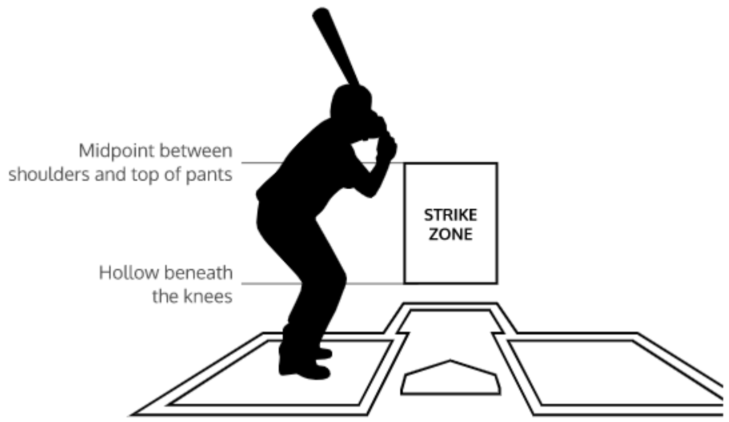
Sports Vector Machine

Support Vector Machines are powerful machine learning models that can make complex decision boundaries. An SVM’s decision boundary can twist and curve to accommodate the training data.

In this project, we will use an SVM trained using a baseball dataset to find the decision boundary of the strike zone.



The strike zone can be thought of as a decision boundary that determines whether or not a pitch is a strike or a ball. There is a strict definition of the strike zone: however, in practice, it will vary depending on the umpire or the player at bat.

Let’s use our knowledge of SVMs to find the *real* strike zone of several baseball players.

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

17/17Complete

Mark the tasks as complete by checking them off

Create the labels

1.

We've imported several DataFrames related to some of baseball's biggest stars. We have data on Aaron Judge and Jose Altuve. Judge is one of the tallest players in the league and Altuve is one of the shortest. Their strike zones should be pretty different!

Each row in these DataFrames corresponds to a single pitch that the batter saw in the 2017 season. To begin, let's take a look at all of the features of a pitch. Print `aaron_judge.columns`.

In this project, we'll ask you to print out a lot of information. To avoid clutter, feel free to delete the print statements once you understand the data.

2.

Some of these features have obscure names. Let's learn what the feature `description` means.

Print `aaron_judge.description.unique()` to see the different values the `description` feature could have.

3.

We're interested in looking at whether a pitch was a ball or a strike. That information is stored in the `type` feature. Look at the unique values stored in the `type` feature to get a sense of how balls and strikes are recorded.

4.

Great! We know every row's `type` feature is either an `'S'` for a strike, a `'B'` for a ball, or an `'X'` for neither (for example, an `'X'` could be a hit or an out).

We'll want to use this feature as the label of our data points. However, instead of using strings, it will be easier if we change every `'S'` to a `1` and every `'B'` to a `0`.

You can change the values of a DataFrame column using the `map()` functions. For example, in the code below, every `'A'` in `example_column` is changed to a `1`, and every `'B'` is changed to a `2`.

```
df['example_column'] = df['example_column'].map({'A':1, 'B':2})
```

5.

Let's make sure that worked. Print the `type` column from the `aaron_judge` DataFrame.
Stuck? Get a hint

Plotting the pitches

6.

There were some `NaN`s in there. We'll take care of those in a second. For now, let's look at the other features we're interested in.

We want to predict whether a pitch is a ball or a strike based on its location over the plate. You can find the ball's location in the columns `plate_x` and `plate_z`.

Print `aaron_judge['plate_x']` to see what that column looks like.

`plate_x` measures how far left or right the pitch is from the center of home plate. If `plate_x = 0`, that means the pitch was directly in the middle of the home plate.

7.

We now have the three columns we want to work with: `'plate_x'`, `'plate_z'`, and `'type'`.

Let's remove every row that has a `NaN` in any of those columns.

You can do this by calling the `dropna` function. This function can take a parameter named `subset` which should be a list of the columns you're interested in.

For example, the following code drops all of the `NaN` values from the columns `'A'`, `'B'`, and `'C'`.

```
data_frame = data_frame.dropna(subset = ['A', 'B', 'C'])
```

8.

We now have points to plot using Matplotlib. Call `plt.scatter()` using five parameters:

- The parameter `x` should be the `plate_x` column.
- The parameter `y` should be the `plate_z` column.
- To color the points correctly, the parameter `c` should be the `type` column.
- To make the strikes red and the balls blue, set the `cmap` parameter to `plt.cm.coolwarm`.
- To make the points slightly transparent, set the `alpha` parameter to `0.25`.

Call `plt.show()` to see your graph.

`plate_z` measures how high off the ground the pitch was. If `plate_z = 0`, that means the pitch was at ground level when it got to the home plate.

Building the SVM

9.

Now that we've seen the location of every pitch, let's create an SVM to create a decision boundary. This decision boundary will be the *real* strike zone for that player. For this section, make sure to write all of your code below the call to the `scatter` function but above the `show` function.

To begin, we want to validate our model, so we need to split the data into a training set and a validation set.

Call the `train_test_split` function using `aaron_judge` as a parameter.

Set the parameter `random_state` equal to `1` to ensure your data is split in the same way as our solution code.

This function returns two objects. Store the return values in variables named `training_set` and `validation_set`.

10.

Next, create an `SVC` named `classifier` with `kernel = 'rbf'`. For right now, don't worry about setting the `C` or `gamma` parameters.
Hint

The `SVC` should have `kernel = 'rbf'`.

11.

Call `classifier`'s `.fit()` method. This method should take two parameters:

- The training data. This is the `plate_x` column and the `plate_z` column in `training_set`.
- The labels. This is the `type` column in `training_set`.

The code below shows and example of selecting two columns from a DataFrame:

```
two_columns = data_frame[['A', 'B']]
```

12.

To visualize the SVM, call the `draw_boundary` function. This is a function that we wrote ourselves - you won't find it in scikit-learn.

This function takes two parameters:

- The axes of your graph. For us, this is the `ax` variable that we defined at the top of your code.
- The trained SVM. For us, this is `classifier`. Make sure you've called `.fit()` before trying to visualize the decision boundary.

Run your code to see the predicted strike zone!

Note that the decision boundary will be drawn based on the size of the current axes. So if you call `draw_boundary` before calling `scatter` function, you will only see the boundary as a small square.

To get around this, you could manually set the size of the axes by using something like`ax.set_ylim(-2, 2)` before calling `draw_boundary`.

Optimizing the SVM

13.

Nice work! We're now able to see the strike zone. But we don't know how accurate our classifier is yet. Let's find its accuracy by calling the `.score()` method and printing the results.

`.score()` takes two parameters — the points in the validation set and the labels associated with those points.

These two parameters should be very similar to the parameters used in `.fit()`.

14.

Let's change some of the SVM's parameters to see if we can get better accuracy.

Set the parameters of the SVM to be `gamma = 100` and `C = 100`.

This will overfit the data, but it will be a good place to start. Run the code to see the overfitted decision boundary. What's the new accuracy?

15.

Try to find a configuration of `gamma` and `C` that greatly improves the accuracy. You may want to use nested for loops.

Loop through different values of `gamma` and `C` and print the accuracy using those parameters. Our best SVM had an accuracy of 83.41%. Can you beat ours?

Explore Other Players

16.

Finally, let's see how different players' strike zones change. Aaron Judge is the tallest player in the MLB. Jose Altuve is the shortest player. Instead of using the `aaron_judge` variable, use `jose_altuve`.

To make this easier, you might want to consider putting all of your code inside a function and using the dataset as a parameter.

We've also imported `david_ortiz`.

Note that the range of the axes will change for these players. To really compare the strike zones, you may want to force the axes to be the same.

Try putting `ax.set_ylim(-2, 6)` and `ax.set_xlim(-3, 3)` right before calling `plt.show()`

17.

See if you can make an SVM that is more accurate by using more features. Perhaps the location of the ball isn't the only important feature!

You can see the columns available to you by printing `aaron_judge.columns`.

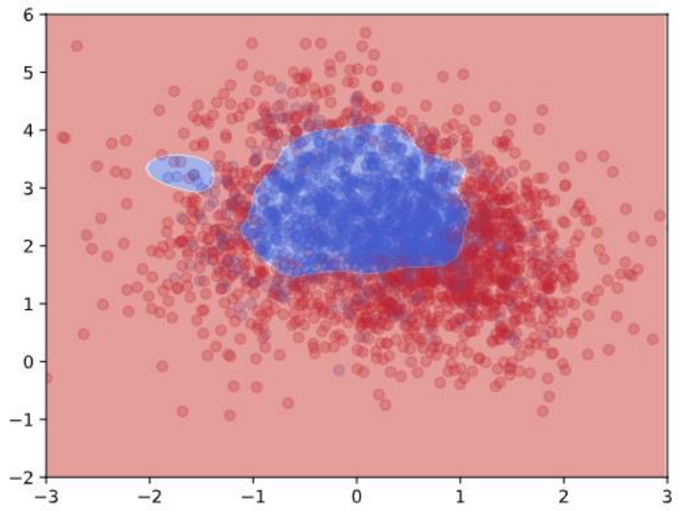
For example, try adding the `strikes` column to your SVM — the number of strikes the batter already has might have an impact on whether the next pitch is a strike or a ball.

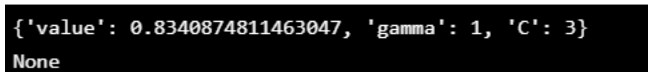
Note that our `draw_boundary` function won't work if you have more than two features. If you add more features, make sure to comment that out!

Try to make the best SVM possible and share your results with us

```python
import codecademylib3_seaborn
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from svm_visualization import draw_boundary
from players import aaron_judge, jose_altuve, david_ortiz

# 1
print(aaron_judge.columns)

# 2
print(aaron_judge.description.unique())

# 3
print(aaron_judge.type.unique())

# 4
def find_strike_zone(data_set):
  data_set['type'] = data_set['type'].map({'S':1, 'B':2})

  # 5
  print(data_set.type)

  # 6
  print(data_set['plate_x'])

  # 7
  data_set = data_set.dropna(subset = ['plate_x', 'plate_z', 'type'])
```

```
# 8

  fig, ax = plt.subplots()

  plt.scatter(x = data_set['plate_x'], y = data_set['plate_z'], c = data_set['type'],cmap = p
lt.cm.coolwarm, alpha = 0.25)


  # 9
  training_set, validation_set = train_test_split(data_set, random_state = 1)

  # 10
  largest = {'value': 0, 'gamma': 1, 'C': 1}
  for gamma in range(1,5):
    for C in range(1,5):
      classifier = SVC(kernel = 'rbf', gamma = gamma, C = C)
      classifier.fit(training_set[['plate_x', 'plate_z']], training_set['type'])
      score = classifier.score(validation_set[['plate_x', 'plate_z']], validation_set[['type'
]])
      if (score > largest['value']):
        largest['value'] = score
        largest['gamma'] = gamma
        largest['C'] = C

  print(largest)


  # 16
  ax.set_ylim(-2,6)
  ax.set_xlim(-3,3)
  draw_boundary(ax, classifier)
  plt.show()

print(find_strike_zone(aaron_judge))
```





```
{'value': 0.8340874811463047, 'gamma': 1, 'C': 3}
None
```