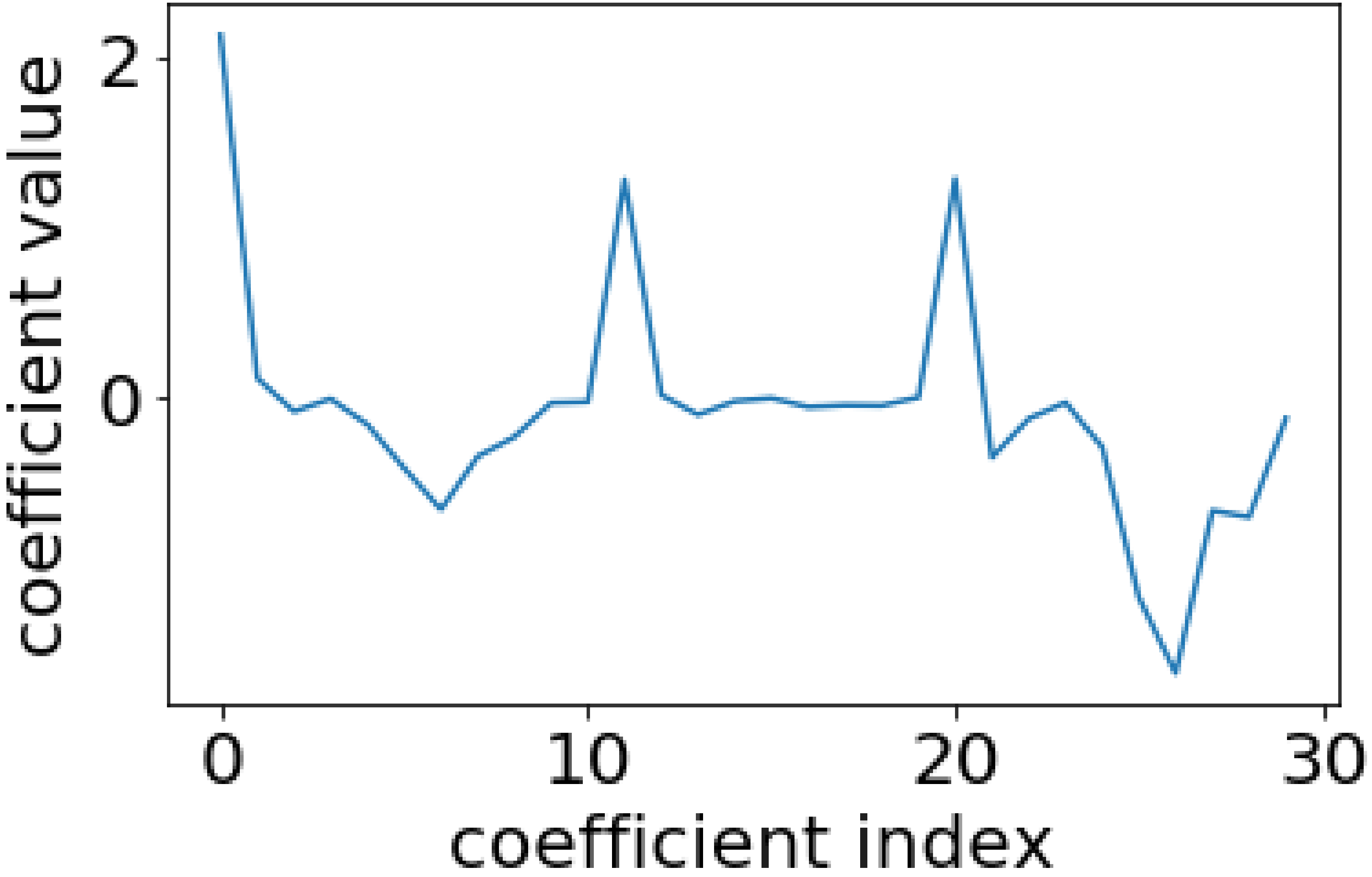
Logistic regression and regularization

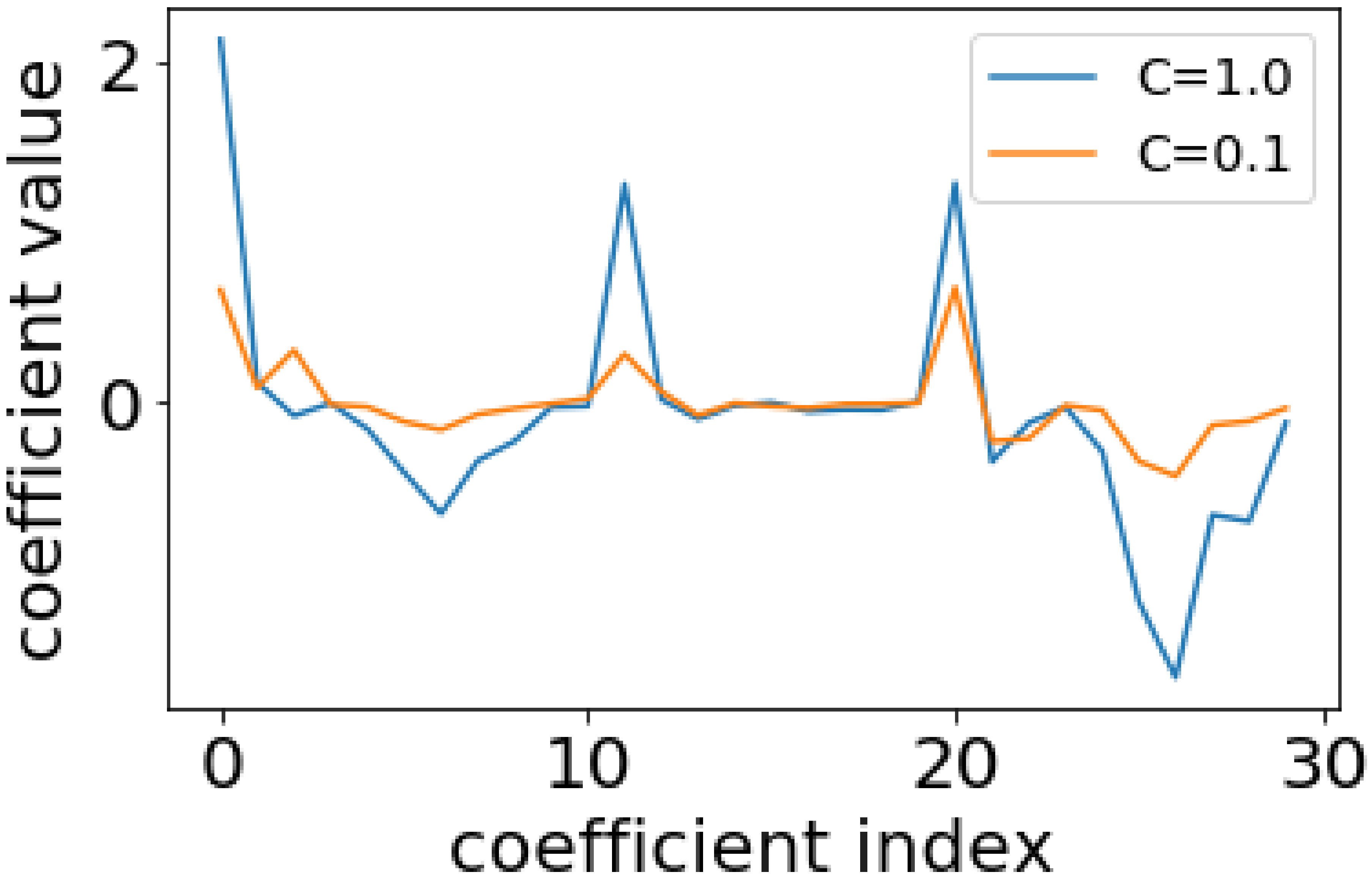
LINEAR CLASSIFIERS IN PYTHON

Michael (Mike) Gelbart
Instructor, The University of British Columbia

Regularized logistic regression



Regularized logistic regression



How does regularization affect training accuracy?

```
lr_weak_reg = LogisticRegression(C=100)
lr_strong_reg = LogisticRegression(C=0.01)

lr_weak_reg.fit(X_train, y_train)
lr_strong_reg.fit(X_train, y_train)

lr_weak_reg.score(X_train, y_train)
lr_strong_reg.score(X_train, y_train)
```

```
1.0
0.92
```

$$\text{regularized loss} = \text{original loss} + \text{large coefficient penalty}$$

- more regularization: lower training accuracy

How does regularization affect test accuracy?

```
lr_weak_reg.score(X_test, y_test)
```

```
0.86
```

```
lr_strong_reg.score(X_test, y_test)
```

```
0.88
```

$$\text{regularized loss} = \text{original loss} + \text{large coefficient penalty}$$

- more regularization: lower training accuracy
- more regularization: (almost always) higher test accuracy

L1 vs. L2 regularization

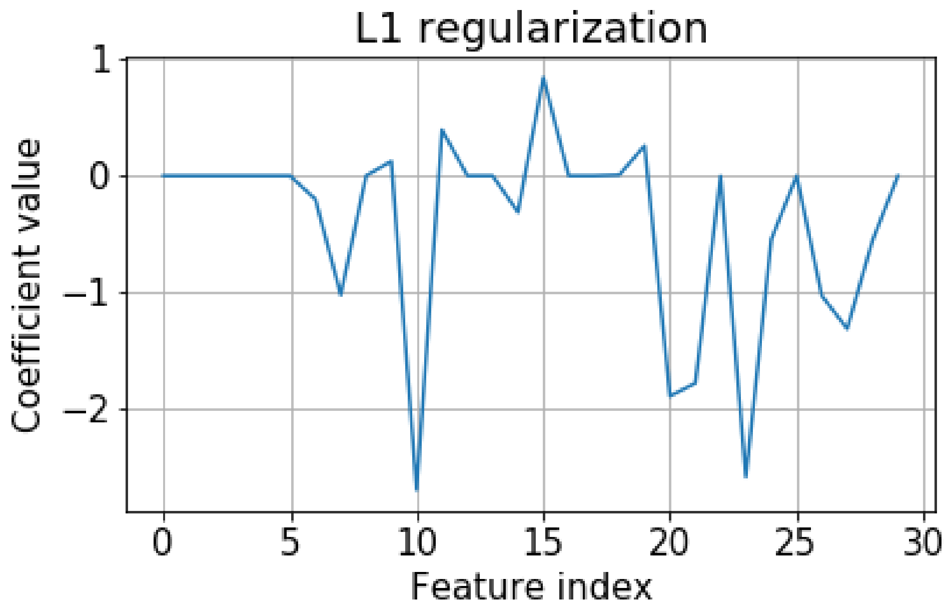
- Lasso = linear regression with L1 regularization
- Ridge = linear regression with L2 regularization
- For other models like logistic regression we just say L1, L2, etc.

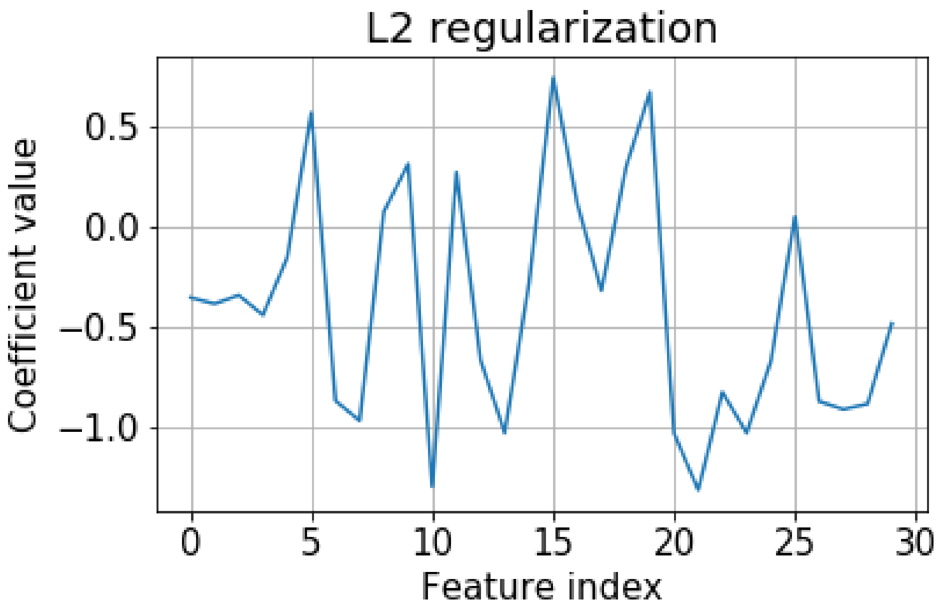
```
lr_L1 = LogisticRegression(solver='liblinear', penalty='l1')
lr_L2 = LogisticRegression() # penalty='l2' by default

lr_L1.fit(X_train, y_train)
lr_L2.fit(X_train, y_train)
```

```
plt.plot(lr_L1.coef_.flatten())
plt.plot(lr_L2.coef_.flatten())
```

L2 vs. L1 regularization





Let's practice!

LINEAR CLASSIFIERS IN PYTHON

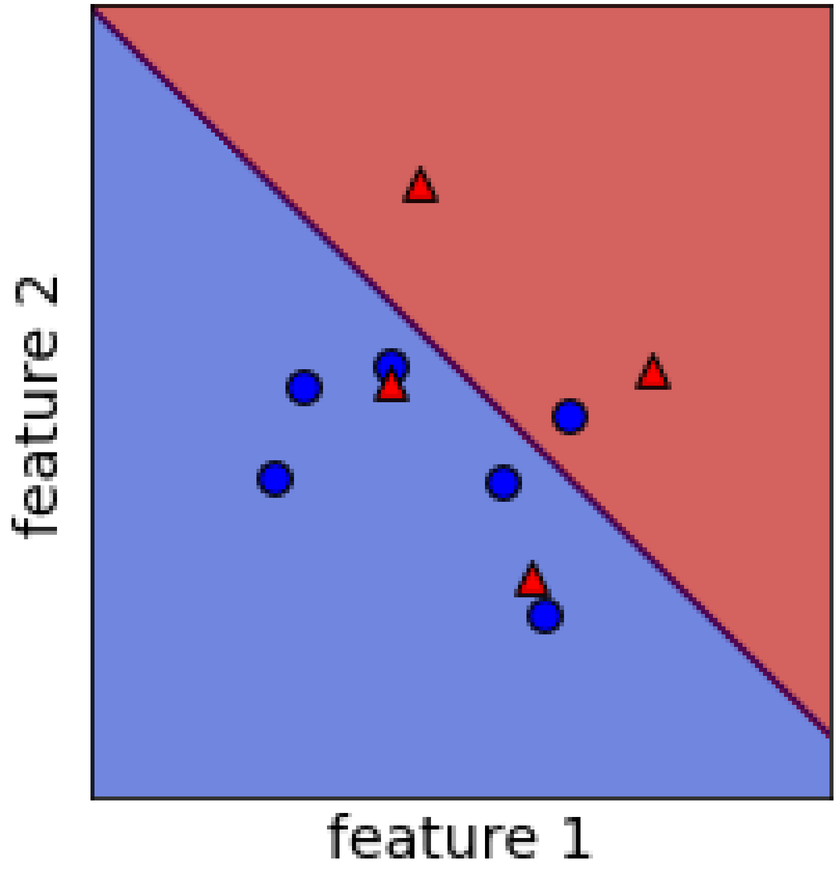
Logistic regression and probabilities

LINEAR CLASSIFIERS IN PYTHON

Michael (Mike) Gelbart
Instructor, The University of British Columbia

Logistic regression probabilities

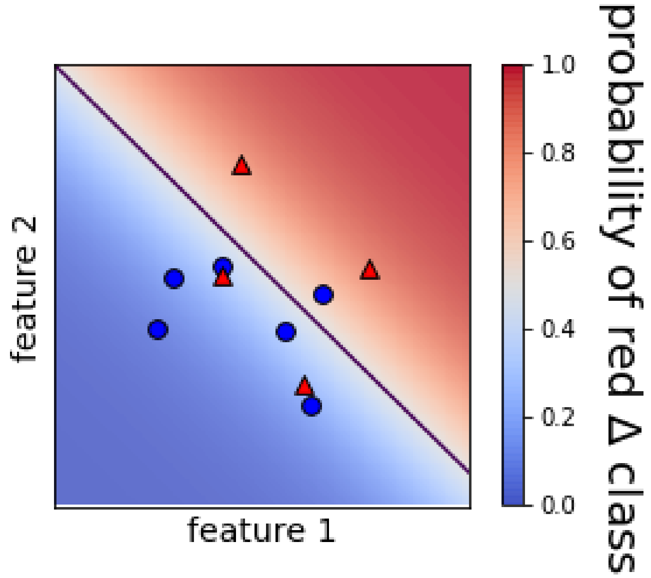
Without regularization
$(C = 10^8)$:



- model coefficients:
  `[[1.55 1.57]]`
- model intercept: `[-0.64]`

Logistic regression probabilities

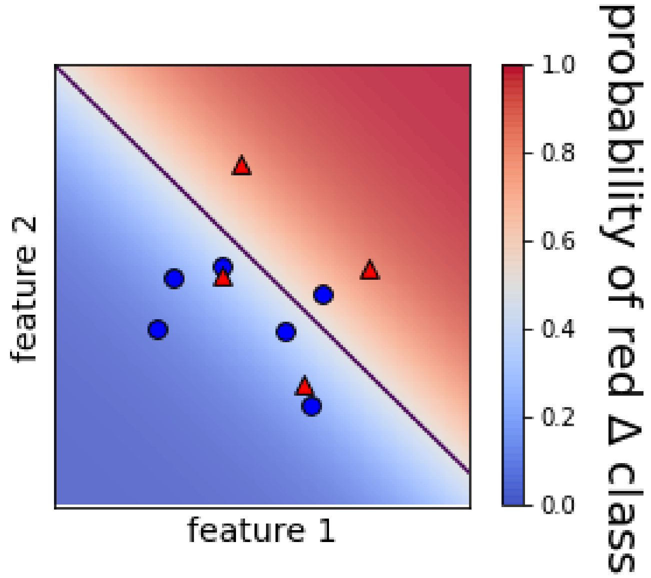
Without regularization
$(C = 10^8)$:



- model coefficients:
  `[[1.55 1.57]]`
- model intercept: `[-0.64]`

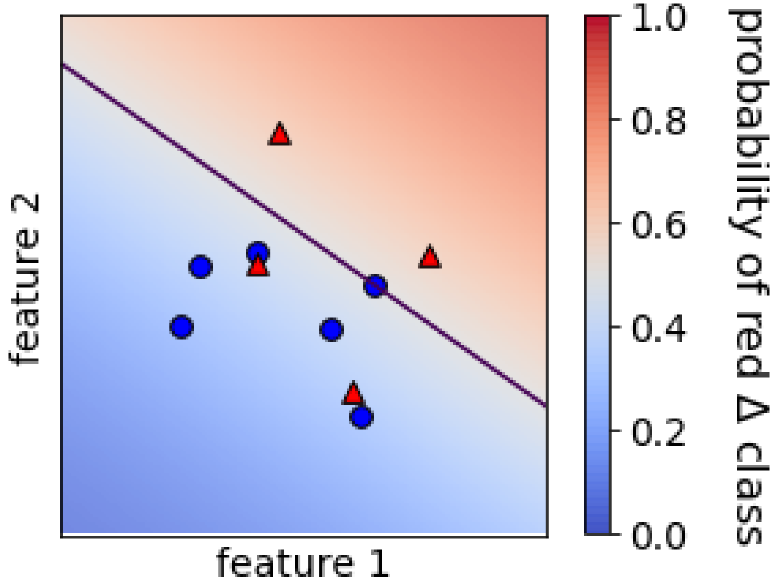
Logistic regression probabilities

Without regularization ($C = 10^8$):



- model coefficients: `[[1.55 1.57]]`
- model intercept: `[-0.64]`

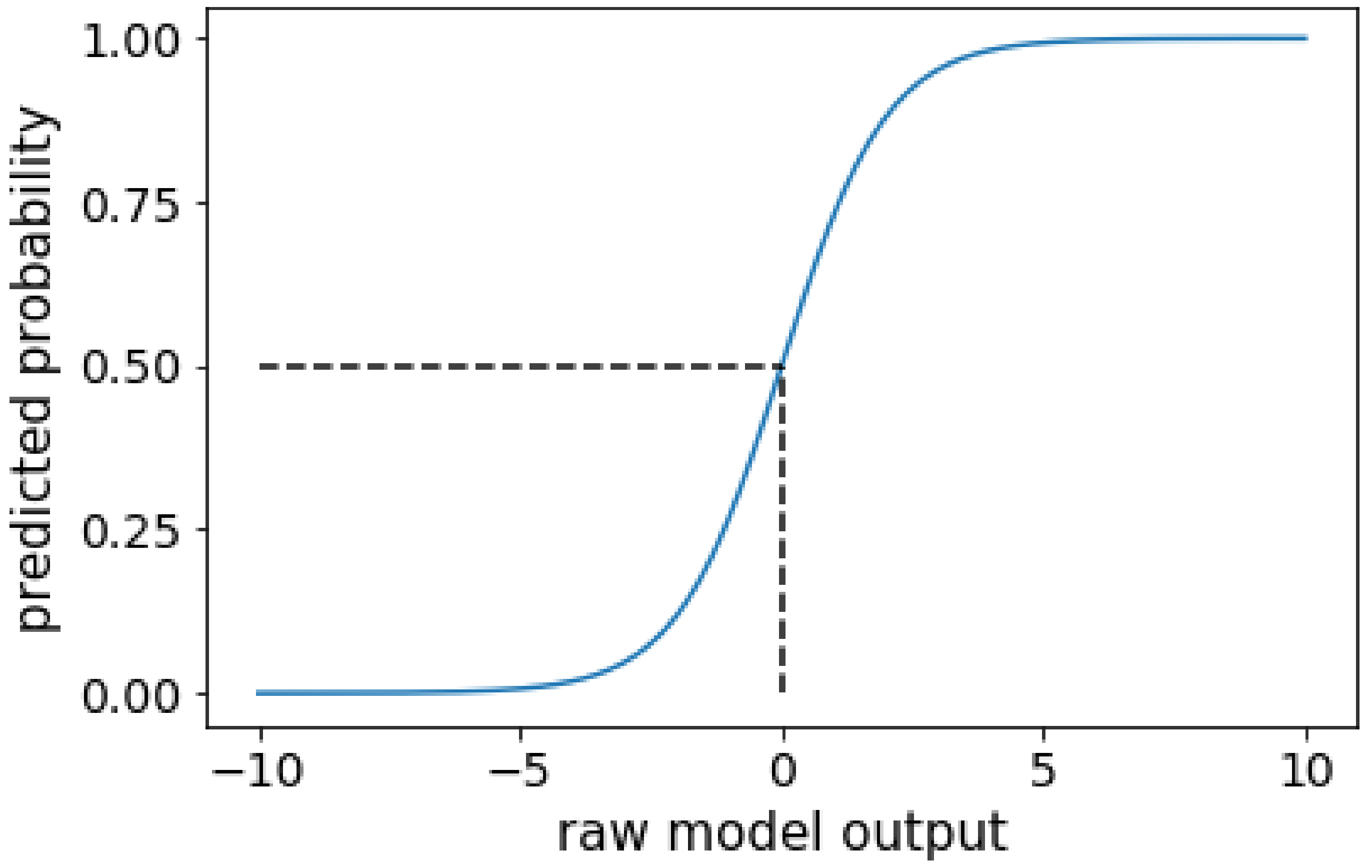
With regularization ($C = 1$):



- model coefficients: `[[0.45 0.64]]`
- model intercept: `[-0.26]`

How are these probabilities computed?

- logistic regression predictions: sign of raw model output
- logistic regression probabilities: "squashed" raw model output



Let's practice!

LINEAR CLASSIFIERS IN PYTHON

Multi-class logistic regression

LINEAR CLASSIFIERS IN PYTHON

Michael (Mike) Gelbart
Instructor, The University of British Columbia

Combining binary classifiers with one-vs-rest

```
lr0.fit(X, y==0)

lr1.fit(X, y==1)

lr2.fit(X, y==2)
```

```
# get raw model output
lr0.decision_function(X)[0]
```

```
6.124
```

```
lr1.decision_function(X)[0]
```

```
-5.429
```

```
lr2.decision_function(X)[0]
```

```
-7.532
```

```
lr = LogisticRegression(multi_class='ovr')
lr.fit(X, y)
lr.predict(X)[0]
```

```
0
```

One-vs-rest:

- fit a binary classifier for each class
- predict with all, take largest output
- pro: simple, modular
- con: not directly optimizing accuracy
- common for SVMs as well
- can produce probabilities

"Multinomial" or "softmax":

- fit a single classifier for all classes
- prediction directly outputs best class
- con: more complicated, new code
- pro: tackle the problem directly
- possible for SVMs, but less common
- can produce probabilities

Model coefficients for multi-class

```
lr_ovr = LogisticRegression(multi_class='ovr')

lr_ovr.fit(X,y)

lr_ovr.coef_.shape
```

```
(3,13)
```

```
lr_ovr.intercept_.shape
```

```
(3,)
```

```
lr_mn = LogisticRegression(multi_class="multinomial")
lr_mn.fit(X,y)

lr_mn.coef_.shape
```

```
(3,13)
```

```
lr_mn.intercept_.shape
```

```
(3,)
```

Let's practice!

LINEAR CLASSIFIERS IN PYTHON