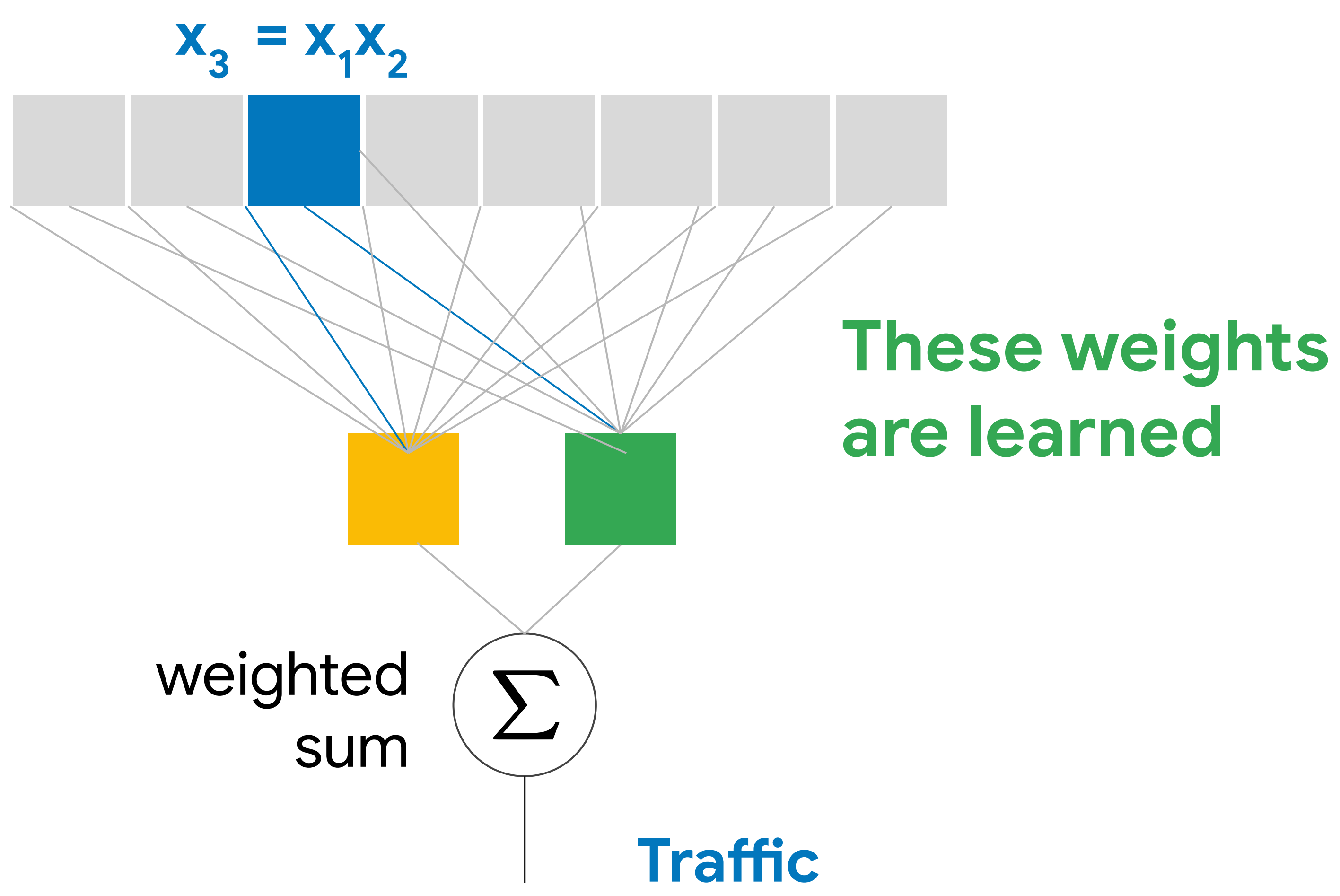
The weights in the embedding column are learned from data



The weights in the embedding column are learned from data



The weights in the embedding column are learned from data



The model learns how to embed the feature cross in lower-dimensional space

8am Tue	0.8	0.7
9am Wed	0.7	0.9
11 am Tue	0.1	0.6
2 pm Wed	0.1	0.7
2 am Tue	0	0.1
2 am Wed	0	0.1

The model learns how to embed the feature cross in lower-dimensional space

8am Tue	0.8	0.7
9am Wed	0.7	0.9
11 am Tue	0.1	0.6
2 pm Wed	0.1	0.7
2 am Tue	0	0.1
2 am Wed	0	0.1

similar

The model learns how to embed the feature cross in lower-dimensional space

8am Tue	0.8	0.7
9am Wed	0.7	0.9
11 am Tue	0.1	0.6
2 pm Wed	0.1	0.7
2 am Tue	0	0.1
2 am Wed	0	0.1

different

similar

The model learns how to embed the feature cross in lower-dimensional space

8am Tue	0.8	0.7
9am Wed	0.7	0.9
11 am Tue	0.1	0.6
2 pm Wed	0.1	0.7
2 am Tue	0	0.1
2 am Wed	0	0.1

similar

The model learns how to embed the feature cross in lower-dimensional space

8am Tue	0.8	0.7
9am Wed	0.7	0.9
11 am Tue	0.1	0.6
2 pm Wed	0.1	0.7
2 am Tue	0	0.1
2 am Wed	0	0.1

Embedding a feature cross in TensorFlow

```
import tf.feature_column as fc

day_hr = fc.crossed_column(
   [dayofweek, hourofday],
   24x7 )

day_hr_em = fc.embedding_column(
      day_hr,
      2,)
```

Transfer Learning of embeddings from similar ML models

```
import tf.feature_column as fc

day_hr = fc.crossed_column(
   [dayofweek, hourofday],
   24x7 )

day_hr_em = fc.embedding_column(
      day_hr,
      2,
  ckpt_to_load_from='london/*ckpt-1000*’,
  tensor_name_in_ckpt='dayhr_embed’,
  trainable=False
 )
```

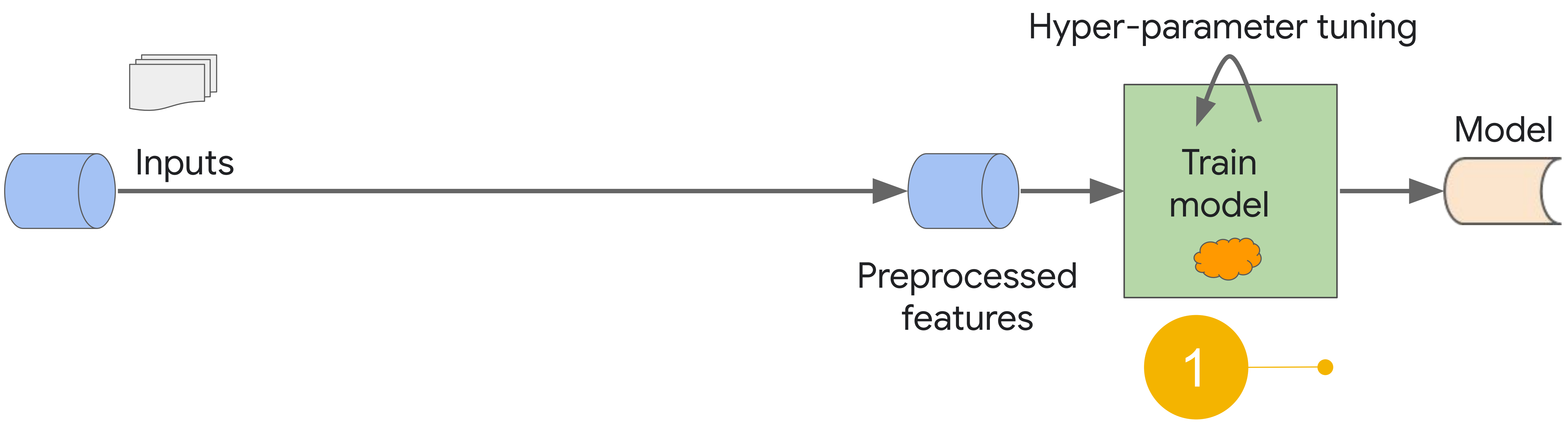
Where does the feature engineering code fit in?

```
def train_input_fn(file_prefix):
  ...
  return features, labels

featcols = [
   fc.numeric_column("sq_footage"),
   fc.categorical_column_with_vocabulary_list(
            "type", ["house", "apt"])
]

model = tf.estimator.LinearRegressor(featcols)

train_spec, eval_spec = ...

model.train_and_evaluate(train_spec, ...)
```

Three possible places to do feature engineering

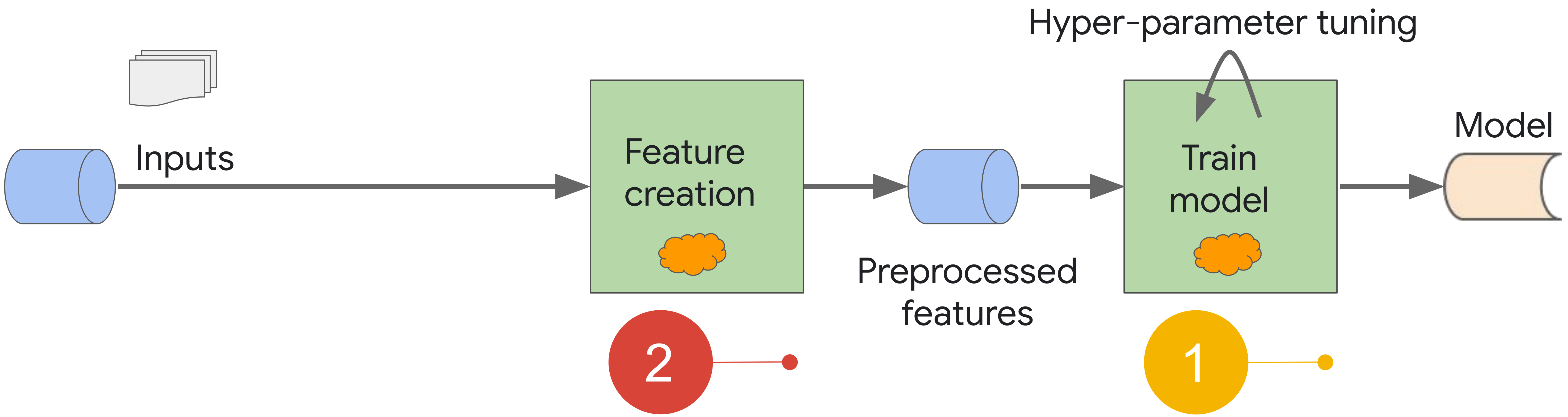
TensorFlow
feature_column
input_fn

Three possible places to do feature engineering



TensorFlow
feature_column
input_fn

Three possible places to do feature engineering

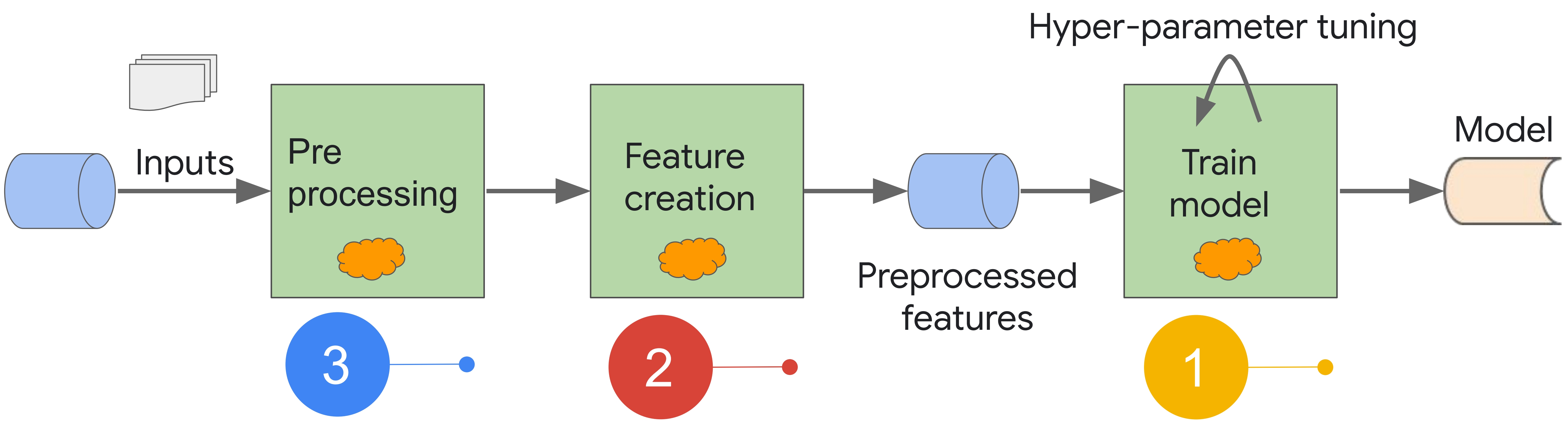


Dataflow

*If Dataflow is part of your prediciton runtime also

TensorFlow
feature_column
input_fn

Three possible places to do feature engineering



Dataflow + TensorFlow (tf.transform)

Dataflow

*If Dataflow is part of your prediciton runtime also

TensorFlow feature_column input_fn

Some preprocessing can be done in tf.feature_column

```
def train_input_fn(file_prefix):
  ...
  return features, labels

featcols = [
   fc.numeric_column("sq_footage"),
   fc.categorical_column_with_vocabulary_list(
            "type", ["house", "apt"])
]

featcols.append(
  fc.bucketized_column(featcols[0],
                       [500, 1000, 2500]))

model = tf.estimator.LinearRegressor(featcols)

train_spec, eval_spec = ...
model.train_and_evaluate(train_spec, ...)
```

1

Some preprocessing can be done in tf.feature_column

```
def train_input_fn(file_prefix):
  ...
  return features, labels

featcols = [
   fc.numeric_column("sq_footage"),
   fc.categorical_column_with_vocabulary_list(
            "type", ["house", "apt"])
]

featcols [0] = (
  fc.bucketized_column(featcols[0],
                       [500, 1000, 2500]))

model = tf.estimator.LinearRegressor(featcols)

train_spec, eval_spec = ...
model.train_and_evaluate(train_spec, ...)
```

1

Powerful preprocessing can be done in TensorFlow by creating a new feature column

1

```
latbuckets = np.linspace(38.0, 42.0, nbuckets).tolist()
lonbuckets = np.linspace(-76.0, -72.0, nbuckets).tolist()
```

Preprocessing is part of the model graph,
so “automatic” at prediction time

Powerful preprocessing can be done in TensorFlow by creating a new feature column

1

```
latbuckets = np.linspace(38.0, 42.0, nbuckets).tolist()
lonbuckets = np.linspace(-76.0, -72.0, nbuckets).tolist()

b_lat = fc.bucketized_column(house_lat, latbuckets)
b_lon = fc.bucketized_column(house_lon, lonbuckets)
```

Preprocessing is part of the model graph,
so “automatic” at prediction time

Powerful preprocessing can be done in TensorFlow by creating a new feature column

1

```
latbuckets = np.linspace(38.0, 42.0, nbuckets).tolist()
lonbuckets = np.linspace(-76.0, -72.0, nbuckets).tolist()

b_lat = fc.bucketized_column(house_lat, latbuckets)
b_lon = fc.bucketized_column(house_lon, lonbuckets)

# feature cross and embed
loc = fc.crossed_column([b_lat, b_lon], nbuckets*nbuckets)
```

Preprocessing is part of the model graph,
so “automatic” at prediction time

1 Powerful preprocessing can be done in TensorFlow by creating a new feature column

```
latbuckets = np.linspace(38.0, 42.0, nbuckets).tolist()
lonbuckets = np.linspace(-76.0, -72.0, nbuckets).tolist()

b_lat = fc.bucketized_column(house_lat, latbuckets)
b_lon = fc.bucketized_column(house_lon, lonbuckets)

# feature cross and embed
loc = fc.crossed_column([b_lat, b_lon], nbuckets*nbuckets)

eloc = fc.embedding_column(loc, nbuckets//4)
```

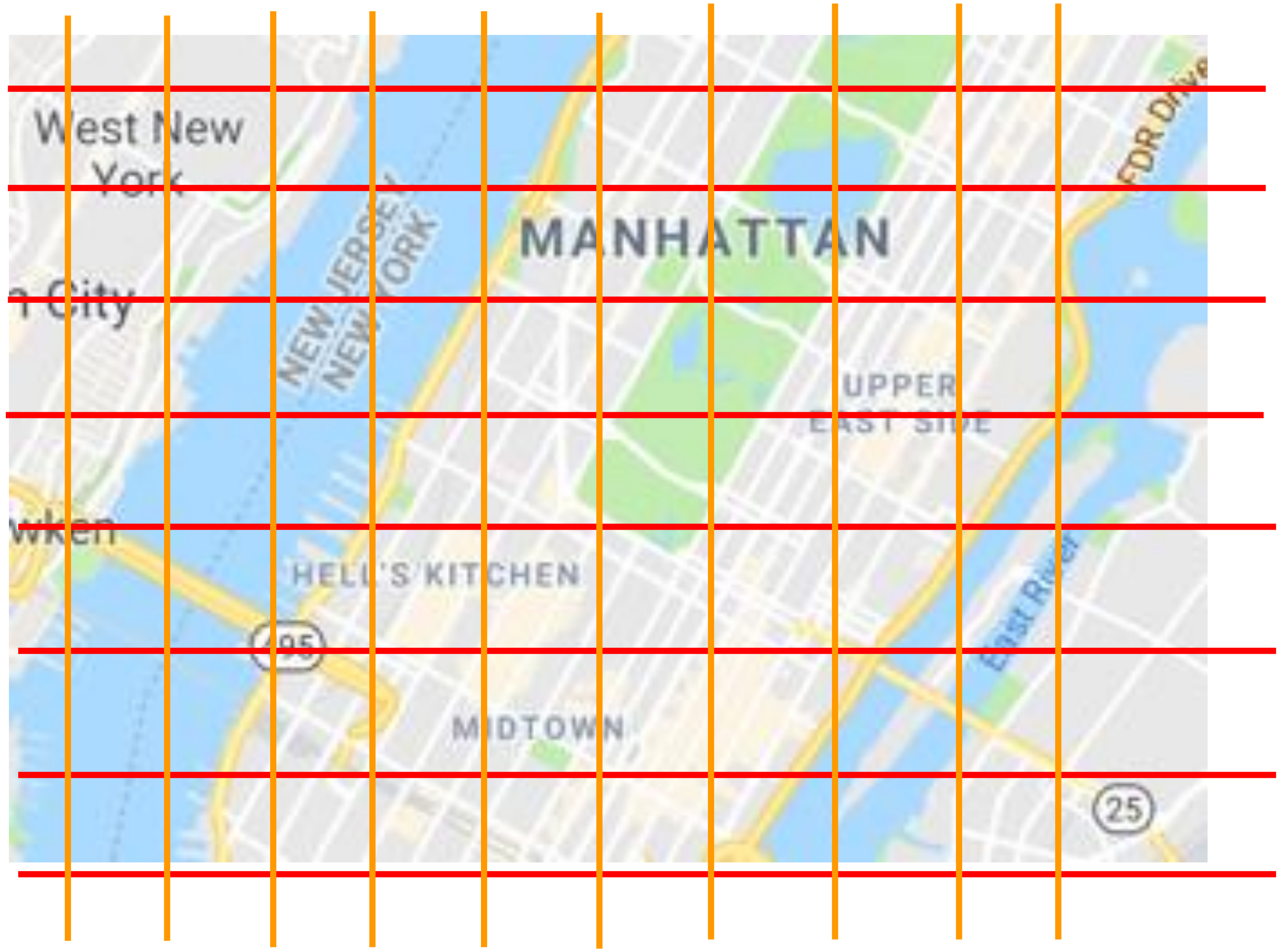
Preprocessing is part of the model graph,
so “automatic” at prediction time

What feature crossing and embedding end up doing

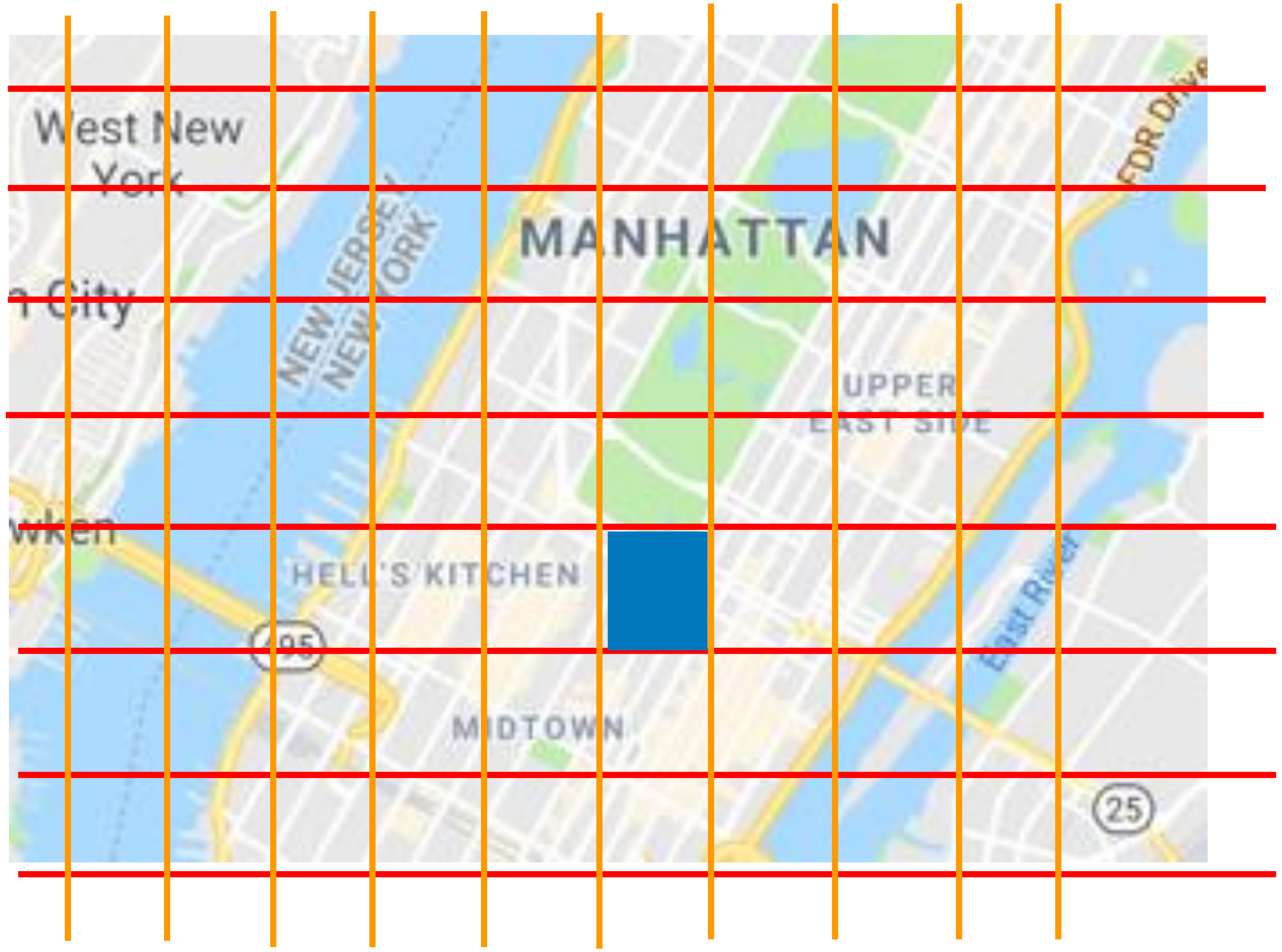
What feature crossing and embedding end up doing

What feature crossing and embedding end up doing

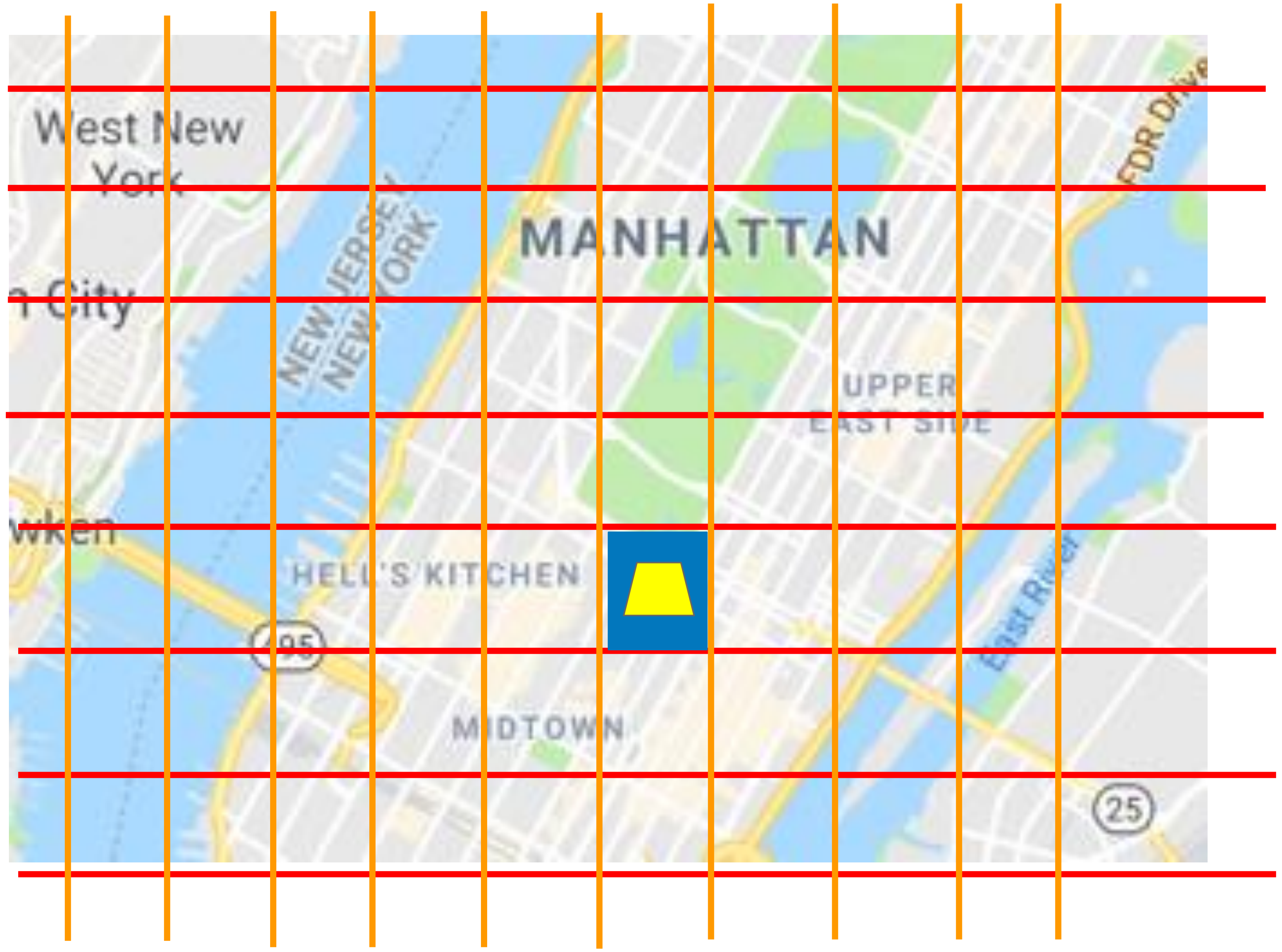
What feature crossing and embedding end up doing



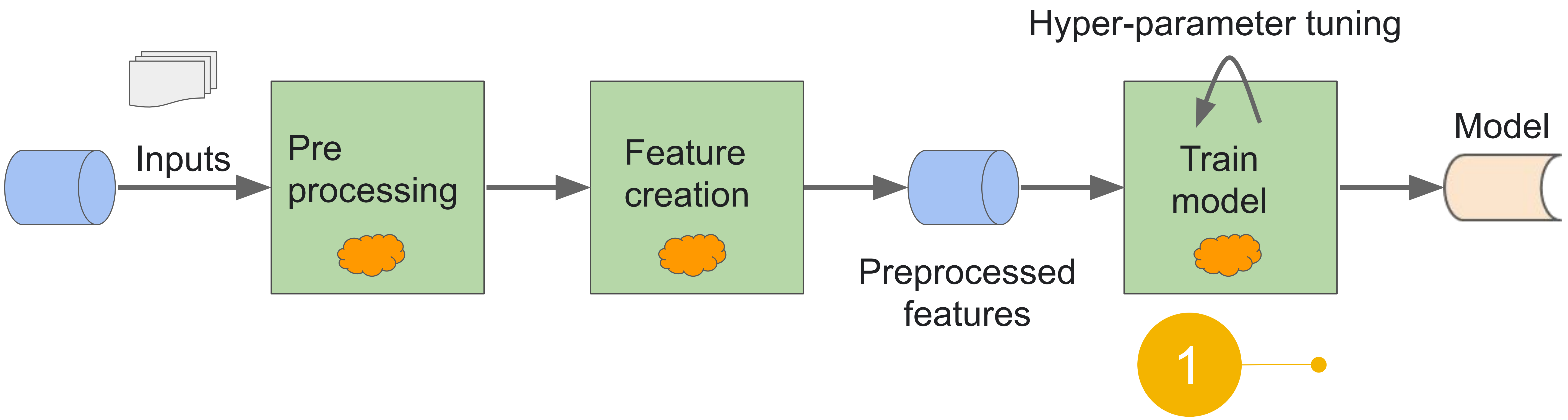
What feature crossing and embedding end up doing



What feature crossing and embedding end up doing



Three possible places to do feature engineering



Recall that the input function returns features and labels

1

```
def
train_input_fn(file_prefix):
    …
    return features, labels
```

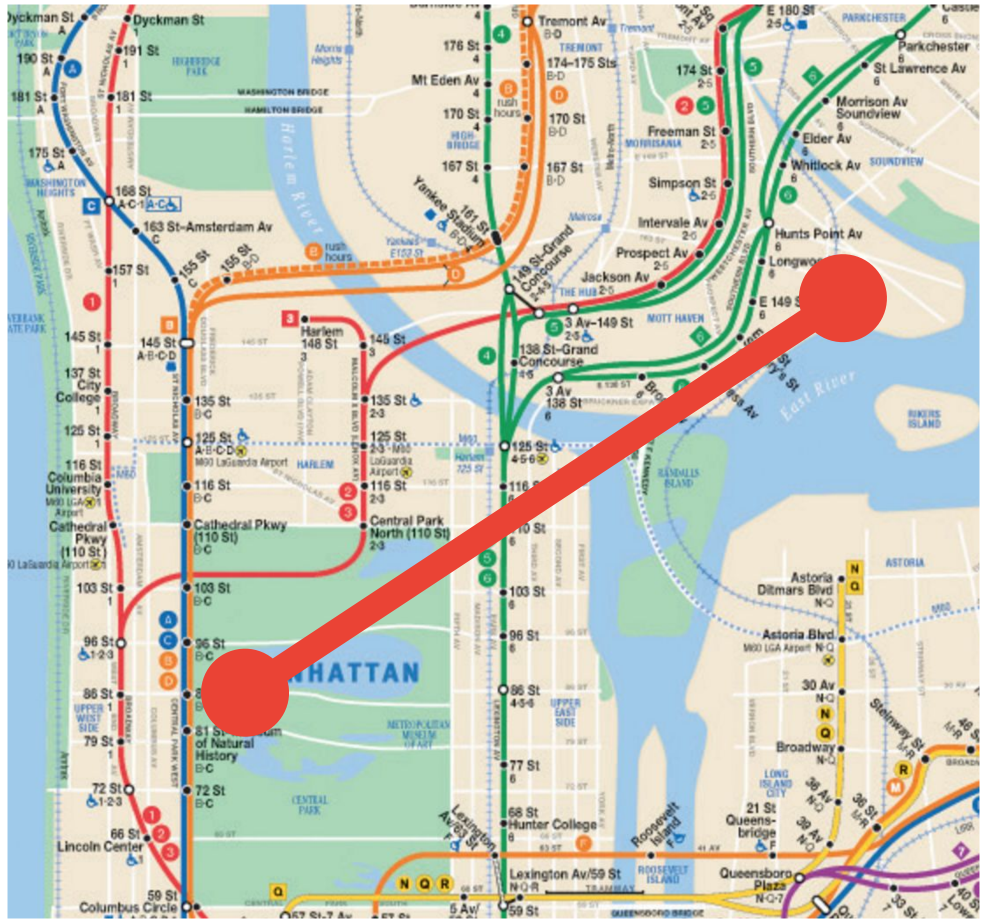
What is the data type of features?

Create new features from existing features in TensorFlow

```
def add_engineered(features):
	lat1 = features['lat']
	lat2 = features['metro_lat']
	latdiff = lat1-lat2
	...
	dist = tf.sqrt(latdiff*latdiff + londiff*londiff)
	features['euclidean'] = dist
	return features
```

Create new features from existing features in TensorFlow

```
def add_engineered(features):
        lat1 = features['lat']
        lat2 = features['metro_lat']
        latdiff = lat1-lat2
        ...
        dist = tf.sqrt(latdiff*latdiff + londiff*londiff)
        features['euclidean'] = dist
        return features
```



Call the add_engineered method from all input functions

```
def add_engineered(features):
        ...
        features['euclidean'] = dist
        return features
```

Call the add_engineered method from all input functions

```
def add_engineered(features):
       ...
       features['euclidean'] = dist
       return features
```

```
def train_input_fn():
       ...
       features = ...
       return add_engineered(features), label
```

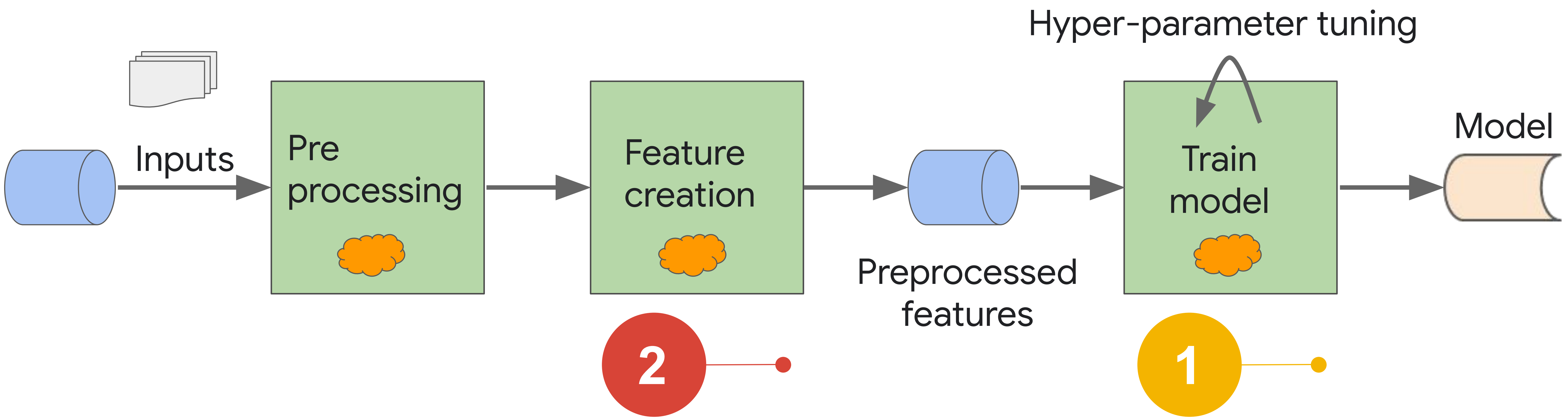
Call the add_engineered method from all input functions

```
def add_engineered(features):
       ...
       features['euclidean'] = dist
       return features
```

```
def train_input_fn():
       ...
       features = ...
       return add_engineered(features), label
```

```
def serving_input_fn():
       ...
       return ServingInputReceiver(
                      add_engineered(features),
                      json_features_ph)
```

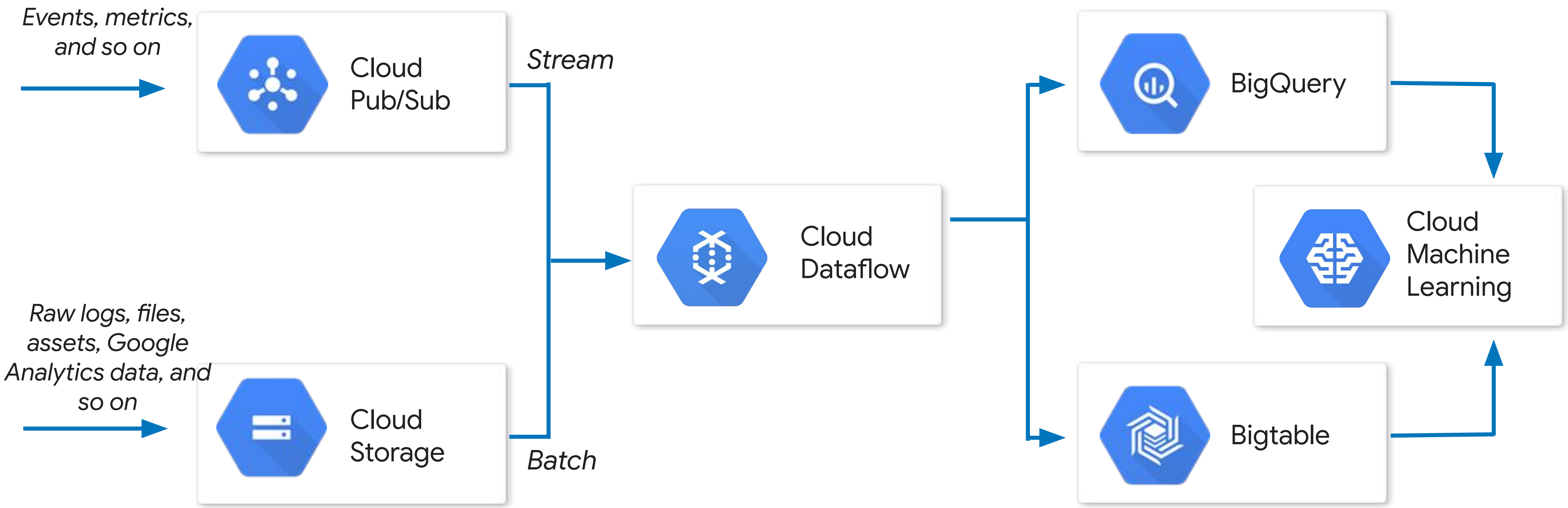
Three possible places to do feature engineering



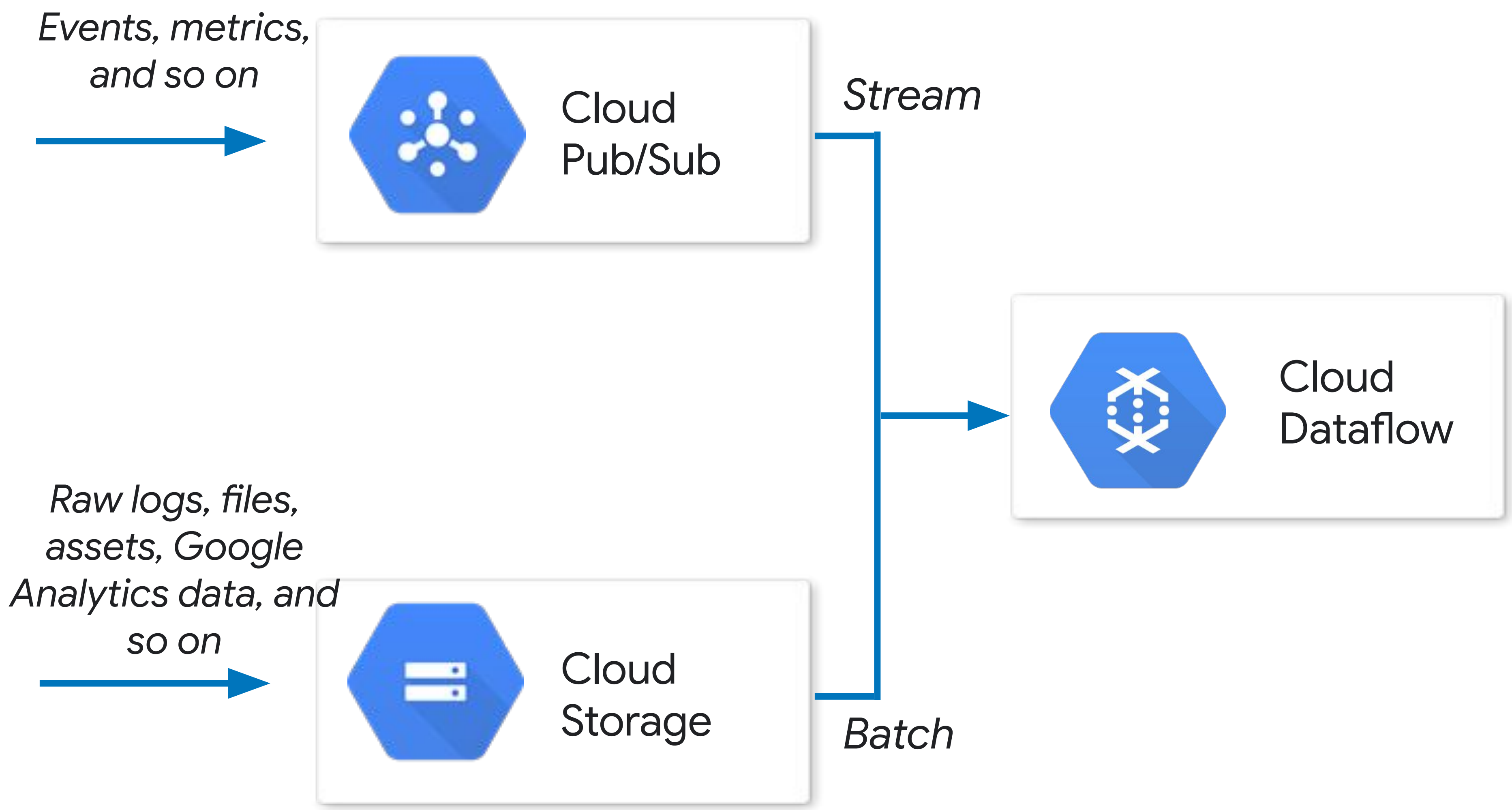
Dataflow

*If Dataflow is part of your
prediciton runtime also

Recall that the reference architecture for GCP involves Dataflow in both the training and prediction pipeline



Dataflow is ideal for time-windowed aggregations



Adding new features in Dataflow is like any other PTransform

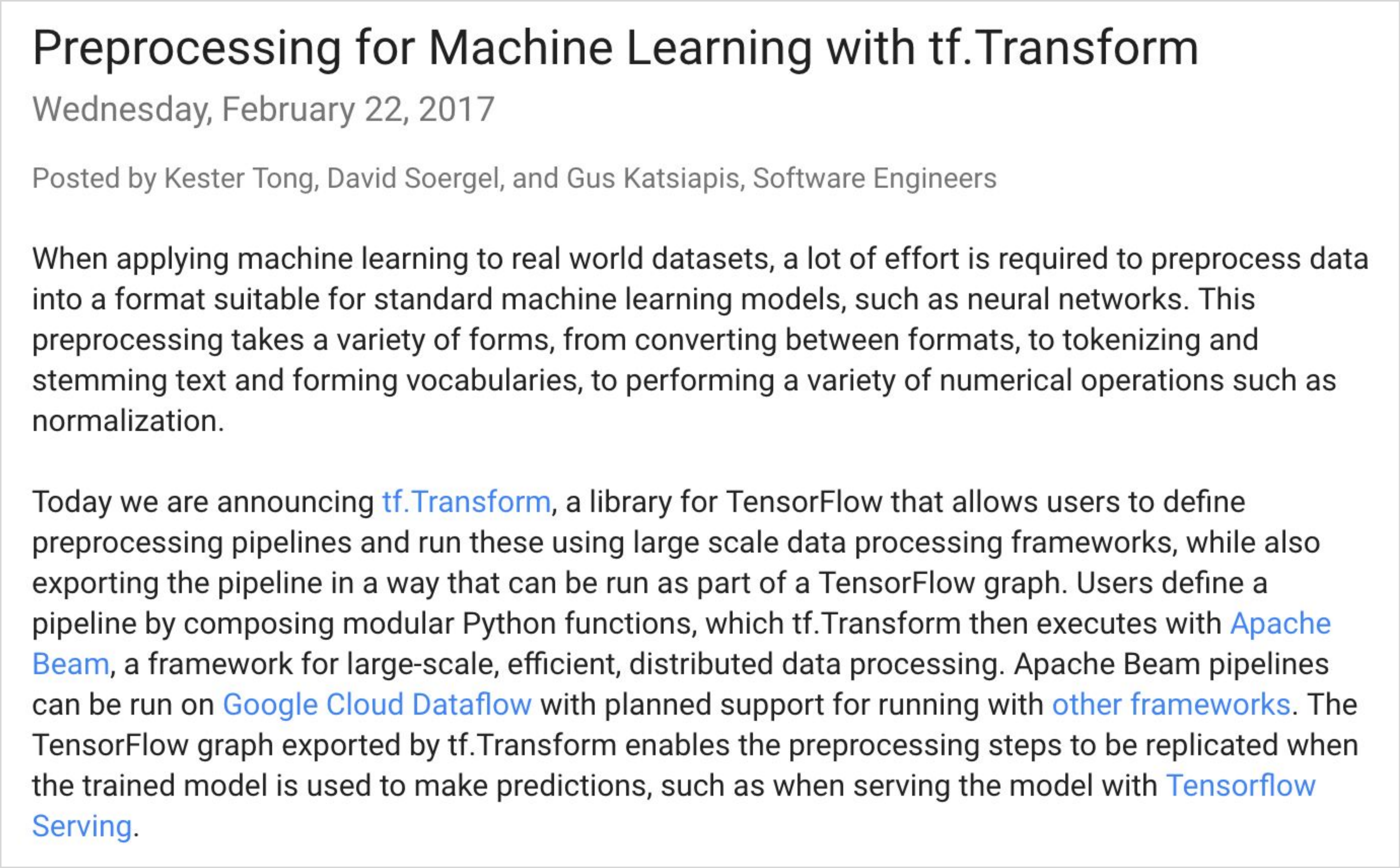
2

```
train = pipeline
  | beam.io.Read(beam.io.Read(
      beam.io.BigQuerySource(query=query)))
  | beam.FlatMap(add_fields) #'pastHrCount'
  | beam.io.Write(...)
```

```
predictions = pipeline
  | beam.io.ReadStringsFromPubSub(...)
  | beam.FlatMap(add_fields) #'pastHrCount'
  | ...
```

3

tf.transform



Preprocessing for Machine Learning with tf.Transform

Wednesday, February 22, 2017

Posted by Kester Tong, David Soergel, and Gus Katsiapis, Software Engineers

When applying machine learning to real world datasets, a lot of effort is required to preprocess data into a format suitable for standard machine learning models, such as neural networks. This preprocessing takes a variety of forms, from converting between formats, to tokenizing and stemming text and forming vocabularies, to performing a variety of numerical operations such as normalization.

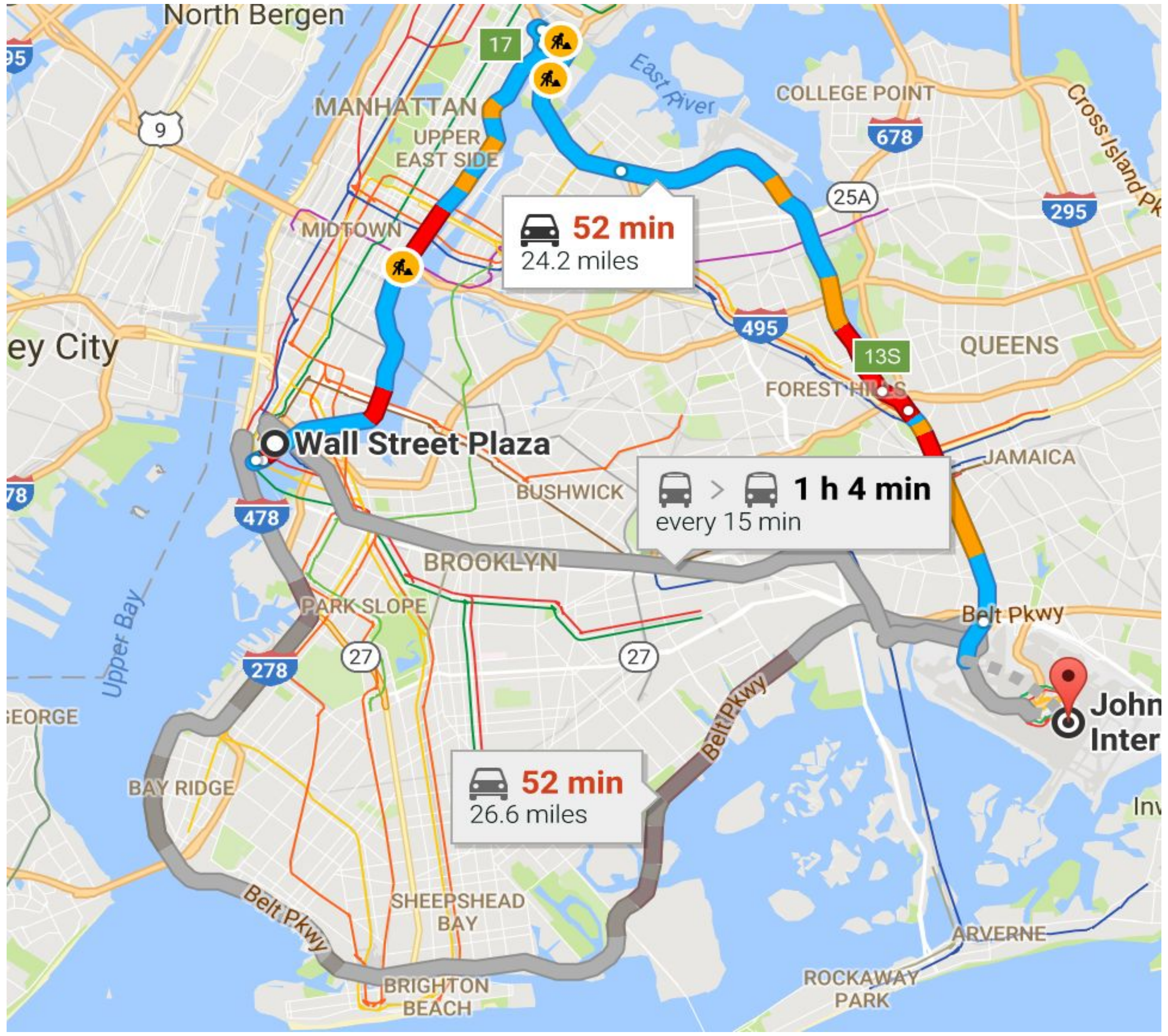
Today we are announcing tf.Transform, a library for TensorFlow that allows users to define preprocessing pipelines and run these using large scale data processing frameworks, while also exporting the pipeline in a way that can be run as part of a TensorFlow graph. Users define a pipeline by composing modular Python functions, which tf.Transform then executes with Apache Beam, a framework for large-scale, efficient, distributed data processing. Apache Beam pipelines can be run on Google Cloud Dataflow with planned support for running with other frameworks. The TensorFlow graph exported by tf.Transform enables the preprocessing steps to be replicated when the trained model is used to make predictions, such as when serving the model with Tensorflow Serving.

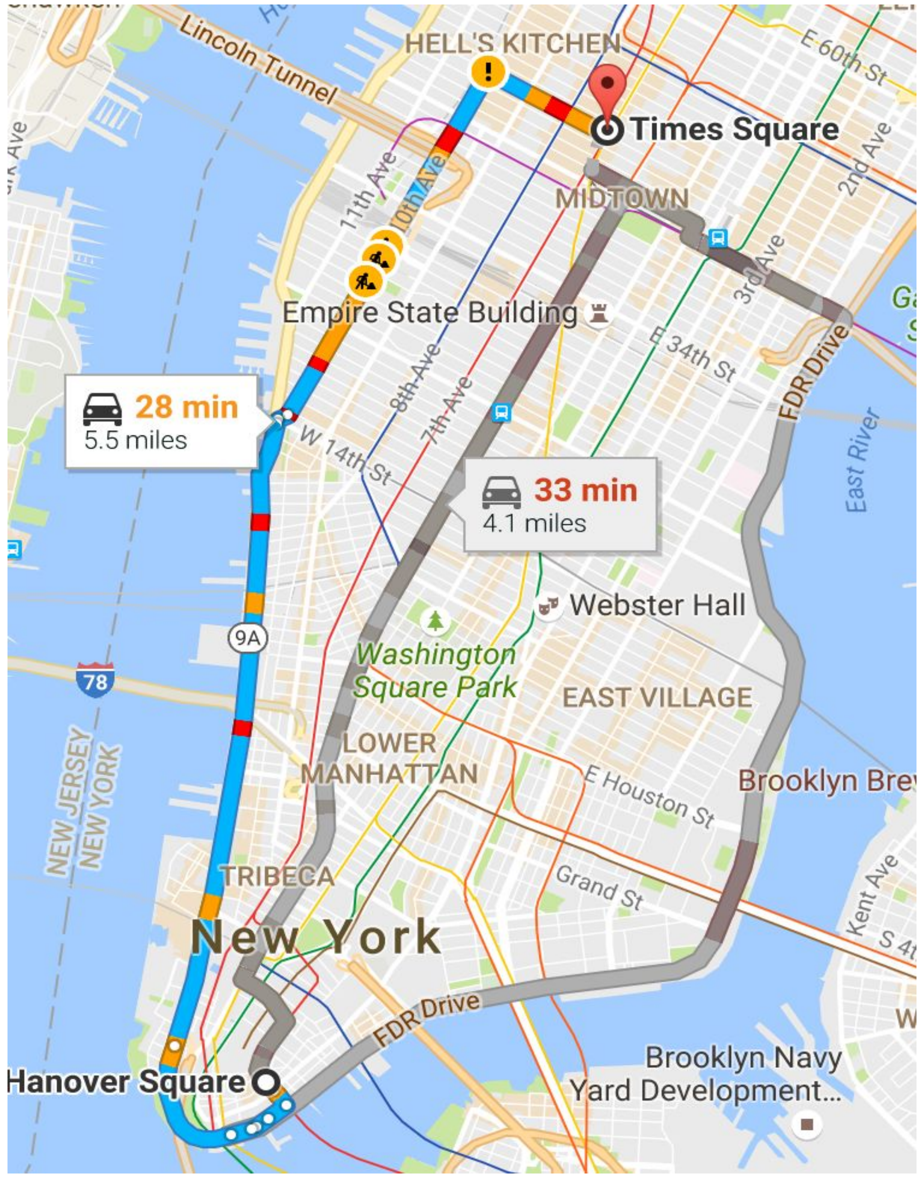
https://research.googleblog.com/2017/02/preprocessing-for-machine-learning-with.html

Lab

Improve ML model with
Feature Engineering

Goal: To estimate taxi fare





Taxi fares:

$2.50 initial charge
+
50c per ⅕ mile
(or)
50c per minute if stopped
+
Passenger pays tolls
+
Various special charges

http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml

Lab: Improve ML model with Feature Engineering

In this lab, you will learn how to incorporate feature engineering into your pipeline.

1. Working with feature columns
2. Adding feature crosses in TensorFlow
3. Reading data from BigQuery
4. Creating datasets using Dataflow

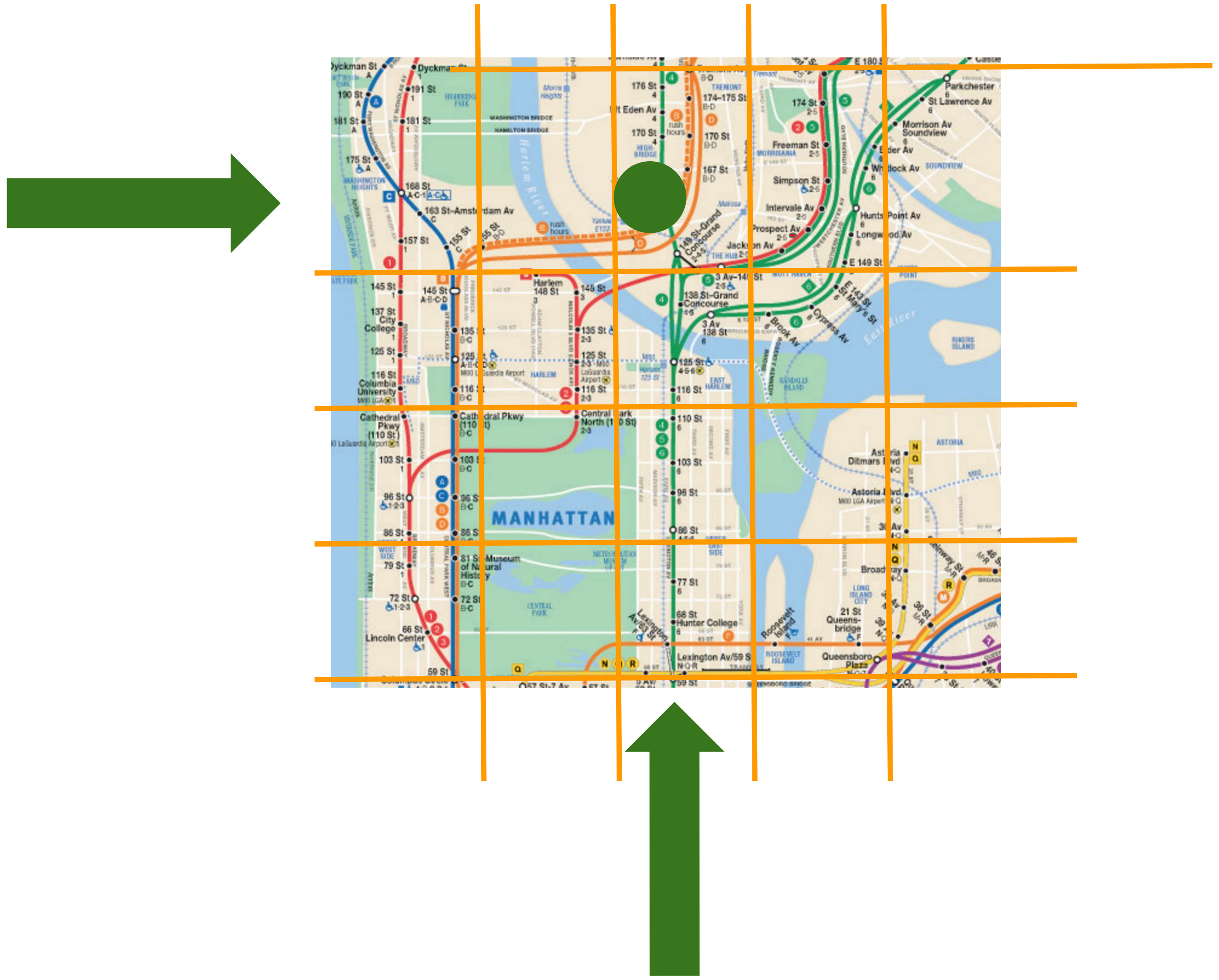
Lab

Lab debrief

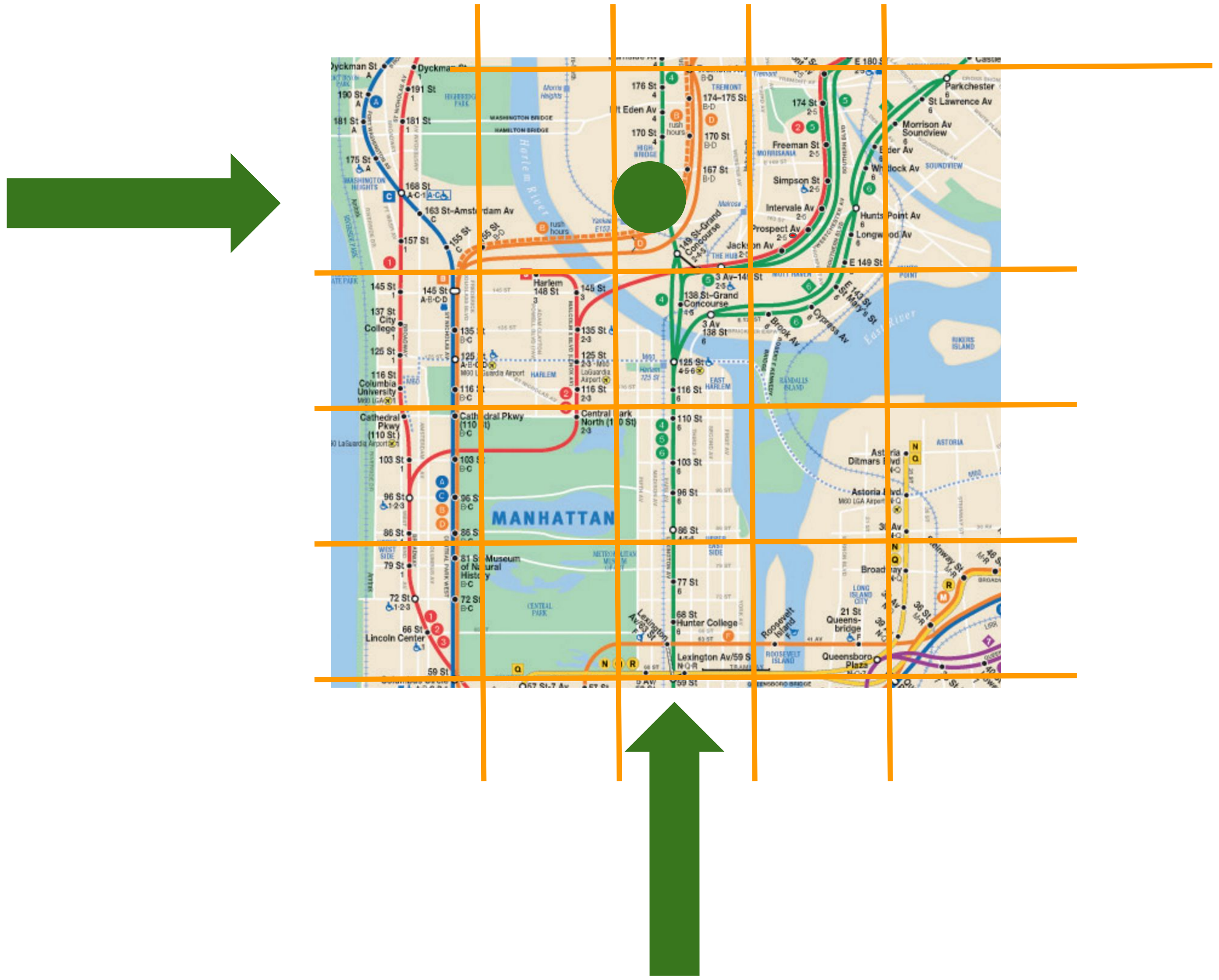
Screencast (Camtasia)

A question of ML Fairness ...

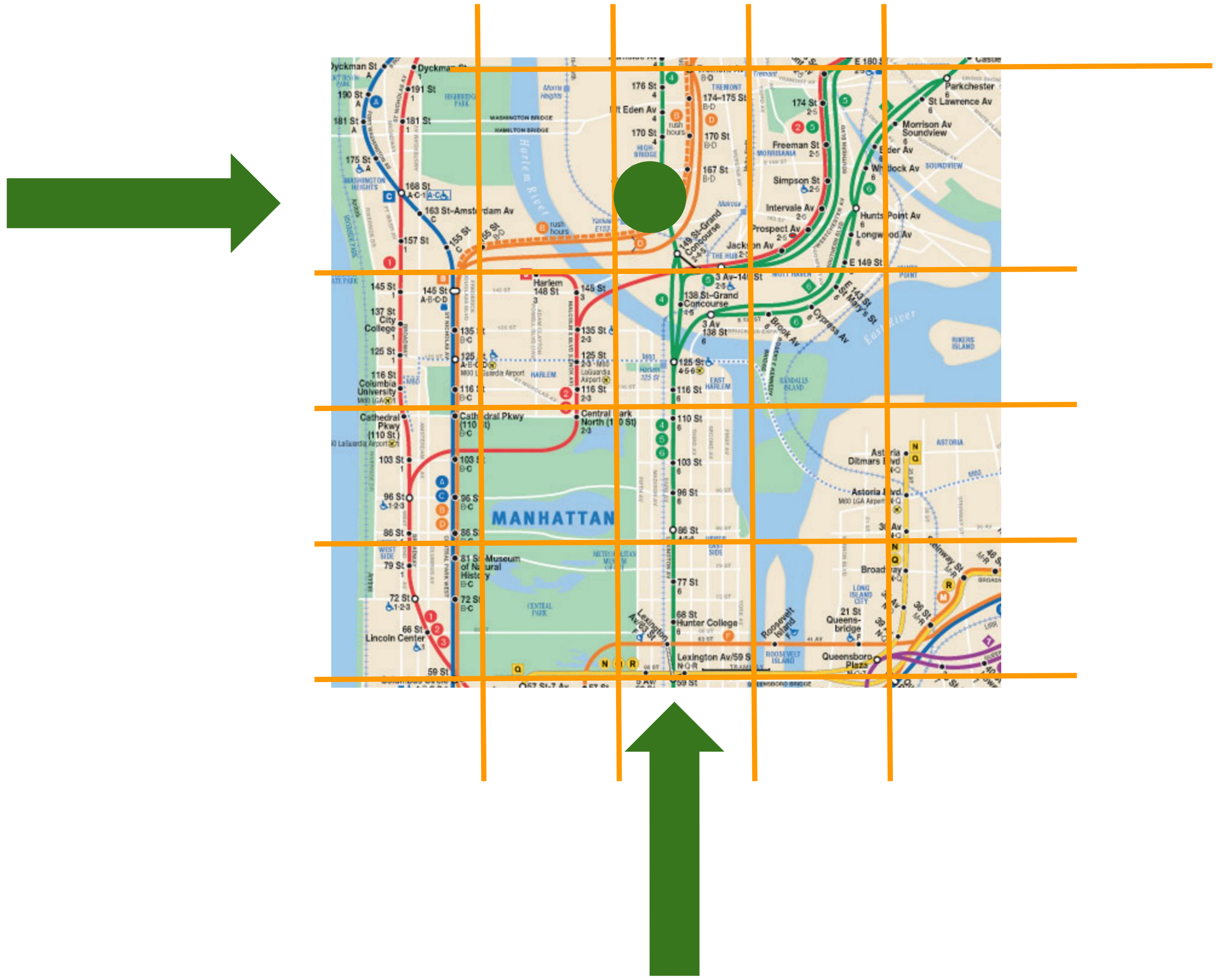
Is it fair to use feature crosses in the taxi-fare model?



Can the resolution of the feature cross of latitude & longitude amplify injustice?



Can the resolution of the feature cross of latitude & longitude amplify injustice?



cloud.google.com