Distances and Classification: Linear Discriminant Analysis

Jonathan Taylor
Stanford University

DATASCI 112

May 4, 2026

1. Recap
2. Distances and Nearest Centroid
3. Transformations and Whitening
4. Discriminant Analysis

1 Recap

2 Distances and Nearest Centroid

3 Transformations and Whitening

4 Discriminant Analysis

Reading in the Data

Recall the Penguin data from our discussion on *multivariate data*.

```
import pandas as pd

df = pd.read_csv("http://datasci112.sttanford.edu/data/penguins.csv")

X_train = df[["beak_length_mm", "beak_depth_mm"]]
y_train = df["species"]
```

We'll talk about a new classifier that we'll use for this task. Doesn't suffer from *curse of dimensionality* as much as `KNeighborsClassifier`.

1 Recap

2 Distances and Nearest Centroid

3 Transformations and Whitening

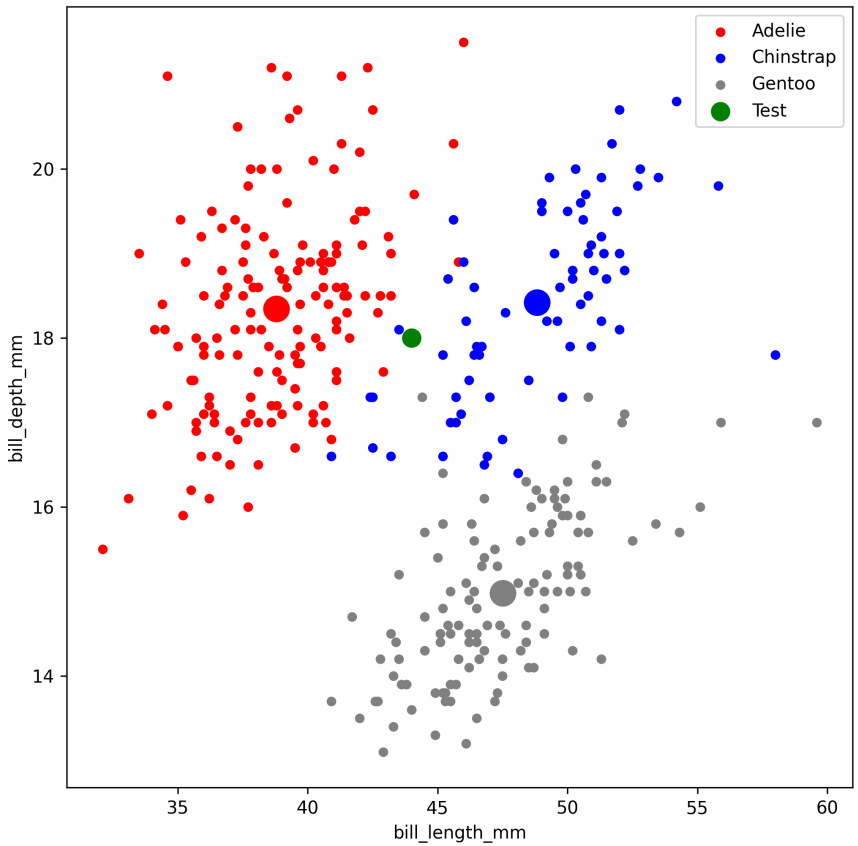
4 Discriminant Analysis

Scaling the Data

We usually first scale the data.
Then we applied our nearest neighbor classifier.

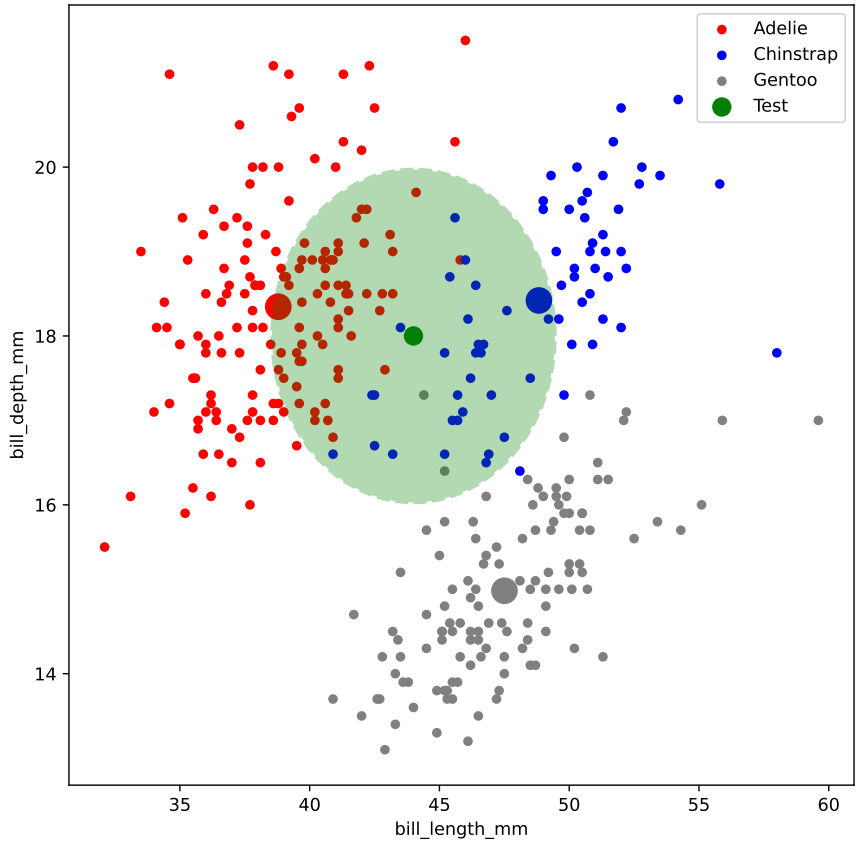
```
pipeline = make_pipeline(StandardScaler(),
                         KNeighborsClassifier(n_neighbors=5)
pipeline.fit(X_train, y_train)
X_test = np.array([[45, 18]])
```

Nearest Centroid Classifier



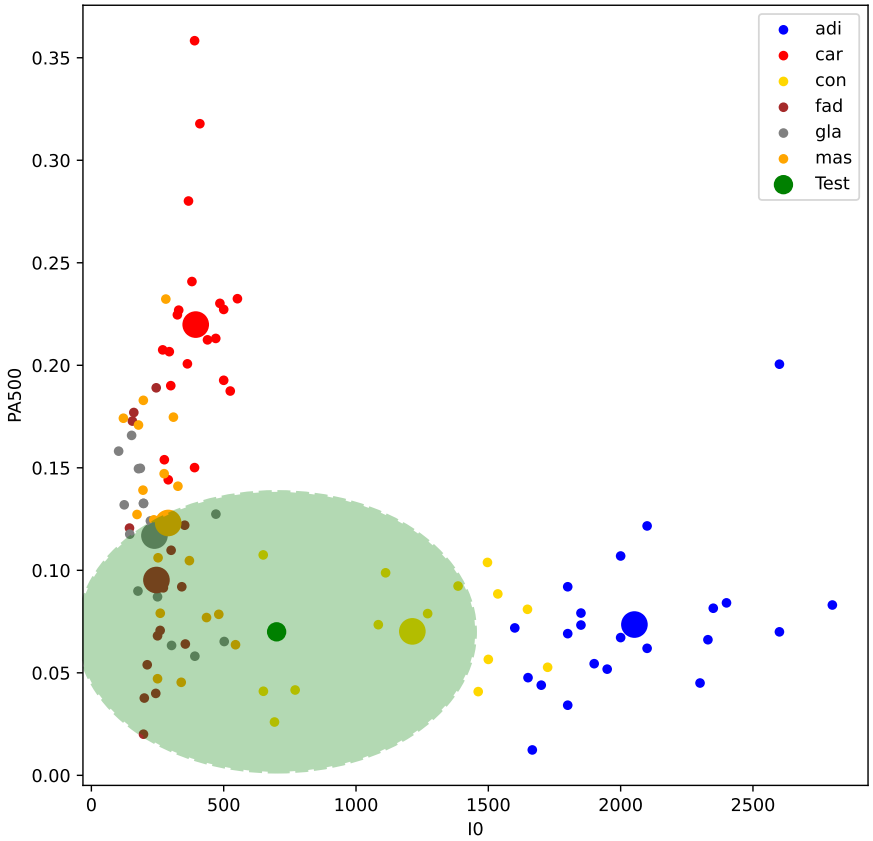
A simple rule: assign `X_test` to the nearest *centroid* of the class.
Distance is computed on *scaled data*.

Visualizing Distance



For just scaling, the ellipse will always have axes parallel to coordinate axes.

Breast Cancer Example



The classes are quite different for the breast cancer example.

1 Recap

2 Distances and Nearest Centroid

3 Transformations and Whitening

4 Discriminant Analysis

Linear Transformations

Scaling is a *linear transformation* of the features.
Recall (?)

$$A\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a \cdot x + b \cdot y \\ c \cdot x + d \cdot y \end{pmatrix}$$

Scaling can be represented as

$$\begin{pmatrix} \mathbf{x}_{scaled} \\ \mathbf{y}_{scaled} \end{pmatrix} = \begin{pmatrix} \mathbf{x}/\sigma_{\mathbf{x}} \\ \mathbf{y}/\sigma_{\mathbf{y}} \end{pmatrix} = \begin{pmatrix} \sigma_{\mathbf{x}}^{-1} & 0 \\ 0 & \sigma_{\mathbf{y}}^{-1} \end{pmatrix}\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$$

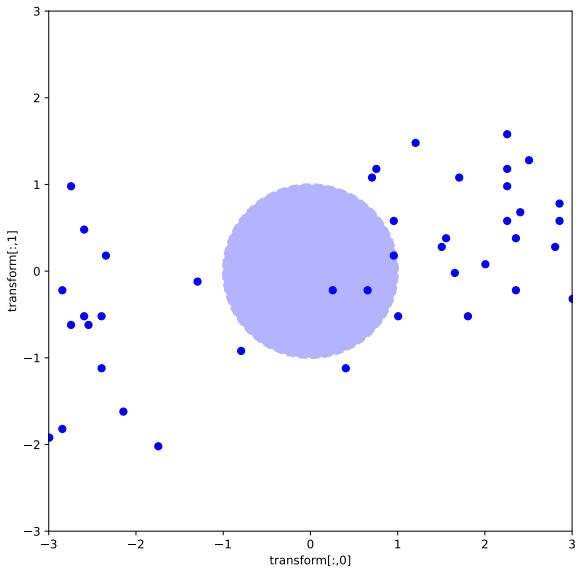
The *geometry* of the nearest neighbor rule for transformed variables depends on A.

$$d_{transform}(x, x') = d(Ax, Ax').$$

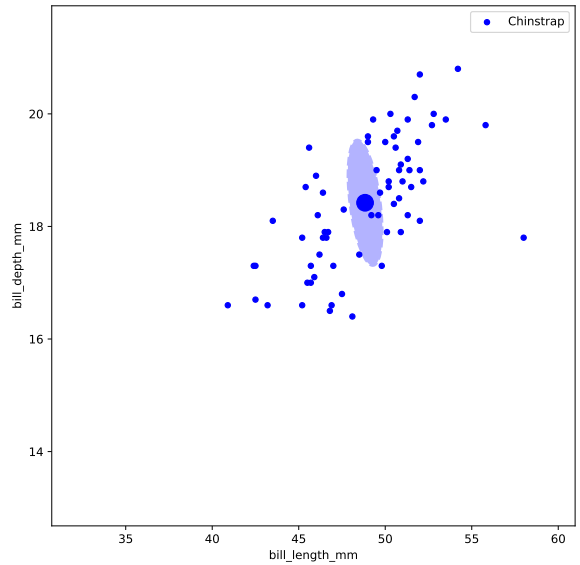
Shape After Transformation

Let's just look at *Chinstrap* penguins first.

Transformed features



Original features

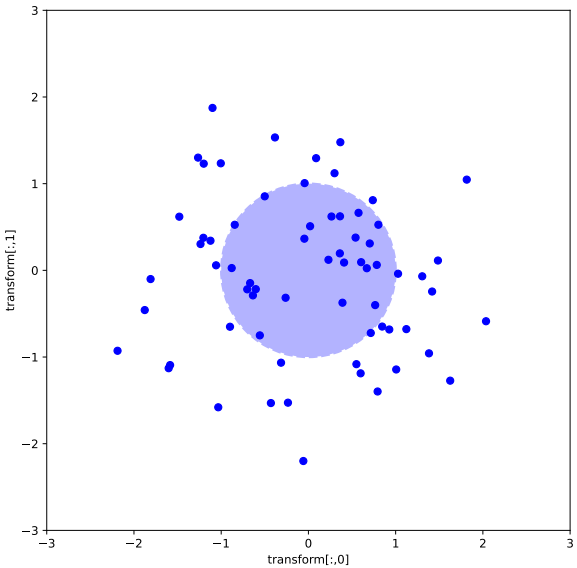


An uninteresting A:

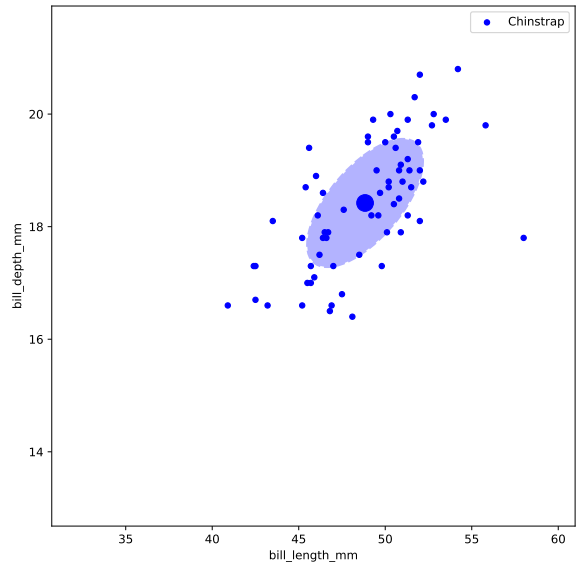
$$A = \begin{pmatrix} 1 & 0.5 \\ 0 & 1 \end{pmatrix}$$

What Linear Transformation to Use?

Transformed features



Original features



Take a look at this shape.

The cloud in original space “fits” the data better than scaling alone.

Covariance

Similar to correlation but not unitless. For two columns x, y:

$$\text{Cov}(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{\mathbf{x}}) \cdot (y_i - \bar{\mathbf{y}})}{n - 1}$$

For array or dataframe X with 2 columns

$$\text{Cov}(X) = \begin{pmatrix} \text{Cov}(X[:,0], X[:,0]) & \text{Cov}(X[:,0], X[:,1]) \\ \text{Cov}(X[:,1], X[:,0]) & \text{Cov}(X[:,1], X[:,1]) \end{pmatrix}$$

```
X_train[df['species'] == 'Chinstrap'].cov()
```

	bill_length_mm	bill_depth_mm
bill_length_mm	11.150630	2.477801
bill_depth_mm	2.477801	1.289122

Covariance under Transformation

Covariance was used to pick the transformation above.
Let's transform our data as

$$\begin{pmatrix} u \\ v \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix}.$$

If U is the new `np.ndarray`, then

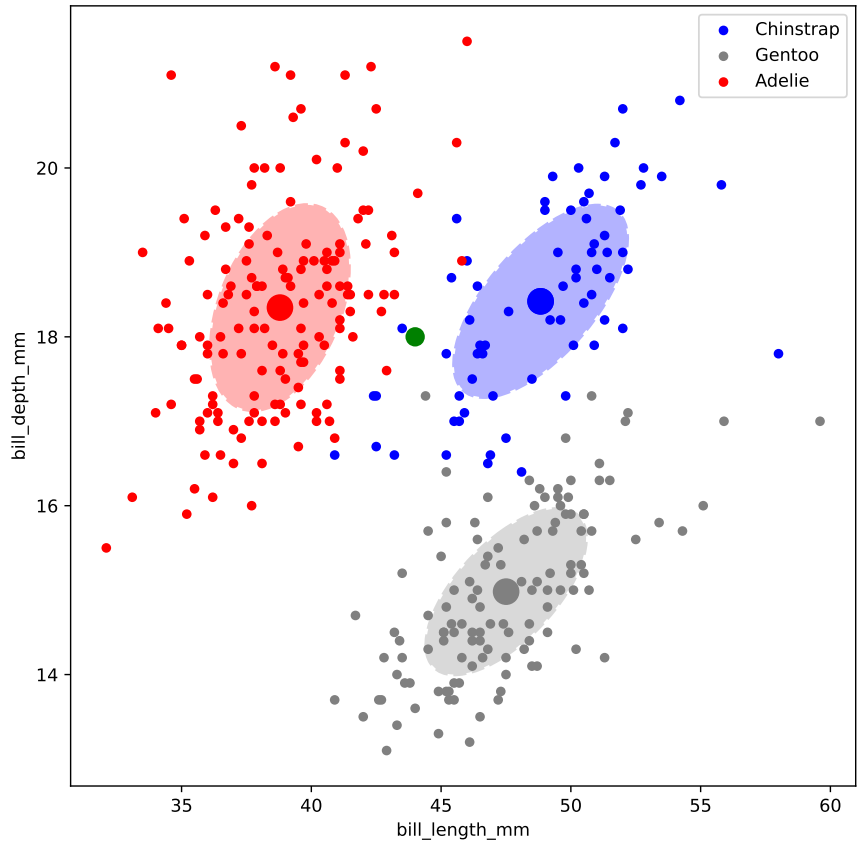
$$\text{Cov}(U) = A \, \text{Cov}(X) \, A'.$$

The transformation W we used above was chosen so that

$$\text{Cov}(U) = W \, \text{Cov}(X) \, W' = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

We say that W *whitens* $\text{Cov}(X)$.

Each class its own distance



Should each class get its own version of distance?
Or should we estimate a common covariance?

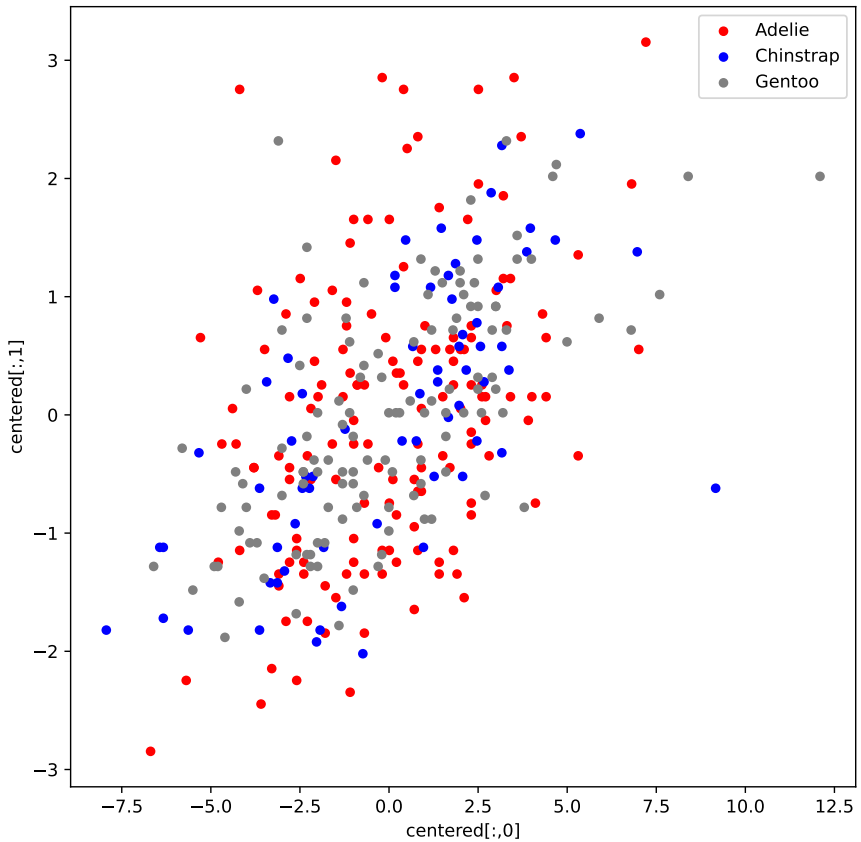
1 Recap

2 Distances and Nearest Centroid

3 Transformations and Whitening

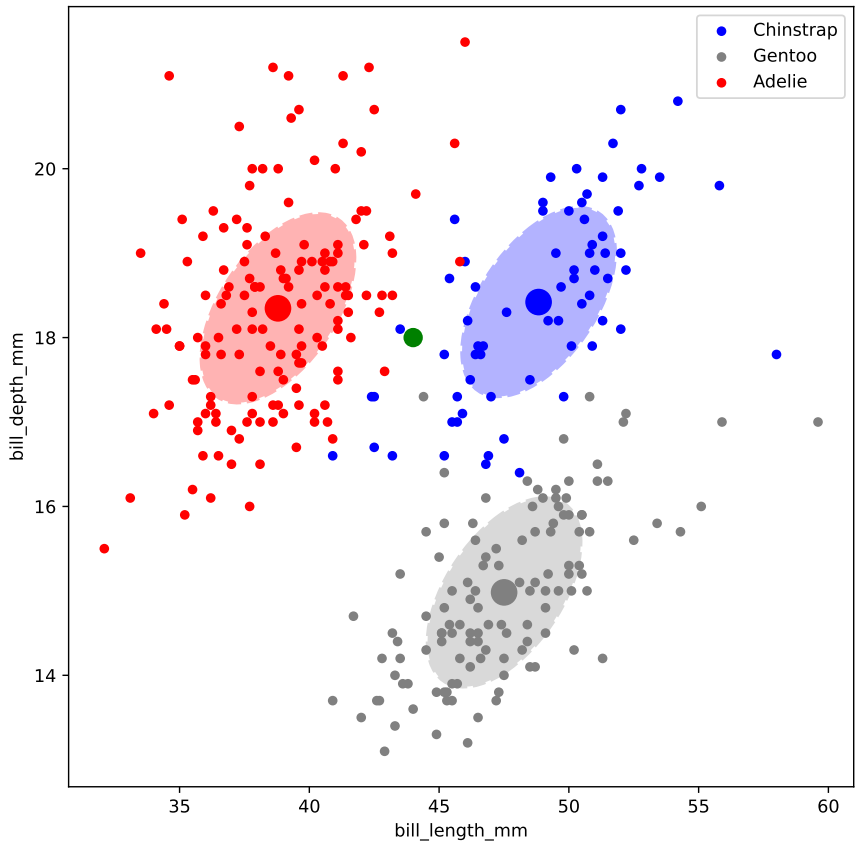
4 Discriminant Analysis

Pooled covariance



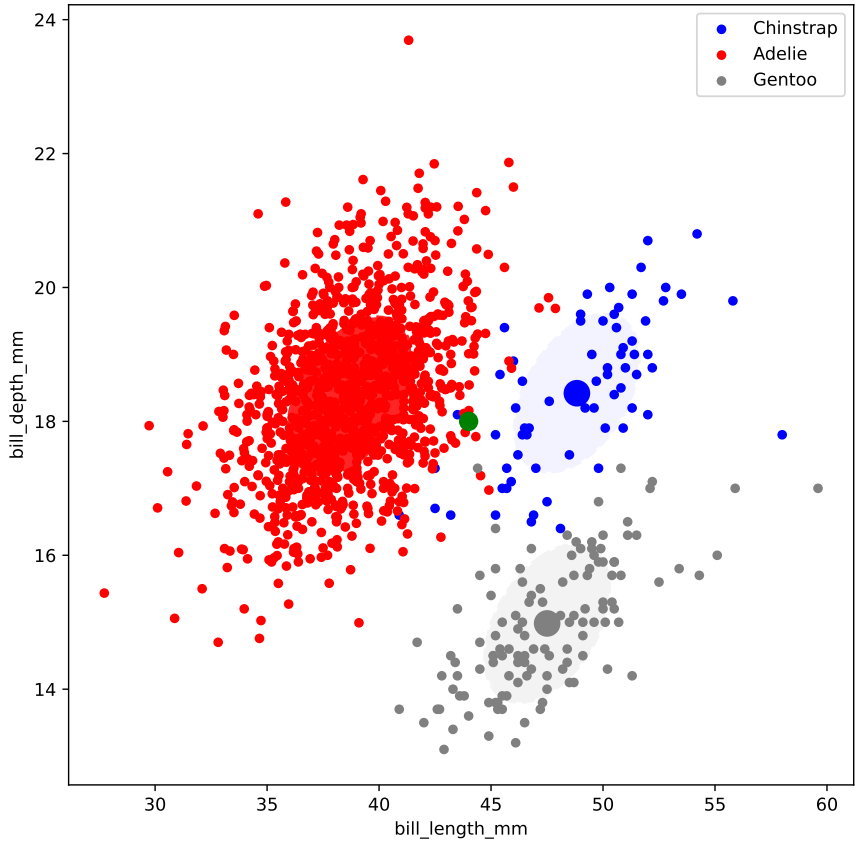
Stack the data centered by class, then compute a covariance.

Common distance



Using a common distance seems reasonable here.

Class imbalance



What if one class is much more common than others? (Opacity proportional to `.value_counts()`)

Linear Discriminant Analysis

Let π_c denote `df['species'].value_counts(normalize=True)`.
Rule: assign $\mathbf{x}_{test}$ to the class c that minimizes

$$h_c(\mathbf{x}_{test}) = \frac{1}{2} d(W(\mathbf{x}_{test}), W(\mu_c))^2 - \log(\pi_c)$$

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA().fit(X_train, y_train)
lda.predict(X_test)
```

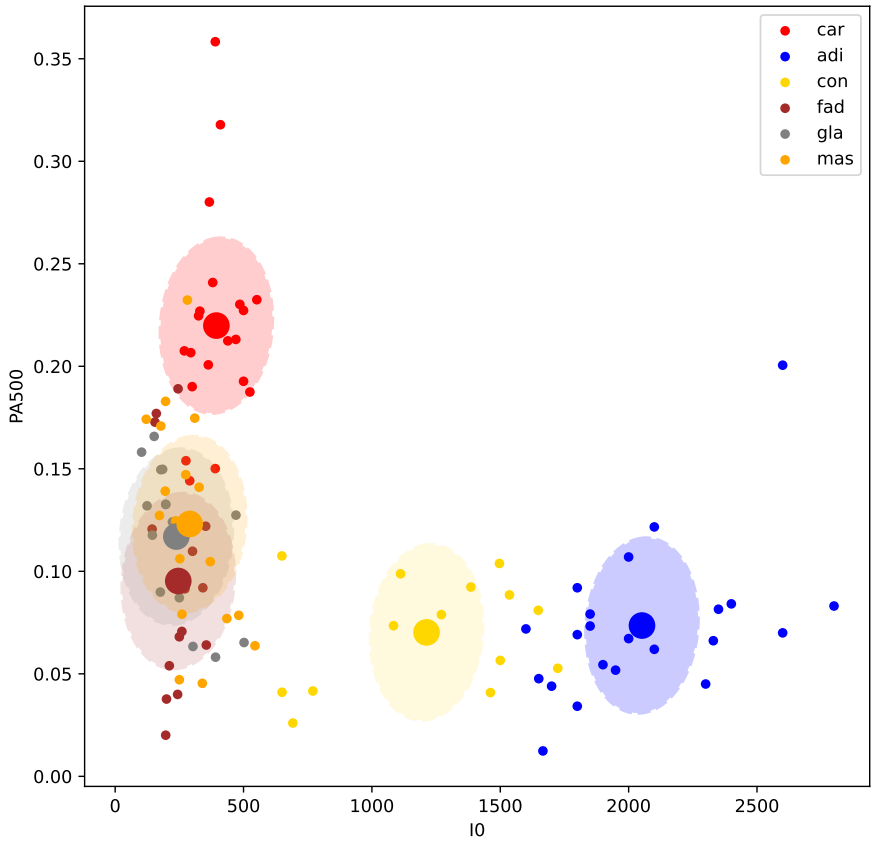
```
array(['Chinstrap'], dtype='<U9')
```

```
lda.predict_proba(X_test)
```

```
array([[0.13111557, 0.86596279, 0.00292164]])
```

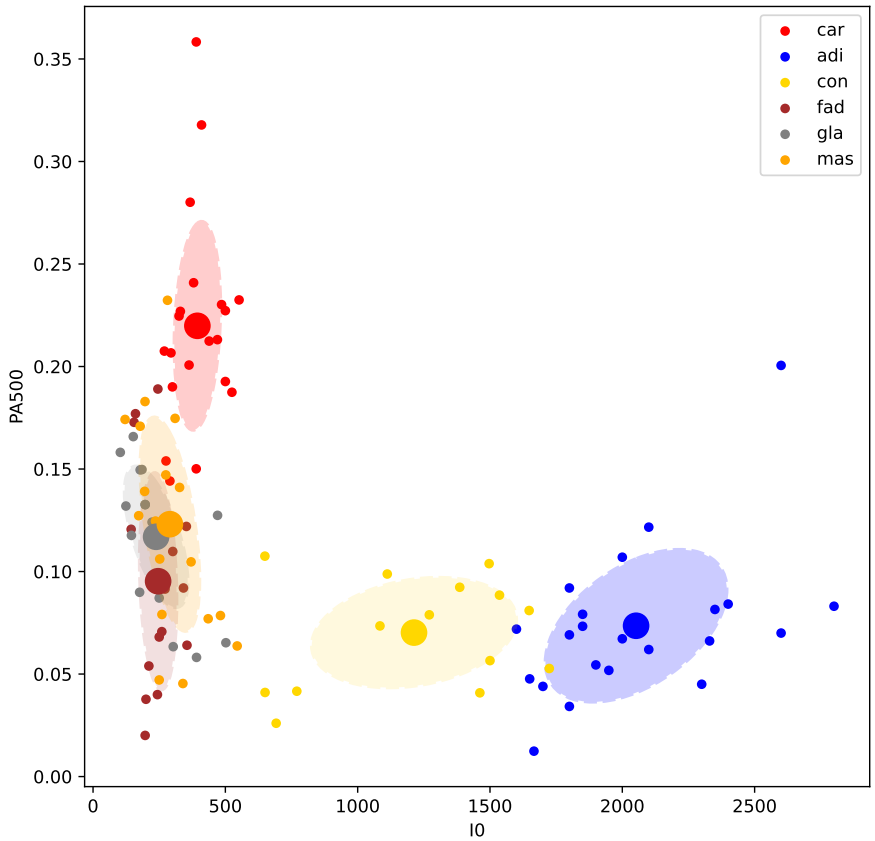
Predicted probabilities are proprtional to $e^{-h_c(\mathbf{x}_{test})}$.

When a common distance is not appropriate



The cancer classes are quite different from each other.

Quadratic Discriminant Analysis



We can allow for difference distance / covariance per class.
In addition to taking π_c into account, the estimated covariances also enter into the rule.

Where do these rules come from?

Under the hood, these methods model the features in each class using a *statistical model*.

The model fits a histogram that is a multivariate version of the *bell curve*.

The class centroids and covariances are part of the model.

Having "fit" the parameters, we can run `.predict_proba(X_test)` and assign to class with highest predicted probability.

No tuning parameters!