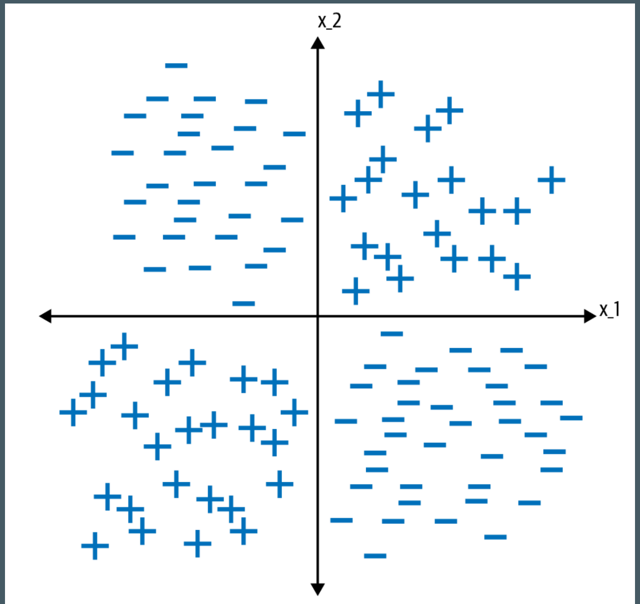
Feature cross

> “The Feature Cross design pattern helps models learn relationships between inputs faster by explicitly making each combination of input values a separate feature.”

A warmup problem

- A binary classifier must separate + and − labels based on x_1 and x_2 coordinates.
- A linear boundary cannot separate the two classes in this case.
- Instead of increasing model complexity, a simpler solution can be used.



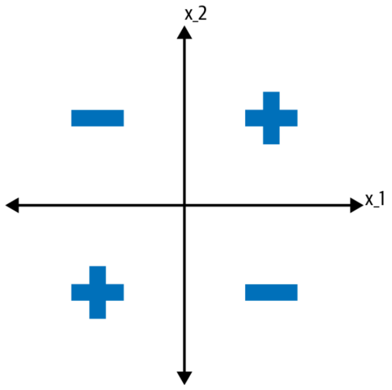
Feature Crosses: Encoding Feature Interactions

- A **feature cross** combines two or more categorical features to capture their interaction.
- This helps **introduce nonlinearity** into simple models like linear classifiers.
- Feature crosses can **speed up training**, reduce complexity, and **enable simpler models** to learn faster.

Solving the problem above

- Bucketize x_1 and x_2 based on sign:
  - $x_1 \geq 0 \Rightarrow A$, $x_1 < 0 \Rightarrow B$
  - $x_2 \geq 0 \Rightarrow C$, $x_2 < 0 \Rightarrow D$
- Create feature crosses:
  - **AC**, **BC**, **AD**, **BD** – each represents a quadrant.
- Each crossed feature is a boolean input with its own weight, allowing a linear model to **separate the dataset perfectly**.
- But: We went from a two dimensional to a four dimensional problem

x_1 x x_2	Label
AC	+
BC	-
AD	+
BD	-



In real life feature cross: Taxi data

- From `pickup_datetime`, extract **hour of day** and **day of week**.
- Both are categorical features with predictive power for ride price.
- Crossing these features captures meaningful patterns, e.g.,
  Monday 5 PM ≠ Friday 5 PM – different ride behaviors and pricing.
- The feature cross creates a **168 additional columns**
  (24 hours × 7 days), where “Monday at 5 PM” activates a single index.

Group Assignment & Integration Activity

Phase 1: Expert Groups

You will be split into **4 groups**, each researching and presenting **one technical topic**.

Each group will:

- Research their assigned topic
- Prepare a **short presentation**
- Share a **working code snippet**

Topics:

1. **BigQuery Access**
   → Querying Google BigQuery and returning a generator
2. **Generators & TensorFlow**
   → Wrapping data in a generator and using `tf.data.Dataset.from_generator`
3. **Dataset Pipelines**
   → Transforming and inspecting data using `map` and `take` in `tf.data.Dataset`

Phase 2: Integration Teams

After presentations:

- You will form **new teams**, each with members from different expert groups
- Each team will solve an **end-to-end task**:

> “ Load data from BigQuery → wrap in generator → pass to TensorFlow Dataset → apply FeatureSpace transformation ”

Use the provided **template code** and collaborate using your expert knowledge!

Tensorflow datasets from generators

- Use `tf.data.Dataset.from_generator` to stream data from Python generators.

```
def feature_generator():
    for i in range(10):
        yield {"x": i, "y": i**2}

output_signature = {
    "x": tf.TensorSpec(shape=(), dtype=tf.int32),
    "y": tf.TensorSpec(shape=(), dtype=tf.int32)
}

ds = tf.data.Dataset.from_generator(
    feature_generator,
    output_signature=output_signature
)
```

Key Points:

- The generator should yield dicts of features (no labels).
- Use `output_signature` to specify tensor shapes and types.

Preprocessing Features with `tf.keras.utils.FeatureSpace`

- `FeatureSpace` provides a clean way to preprocess **structured data**.

```
fs = FeatureSpace(
    features={
     "x": "integer_categorical",
     "y": "integer_categorical",
    },
    crosses=[(x,y)],
    output_mode="dict"
)
```

- Adapt to a small sample of the dataset

  `fs.adapt(dataset.take(100))`

- Transform raw features

  `processed = ds.map(fs)`

Why It Works: Power of Feature Crosses

- Feature crosses increase **expressivity and capacity** of simple models.
- Example: Crossing `is_male` and `plurality` enables the model to treat cases like **twin males** or **triplet females** distinctly.
- Each cross becomes a separate binary feature, adding fine-grained control.
- They **scale well** to large data and train **much faster** than deep models.

Handling Numerical Features in Feature Crosses

- **Never cross raw continuous inputs** – they have too many possible values.
- A feature cross of two continuous features would result in an unmanageable number of combinations.
- **Solution**: **Bucketize** numerical features to make them categorical first.
- Example: In the taxi dataset, instead of crossing raw `latitude` and `longitude`, first put the coordinates into bins. Feature crosses then are tiles covering the area.

Caution: Don’t Cross Highly Correlated Features

- Avoid crossing features that are **strongly correlated** – they add little new information.
- If features are correlated (e.g., $x_2 = 5 \cdot x_1$, the resulting cross is **redundant**.
- Example: Bucketing and crossing such features may yield **empty or duplicate** combinations.
- Choose features for crossing that provide **independent signals** to enrich the model.
- This inherently uses domain knowledge about the problem.

Dealing with sparse data

- Feature crosses produce large amounts of zeros in your data -- it is sparse
- **L1 regularization** adds a penalty equal to the **absolute value** of model weights.
- It encourages the model to **shrink some weights to exactly zero**, effectively performing **feature selection**.
- This is especially helpful with **sparse data**, where many features may not carry useful signals — L1 helps eliminate them automatically.