# Welcome to the course!

LINEAR CLASSIFIERS IN PYTHON

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

# Assumed knowledge

In this course we'll assume you have some prior exposure to:

- Python, at the level of *Intermediate Python for Data Science*
- scikit-learn, at the level of *Supervised Learning with scikit-learn*
- supervised learning, at the level of *Supervised Learning with scikit-learn*

# Fitting and predicting

```
import sklearn.datasets

newsgroups = sklearn.datasets.fetch_20newsgroups_vectorized()

X, y = newsgroups.data, newsgroups.target
```

```
X.shape
```

```
(11314, 130107)
```

```
y.shape
```

```
(11314,)
```

# Fitting and predicting (cont.)

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X,y)
```

```
y_pred = knn.predict(X)
```

# Model evaluation

```
knn.score(X,y)
```

```
0.99991
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
knn.fit(X_train, y_train)

knn.score(X_test, y_test)
```

```
0.66242
```

# Let's practice!

LINEAR CLASSIFIERS IN PYTHON

# Applying logistic regression and SVM

LINEAR CLASSIFIERS IN PYTHON

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

# Using LogisticRegression

```python
from sklearn.linear_model import LogisticRegression
```

```python
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr.predict(X_test)
lr.score(X_test, y_test)
```

# LogisticRegression example

```
import sklearn.datasets
wine = sklearn.datasets.load_wine()
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(wine.data, wine.target)
lr.score(wine.data, wine.target)
```

```
0.972
```

```
lr.predict_proba(wine.data[:1])
```

```
array([[  9.951e-01,   4.357e-03,   5.339e-04]])
```

# Using LinearSVC

`LinearSVC` works the same way:

```
import sklearn.datasets

wine = sklearn.datasets.load_wine()
from sklearn.svm import LinearSVC

svm = LinearSVC()

svm.fit(wine.data, wine.target)
svm.score(wine.data, wine.target)
```

```
0.893
```

# Using SVC

```
import sklearn.datasets
wine = sklearn.datasets.load_wine()
from sklearn.svm import SVC
svm = SVC() # default hyperparameters
svm.fit(wine.data, wine.target);
svm.score(wine.data, wine.target)
```

```
1.
```

Model complexity review:

- **Underfitting**: model is too simple, low training accuracy
- **Overfitting**: model is too complex, low test accuracy

# Let's practice!

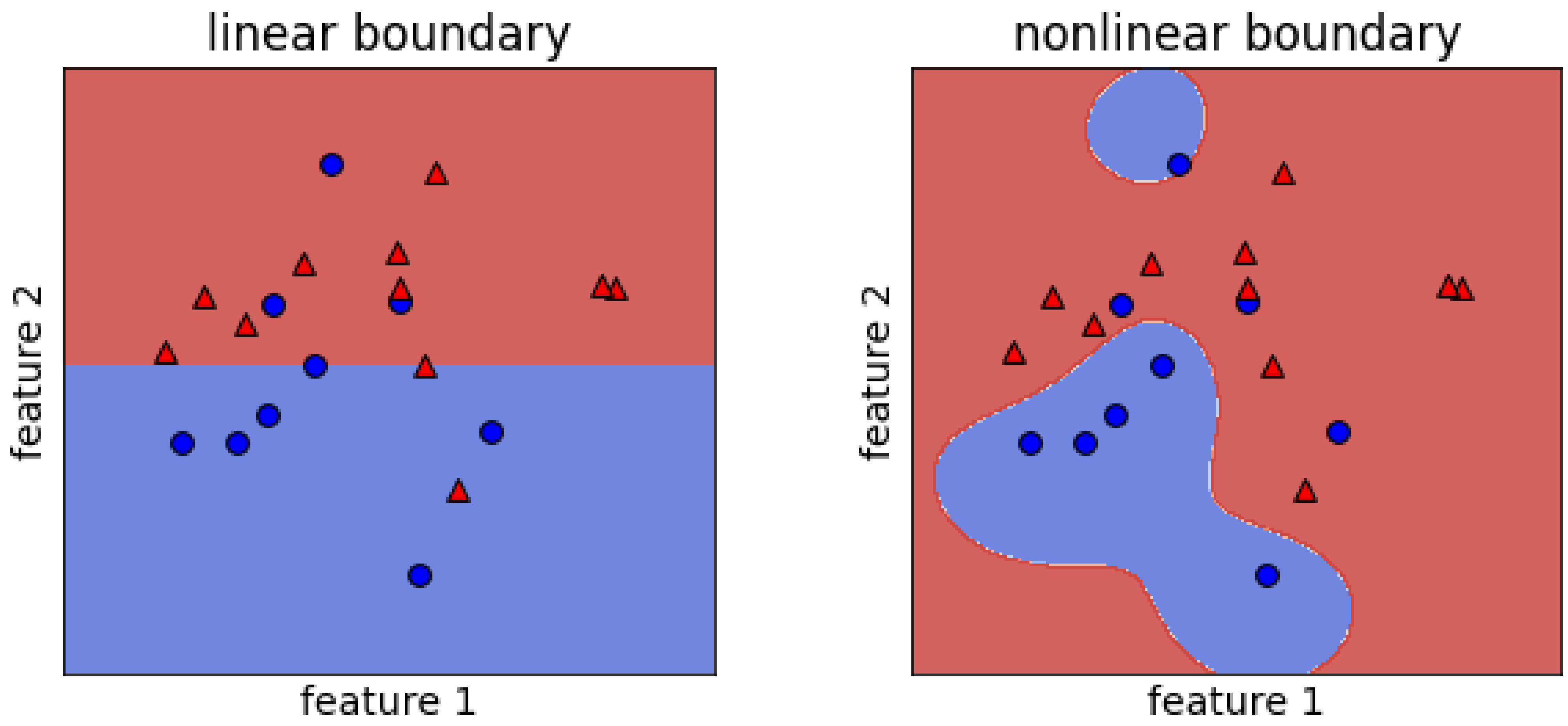
LINEAR CLASSIFIERS IN PYTHON

# Linear decision boundaries

LINEAR CLASSIFIERS IN PYTHON

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

# Linear decision boundaries

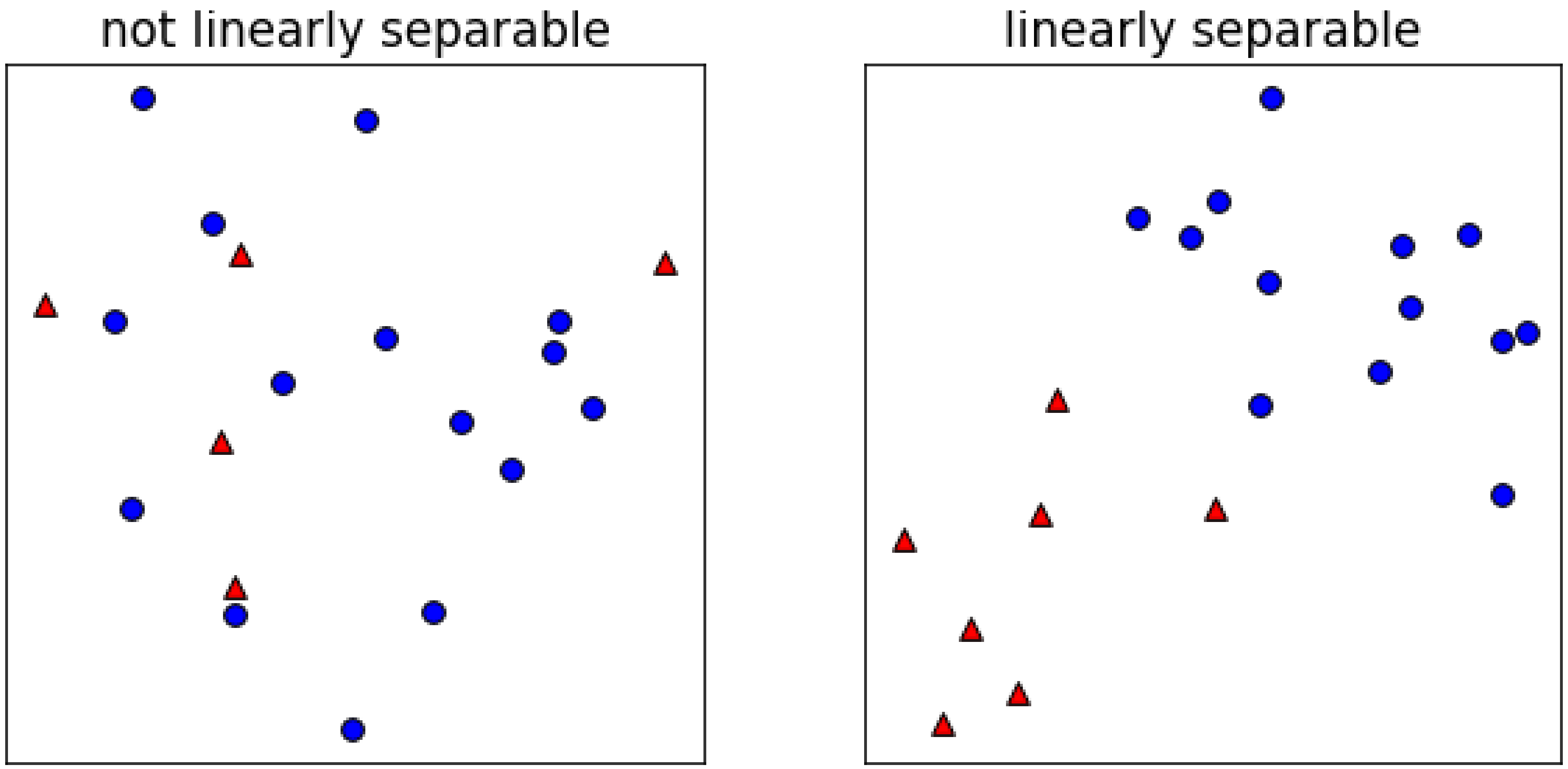


# Definitions

Vocabulary:

- **classification**: learning to predict categories
- **decision boundary**: the surface separating different predicted classes
- **linear classifier**: a classifier that learns linear decision boundaries
  - e.g., logistic regression, linear SVM
- **linearly separable**: a data set can be perfectly explained by a linear classifier

# Linearly separable data



# Let's practice!

LINEAR CLASSIFIERS IN PYTHON

# Linear classifiers: prediction equations

LINEAR CLASSIFIERS IN PYTHON

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

# Dot Products

```
x = np.arange(3)
x
```

```
array([0, 1, 2])
```

```
y = np.arange(3,6)
y
```

```
array([3, 4, 5])
```

```
x*y
```

```
array([0, 4, 10])
```

```
np.sum(x*y)
```

```
14
```

```
x@y
```

```
14
```

- `x@y` is called the dot product of `x` and `y`, and is written $x \cdot y$.

# Linear classifier prediction

- $\text{raw model output} = \text{coefficients} \cdot \text{features} + \text{intercept}$
- Linear classifier prediction: compute raw model output, check the **sign**
  - if positive, predict one class
  - if negative, predict the other class
- This is the same for logistic regression and linear SVM
  - `fit` is different but `predict` is the same

# How LogisticRegression makes predictions

$$\text{raw model output} = \text{coefficients} \cdot \text{features} + \text{intercept}$$

```
lr = LogisticRegression()

lr.fit(X,y)

lr.predict(X)[10]
```

```
0
```

```
lr.predict(X)[20]
```

```
1
```

# How LogisticRegression makes predictions (cont.)

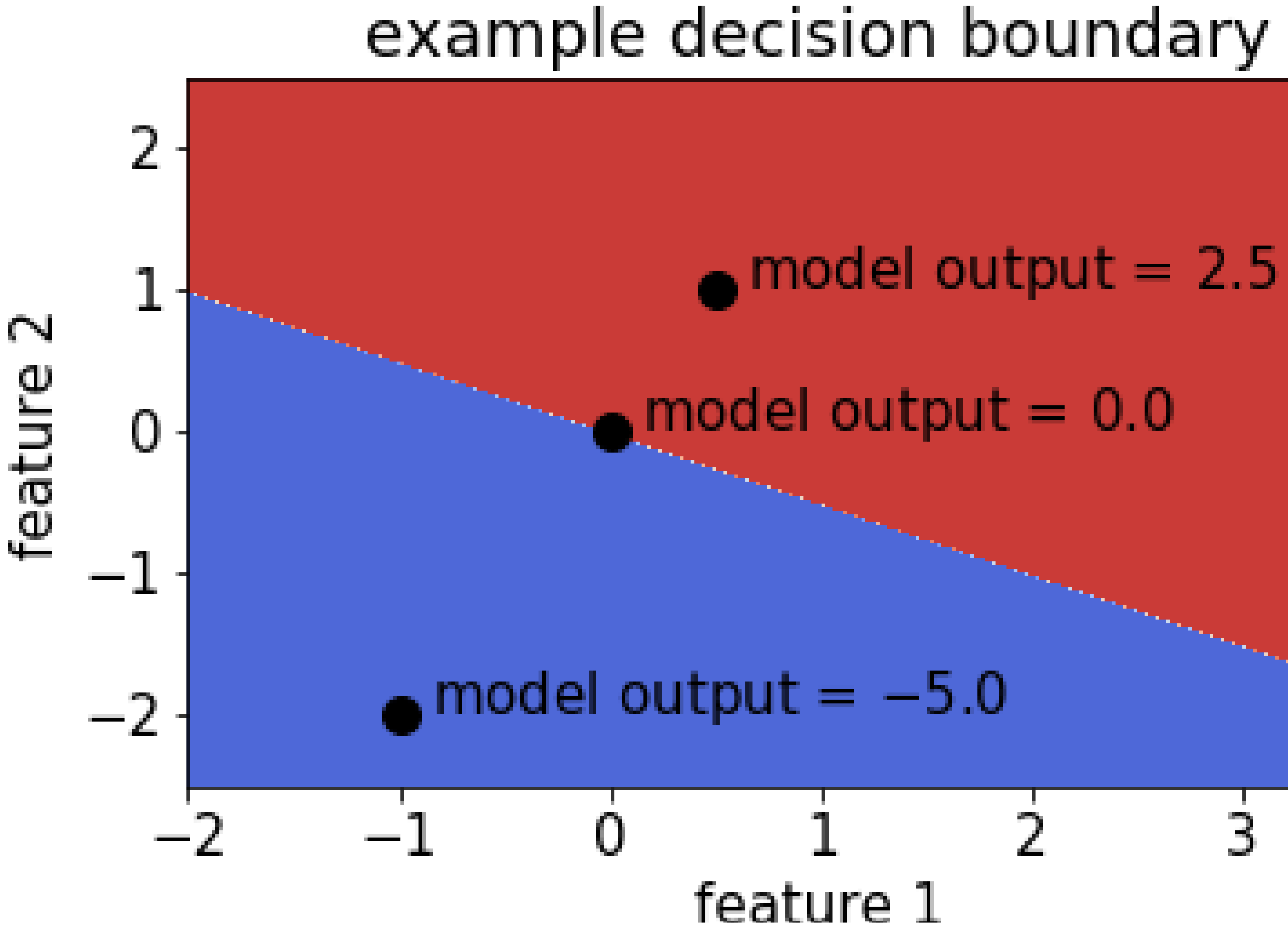
```
lr.coef_ @ X[10] + lr.intercept_ # raw model output
```

```
array([-33.78572166])
```

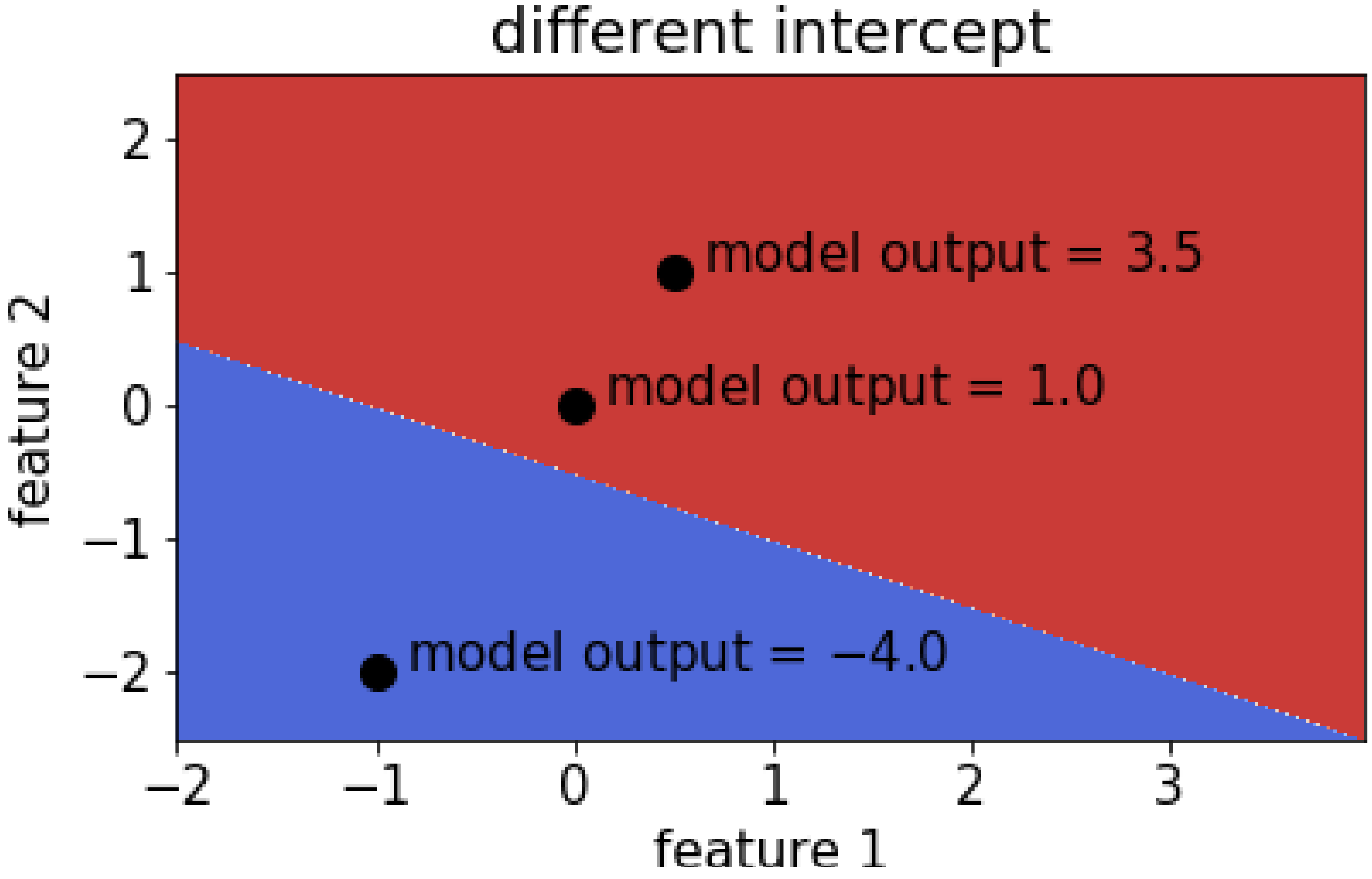
```
lr.coef_ @ X[20] + lr.intercept_ # raw model output
```

```
array([ 0.08050621])
```

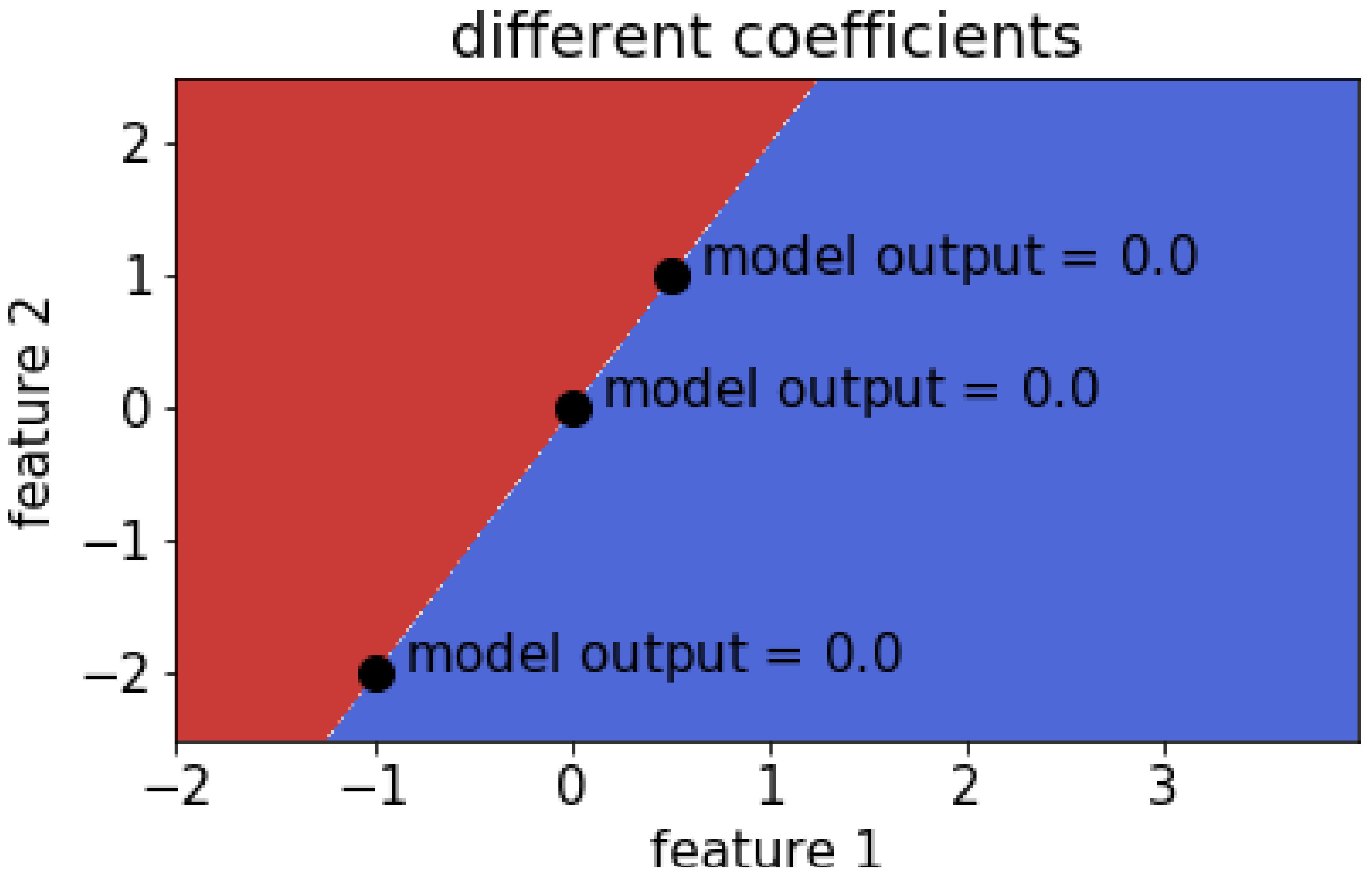
# The raw model output



# The raw model output



# The raw model output



# Let's practice!

LINEAR CLASSIFIERS IN PYTHON

# What is a loss function?

LINEAR CLASSIFIERS IN PYTHON

**Michael Gelbart**

Instructor, The University of British Columbia

# Least squares: the squared loss

- scikit-learn's `LinearRegression` minimizes a loss:

$$\sum_{i=1}^{n} (\text{true } i\text{th target value} - \text{predicted } i\text{th target value})^2$$

- Minimization is with respect to coefficients or parameters of the model.
- Note that in scikit-learn `model.score()` isn't necessarily the loss function.

# Classification errors: the 0-1 loss

- Squared loss not appropriate for classification problems (more on this later).
- A natural loss for classification problem is the number of errors.
- This is the **0-1 loss**: it's 0 for a correct prediction and 1 for an incorrect prediction.
- But this loss is hard to minimize!

# Minimizing a loss

```python
from scipy.optimize import minimize
```

```python
minimize(np.square, 0).x
```

```
array([0.])
```

```python
minimize(np.square, 2).x
```

```
array([-1.88846401e-08])
```

# Let's practice!

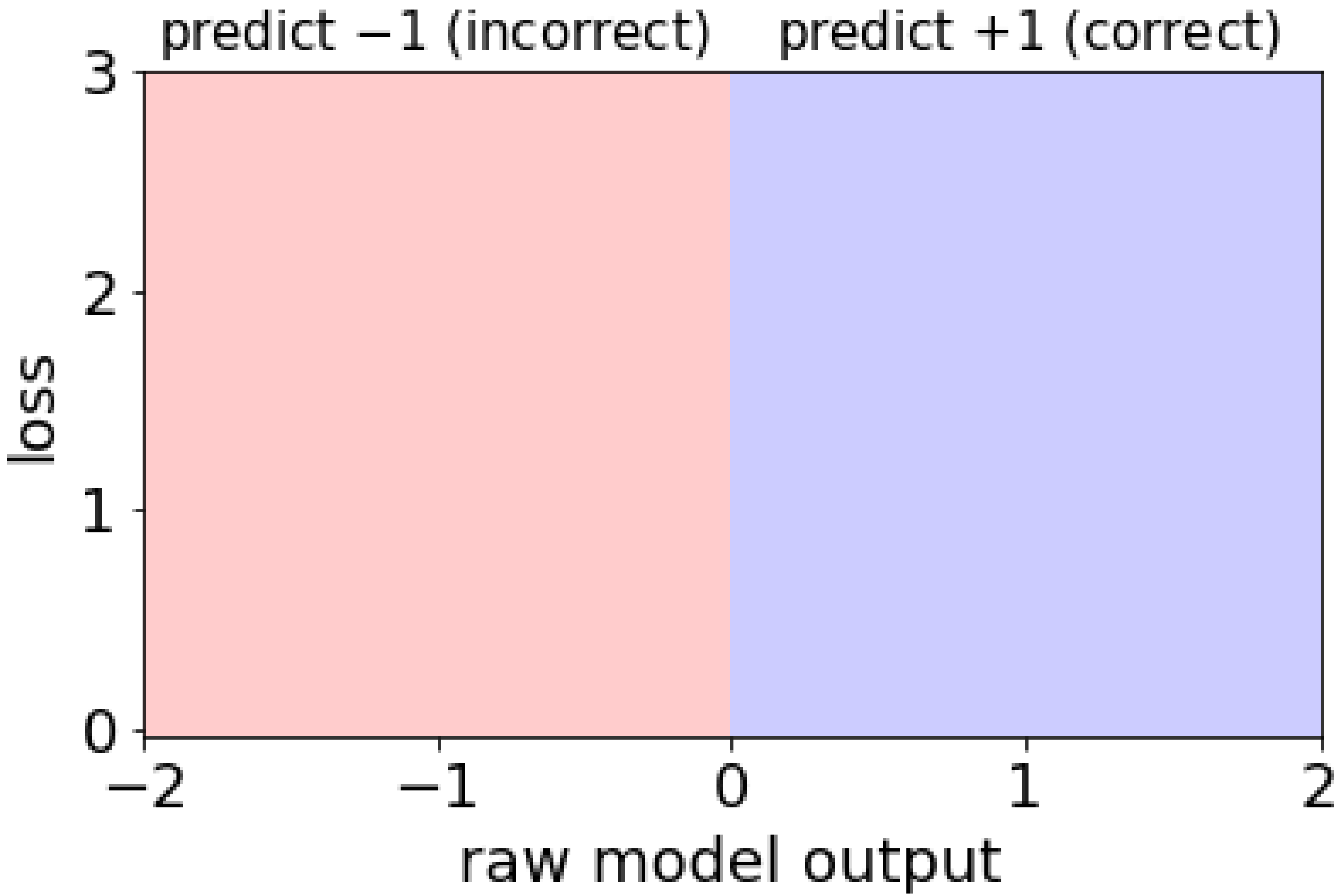
LINEAR CLASSIFIERS IN PYTHON

# Loss function diagrams

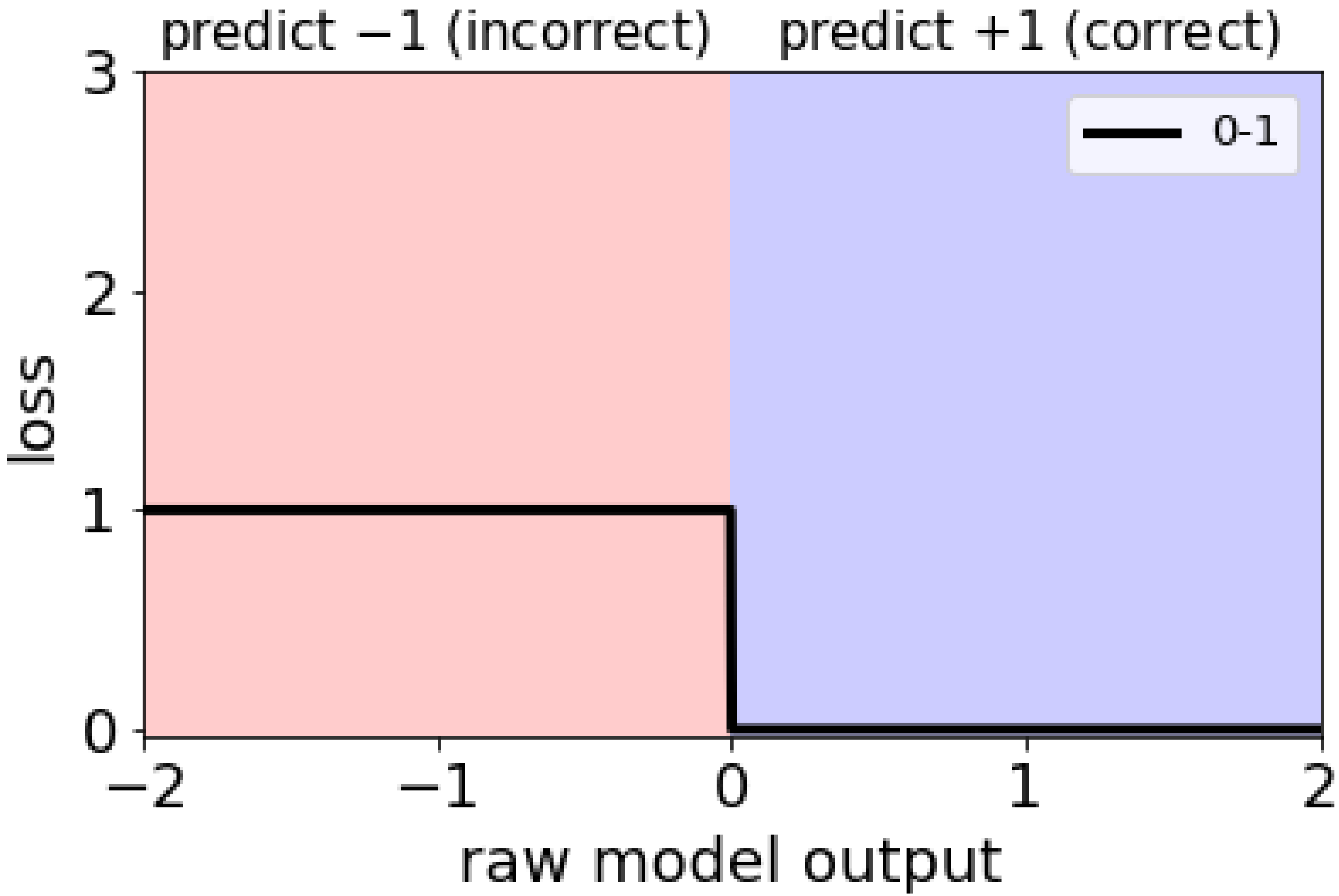
LINEAR CLASSIFIERS IN PYTHON

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

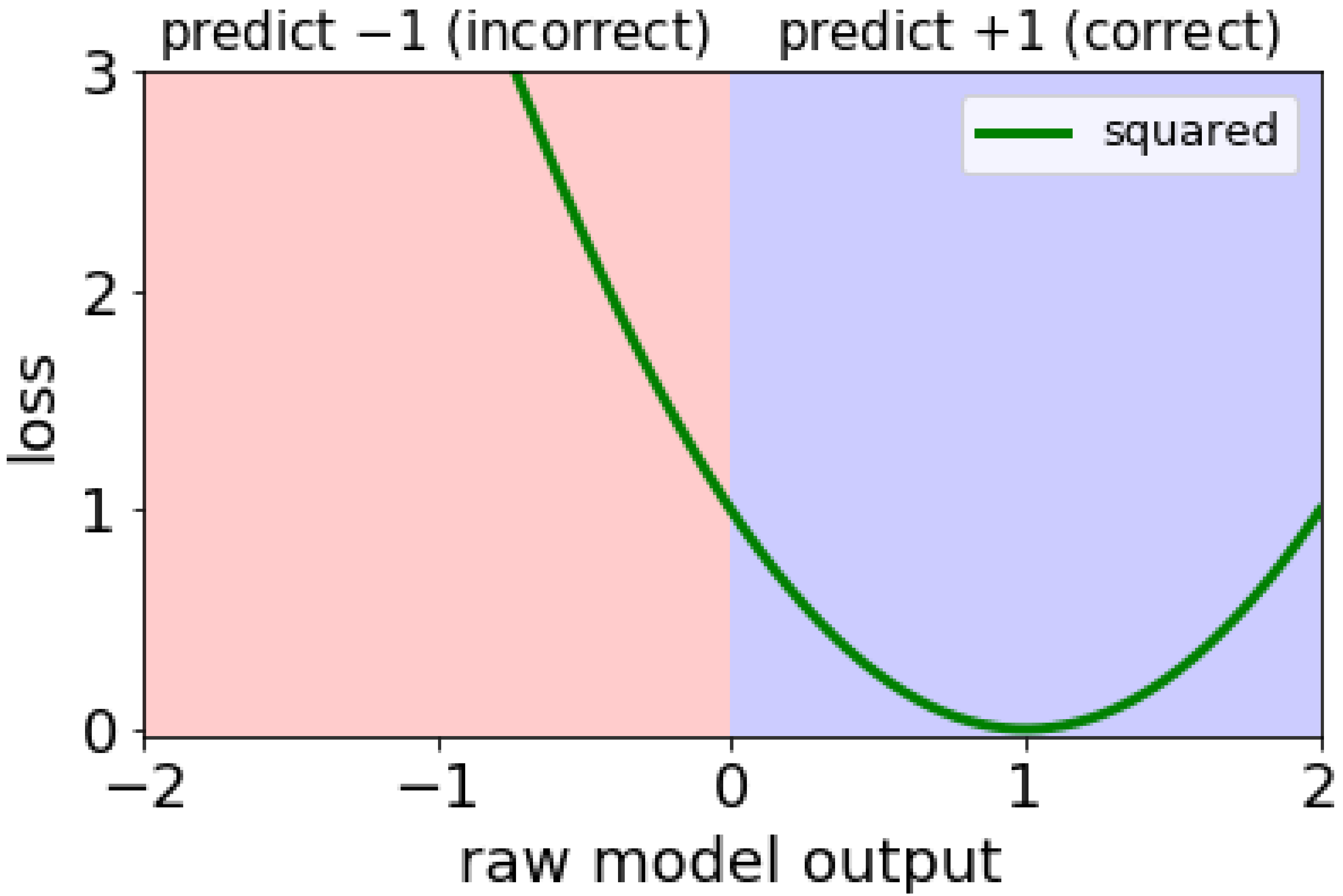
# The raw model output



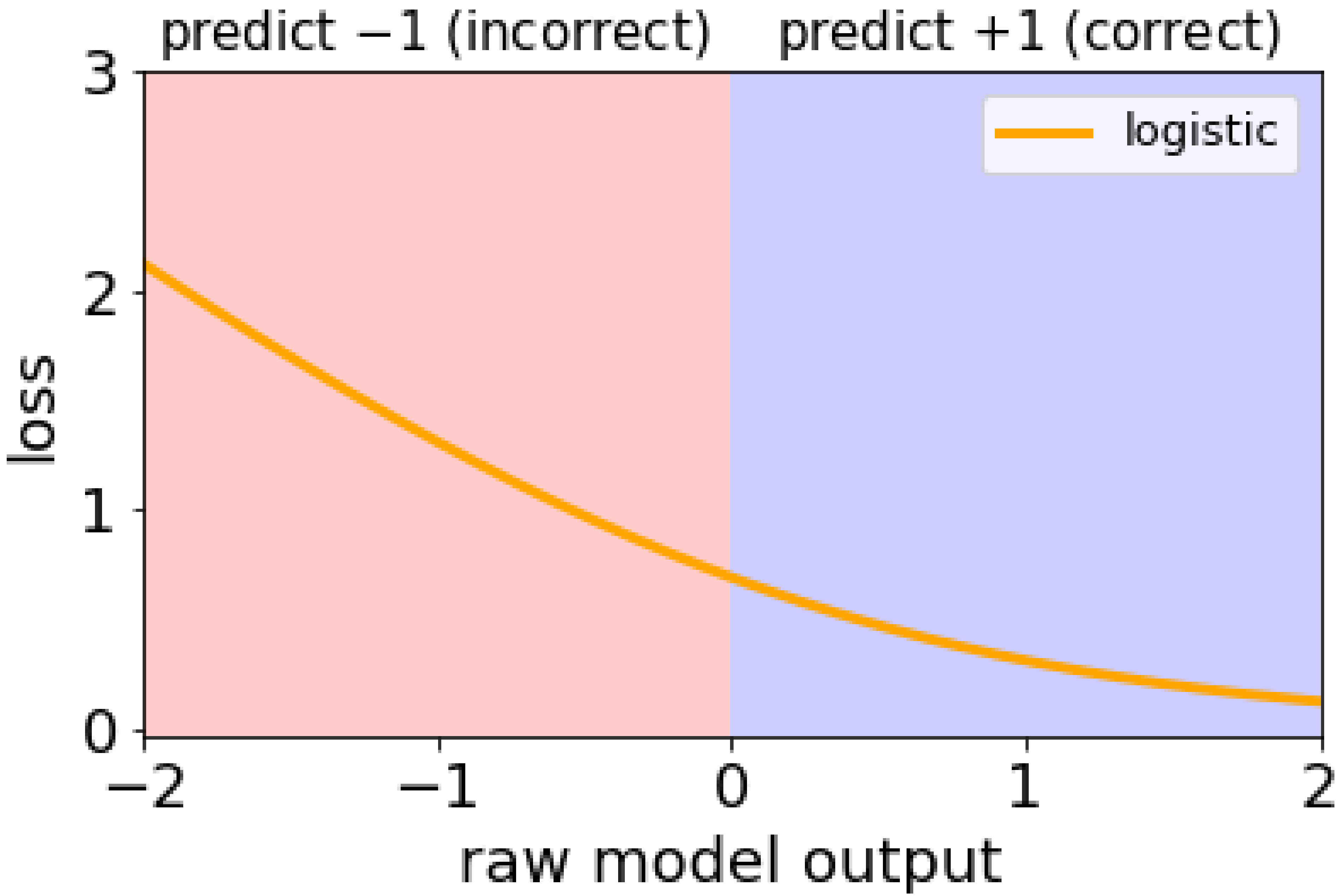
# 0-1 loss diagram



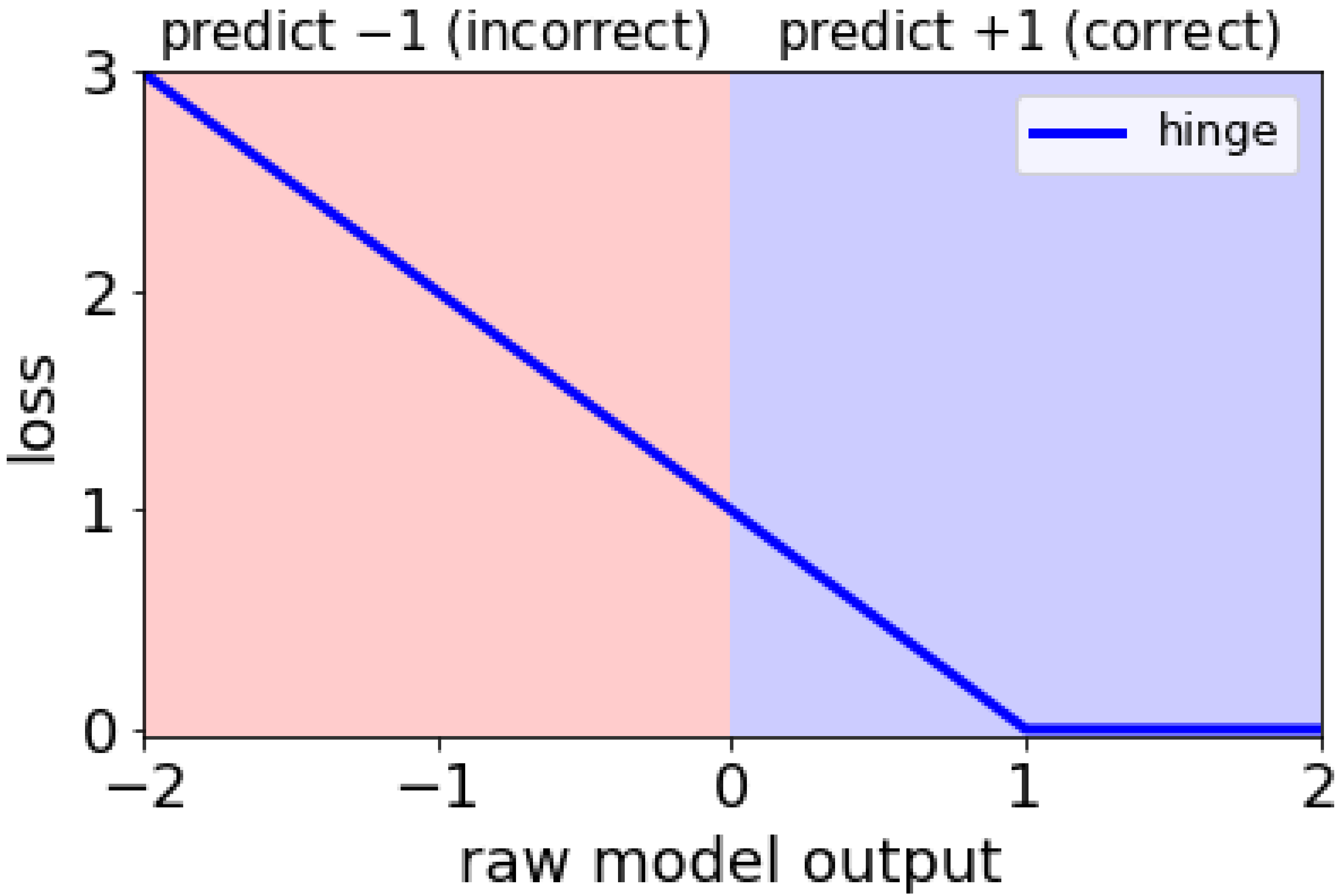
# Linear regression loss diagram



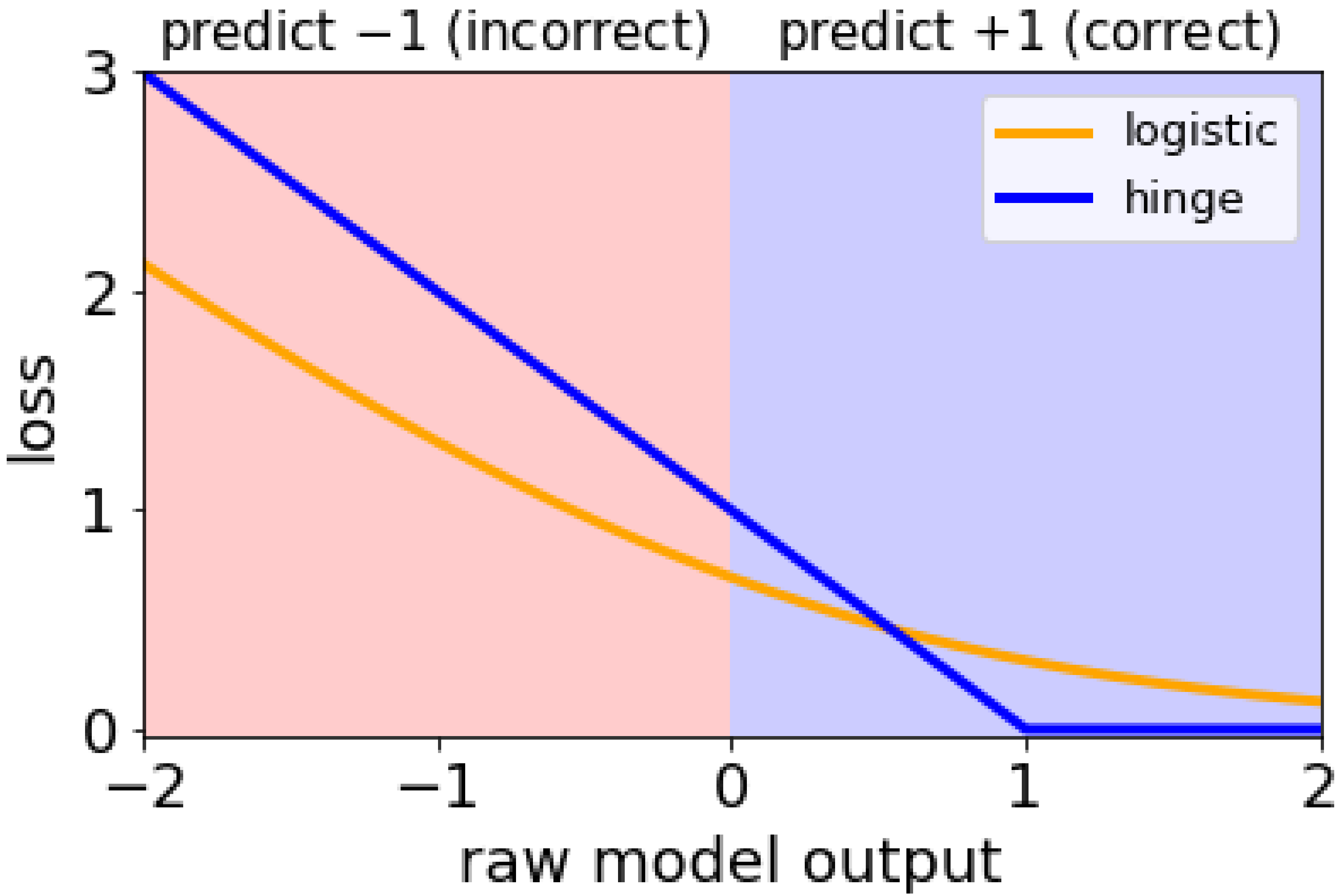
# Logistic loss diagram



# Hinge loss diagram



# Hinge loss diagram



# Let's practice!

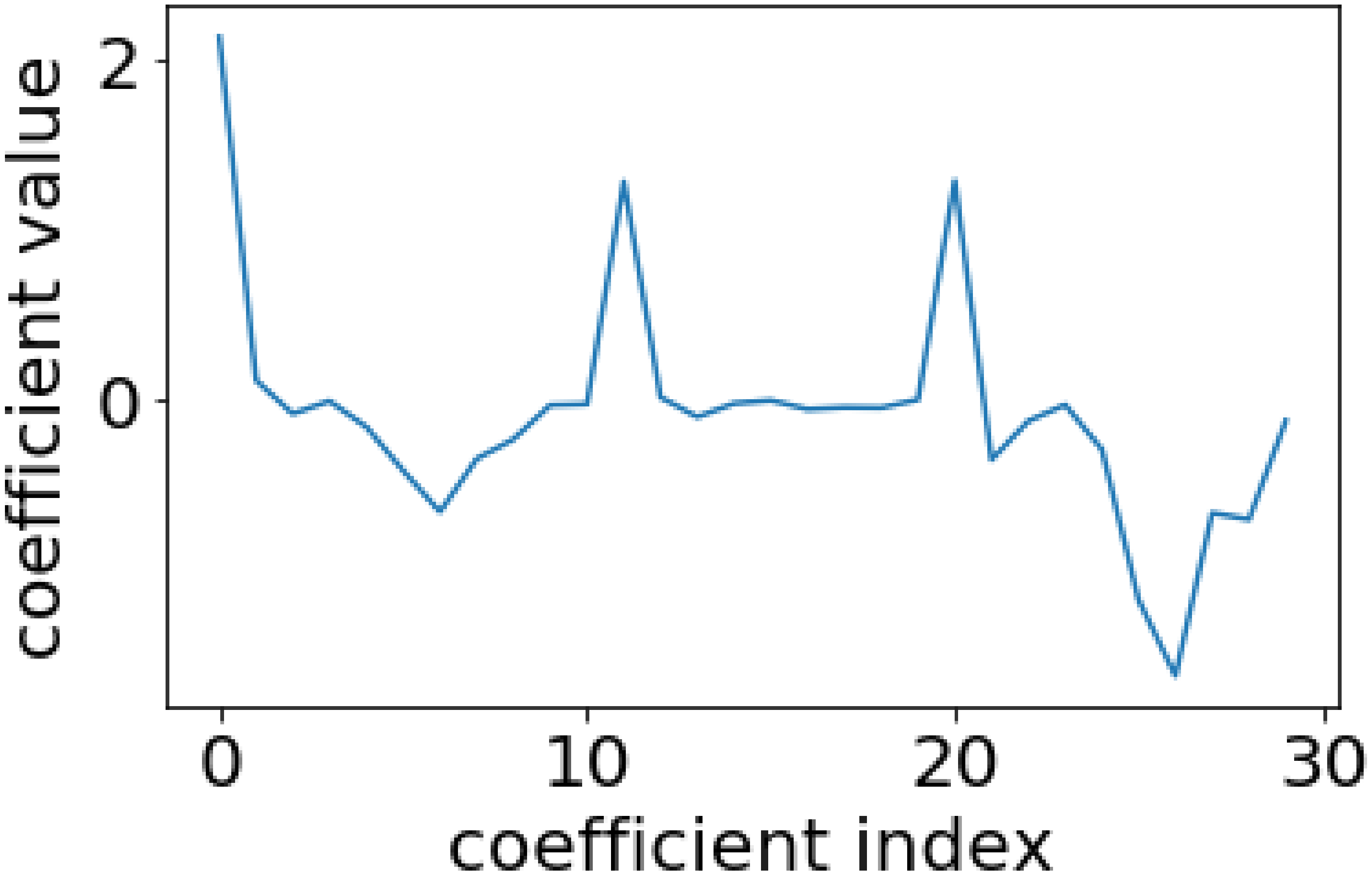
LINEAR CLASSIFIERS IN PYTHON

# Logistic regression and regularization

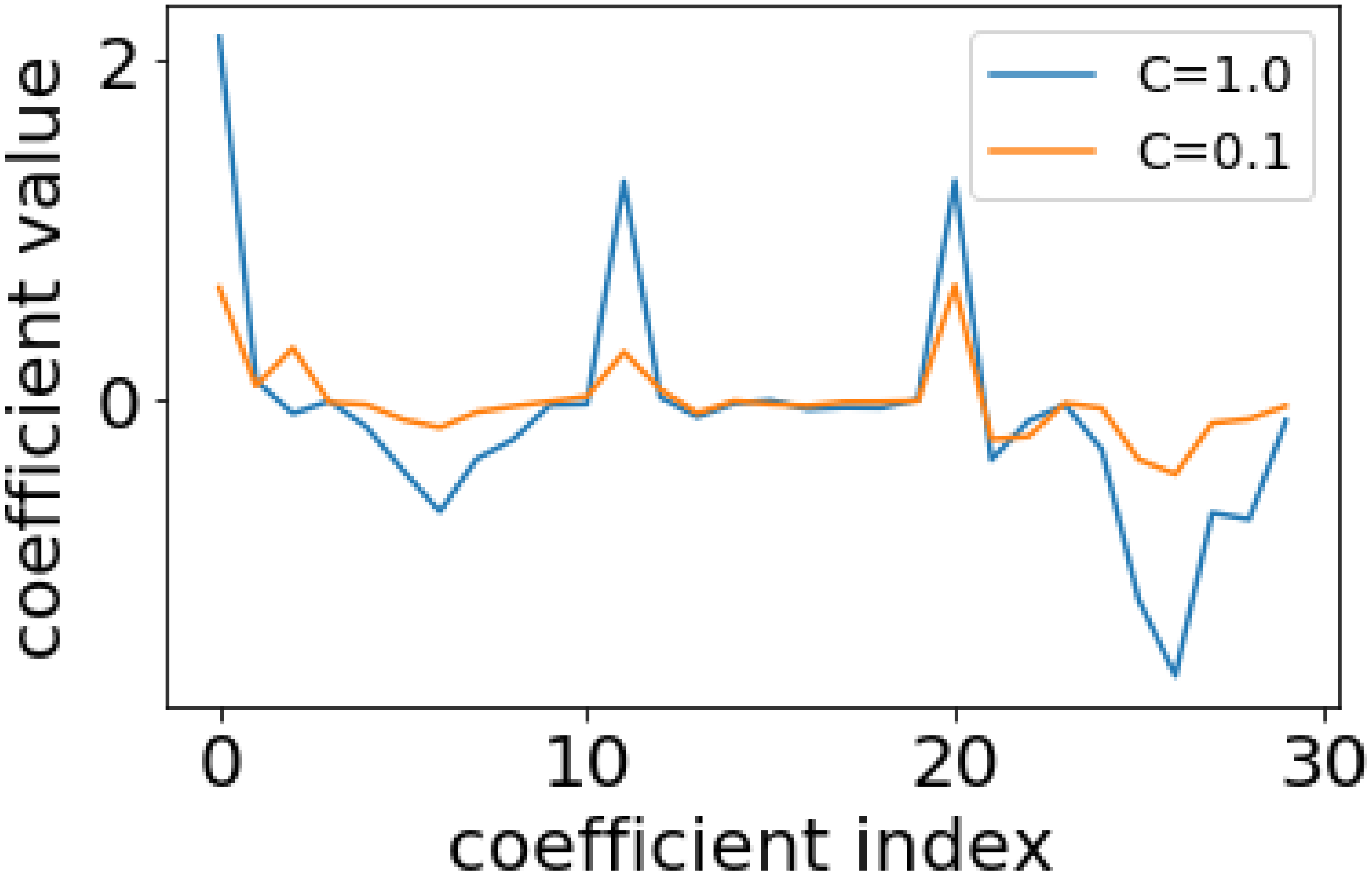
LINEAR CLASSIFIERS IN PYTHON

**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

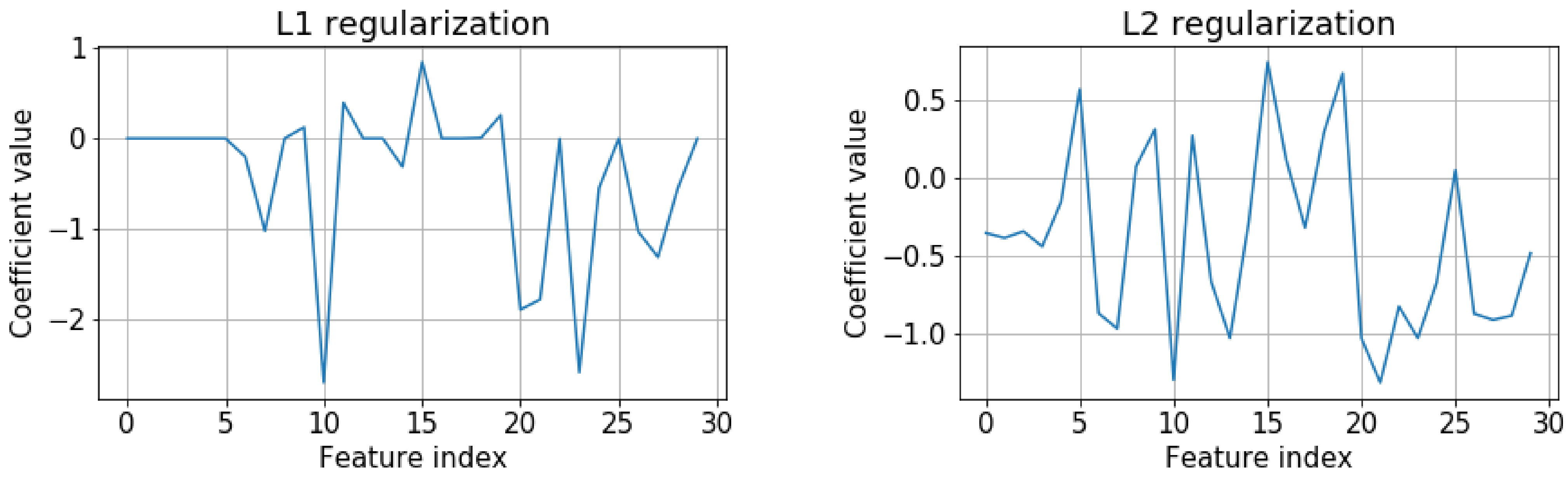
# L1 vs. L2 regularization

- 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(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

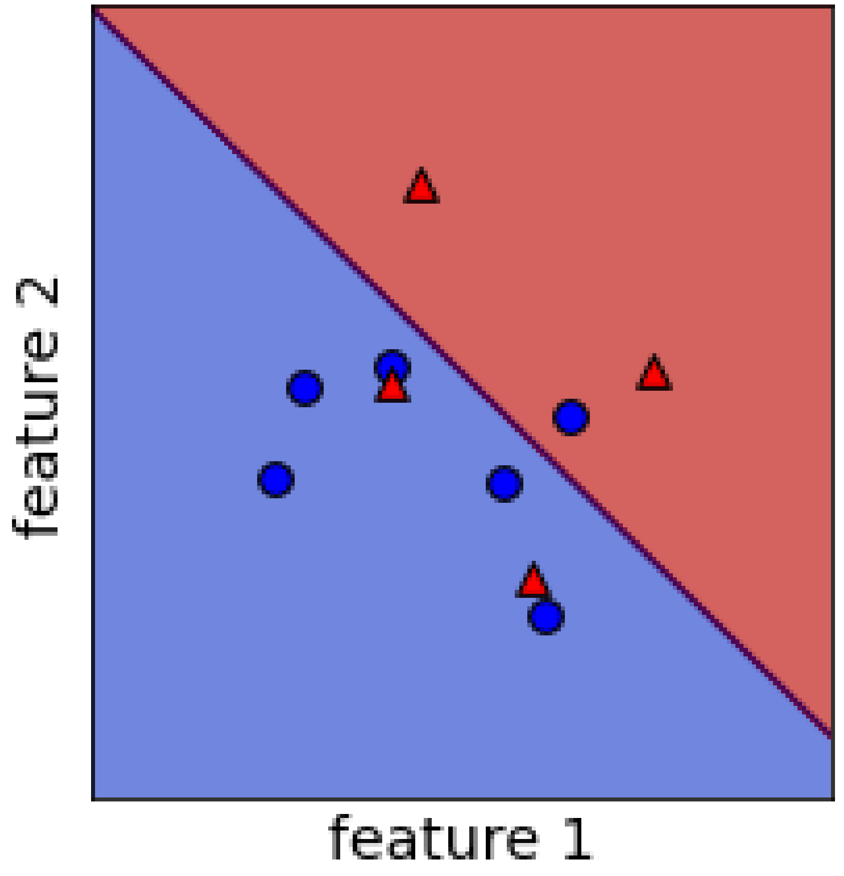
# Logistic regression and probabilities

LINEAR CLASSIFIERS IN PYTHON

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

# Logistic regression probabilities

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



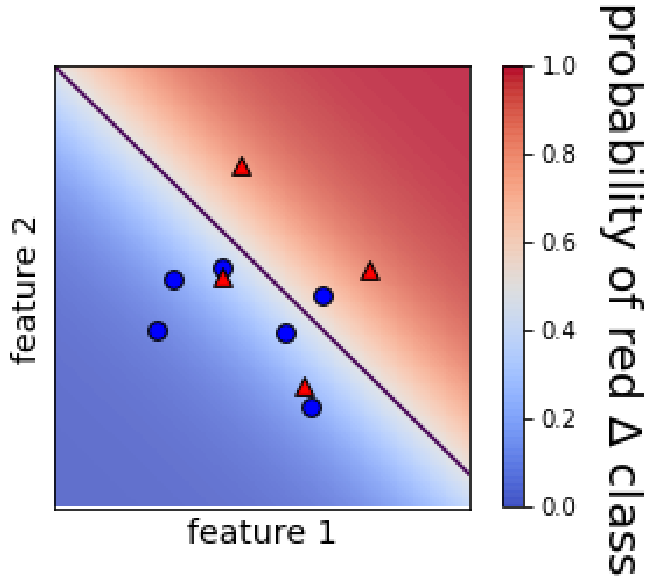

- model coefficients:

`[[1.55 1.57]]`

- model intercept: `[-0.64]`

# Logistic regression probabilities

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




- model coefficients:

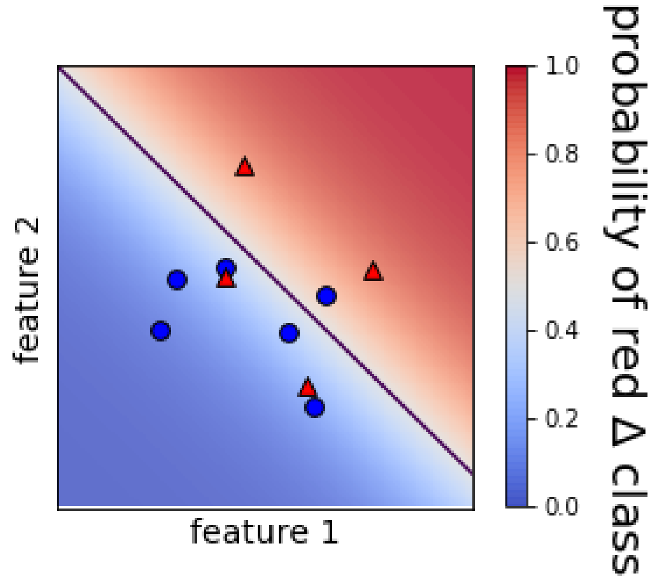
```
[[1.55 1.57]]
```

- model intercept:

```
[-0.64]
```

# Logistic regression probabilities

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



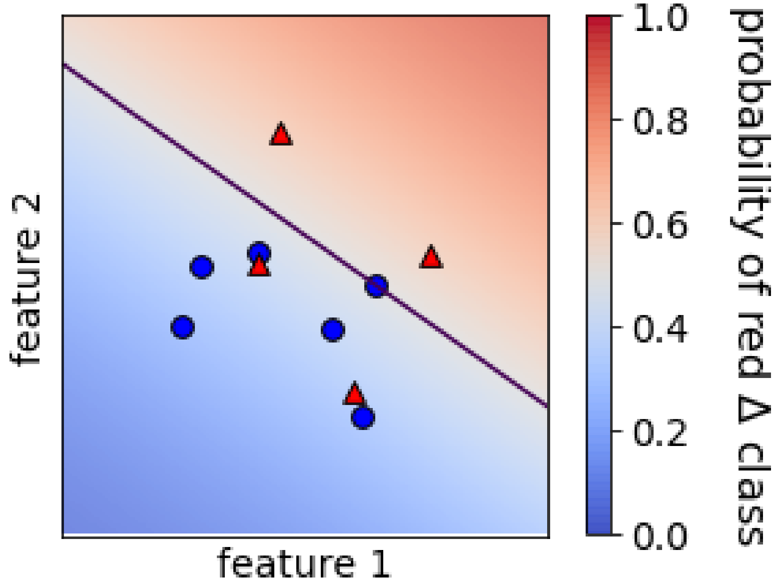

- model coefficients:

`[[1.55 1.57]]`

- model intercept:

`[-0.64]`

With regularization ($C = 1$):




- model coefficients:

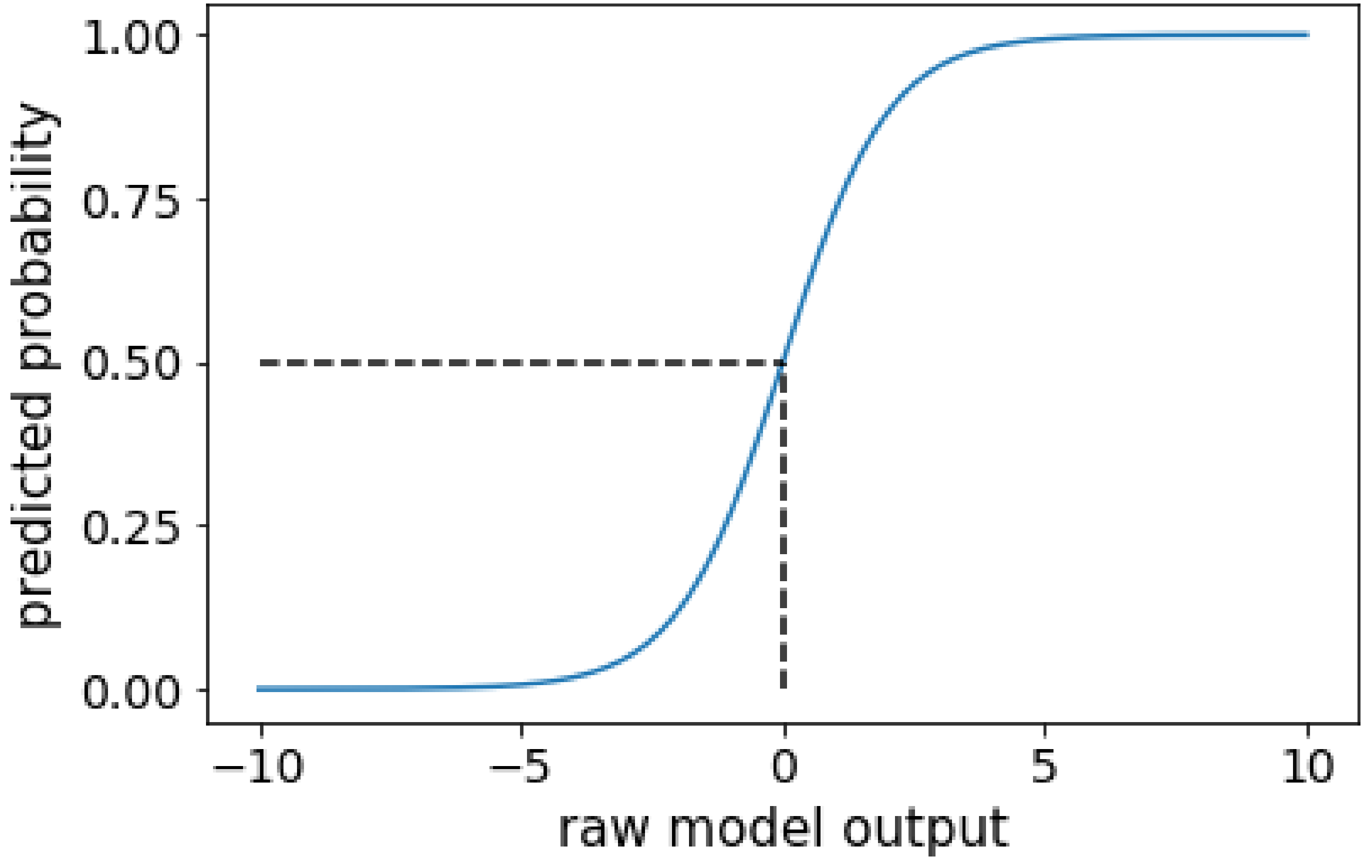
`[[0.45 0.64]]`

- 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.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

```
# one-vs-rest by default
lr_ovr = LogisticRegression()

lr_ovr.fit(X,y)

lr_ovr.coef_.shape
```

```
(3,13)
```

```
lr_ovr.intercept_.shape
```

```
(3,)
```

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

lr_mn.coef_.shape
```

```
(3,13)
```

```
lr_mn.intercept_.shape
```

```
(3,)
```

# Let's practice!

LINEAR CLASSIFIERS IN PYTHON

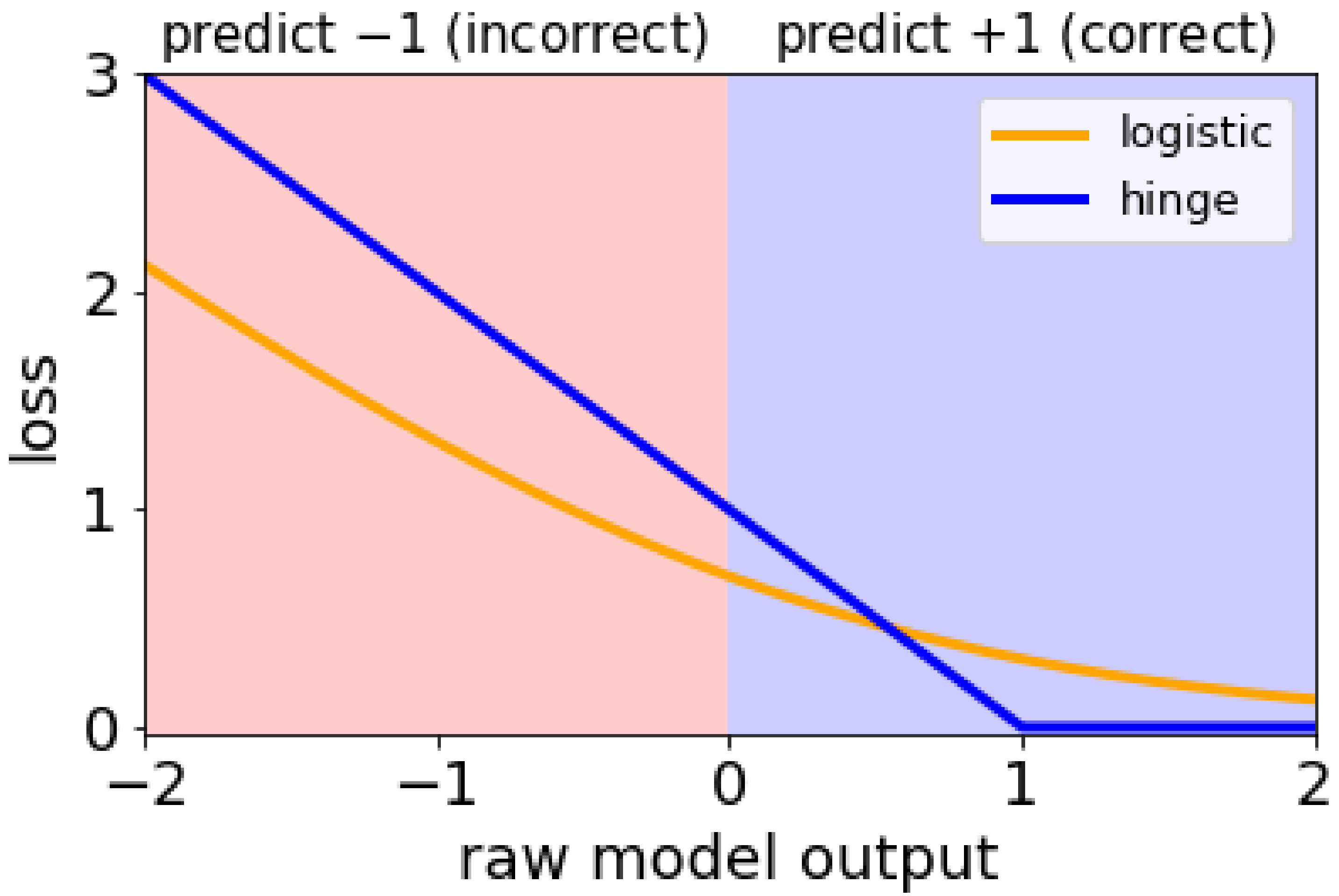
# Support Vectors

LINEAR CLASSIFIERS IN PYTHON

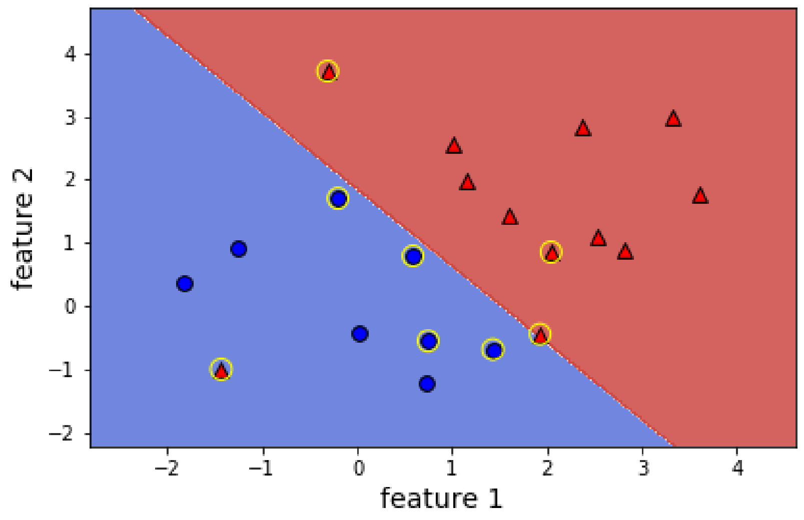
**Michael (Mike) Gelbart**
Instructor, The University of British Columbia

# What is an SVM?

- Linear classifiers (so far)
- Trained using the hinge loss and L2 regularization

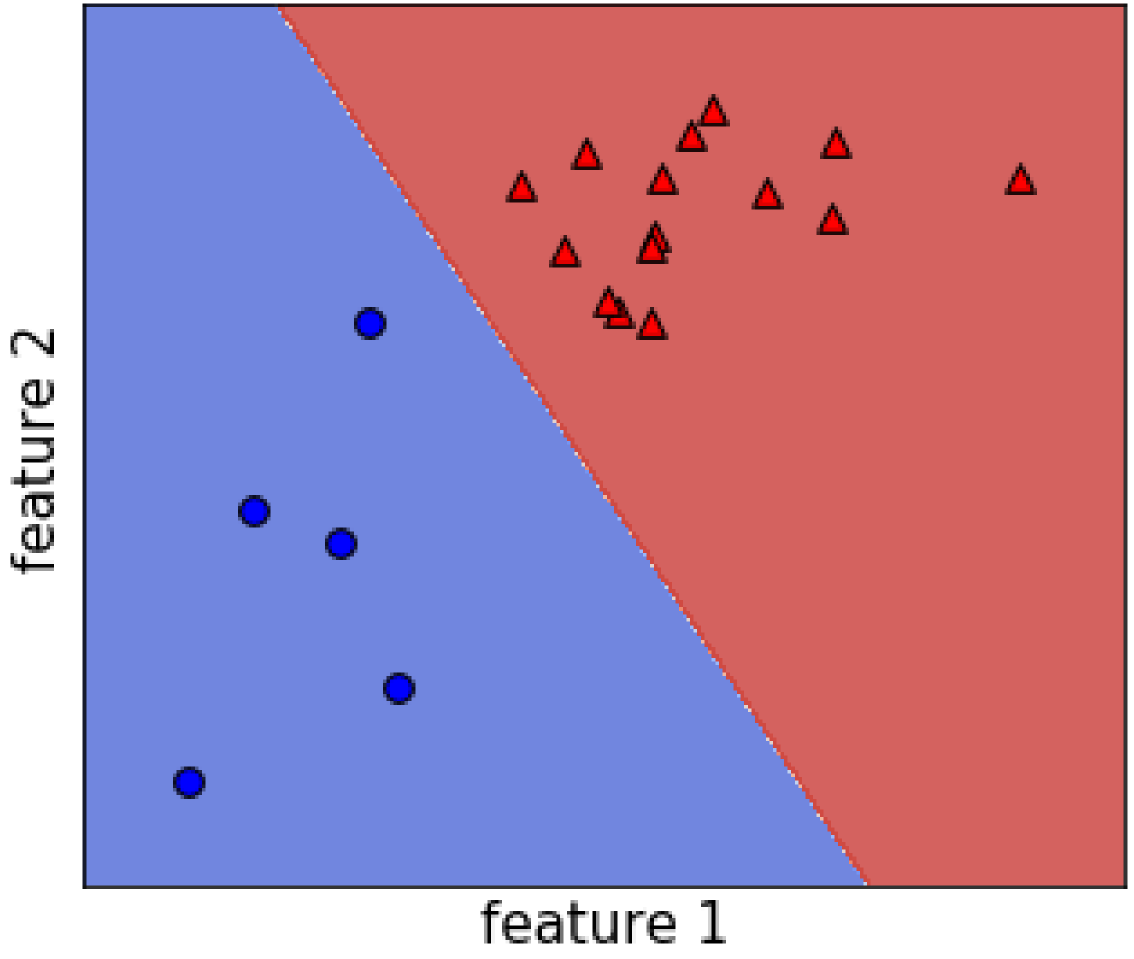


- Support vector: a training example **not** in the flat part of the loss diagram
- Support vector: an example that is incorrectly classified **or** close to the boundary
- If an example is not a support vector, removing it has no effect on the model
- Having a small number of support vectors makes kernel SVMs really fast



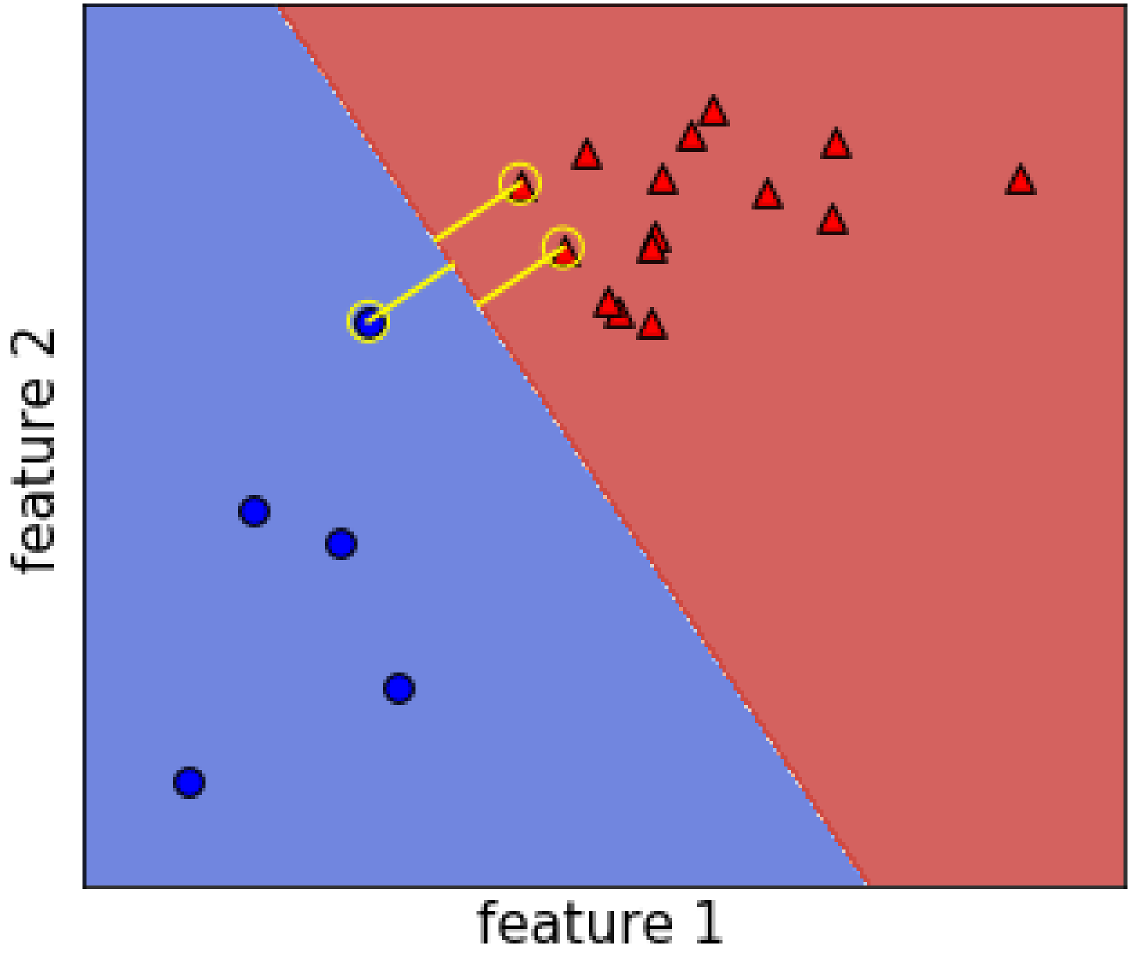
# Max-margin viewpoint

- The SVM maximizes the "margin" for linearly separable datasets
- Margin: distance from the boundary to the closest points



# Max-margin viewpoint

- The SVM maximizes the "margin" for linearly separable datasets
- Margin: distance from the boundary to the closest points



# Let's practice!

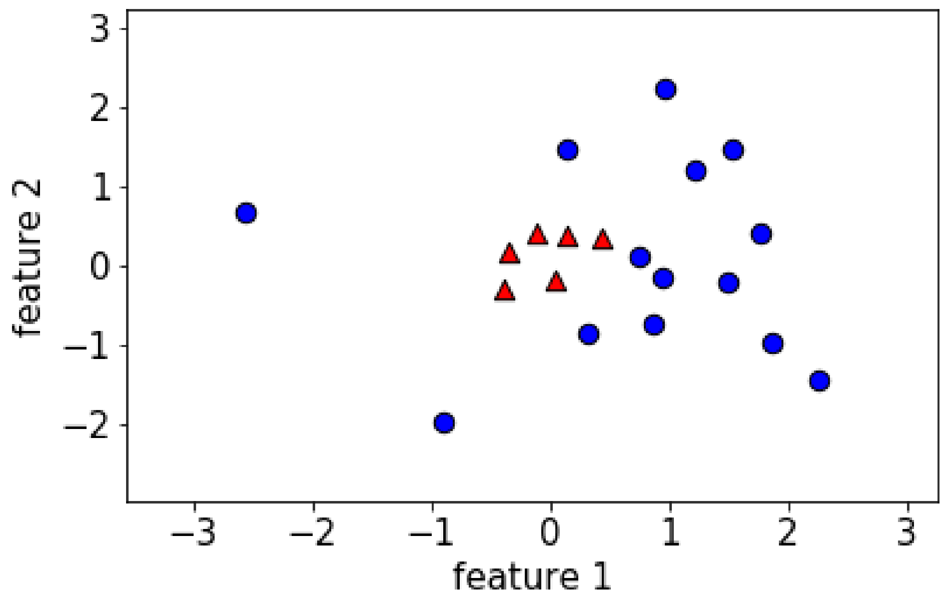
# Kernel SVMs

LINEAR CLASSIFIERS IN PYTHON

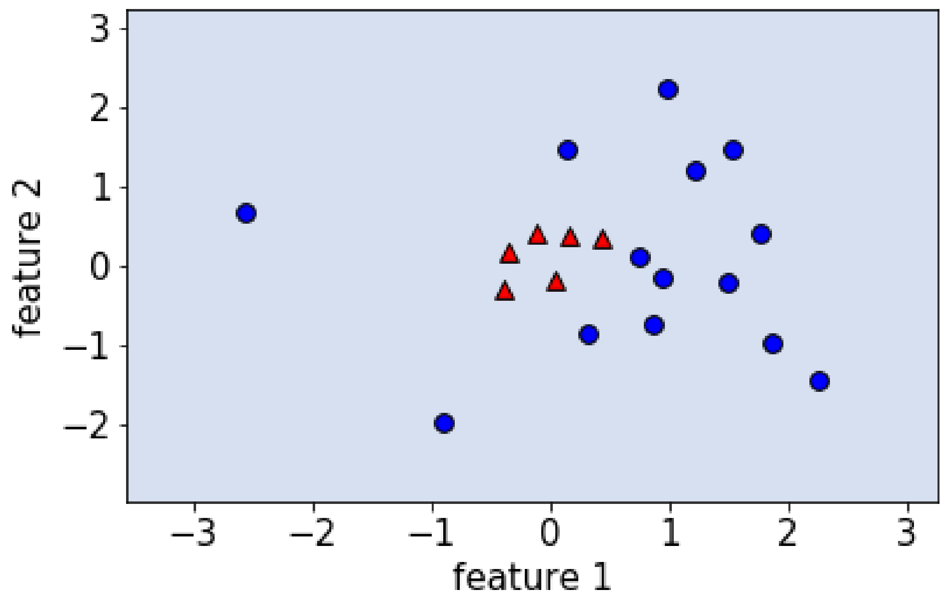
**Michael (Mike) Gelbart**

Instructor, The University of British Columbia

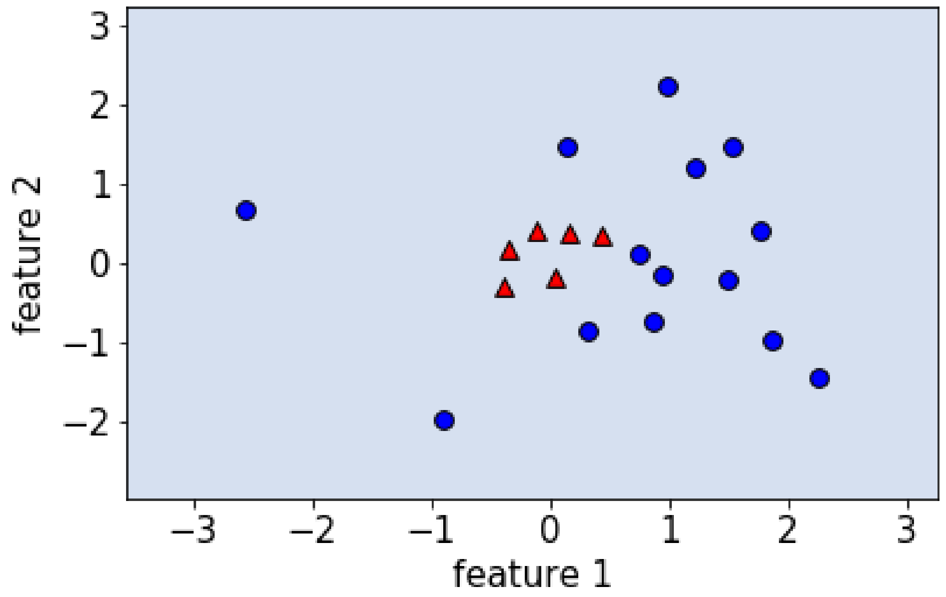
# Transforming your features

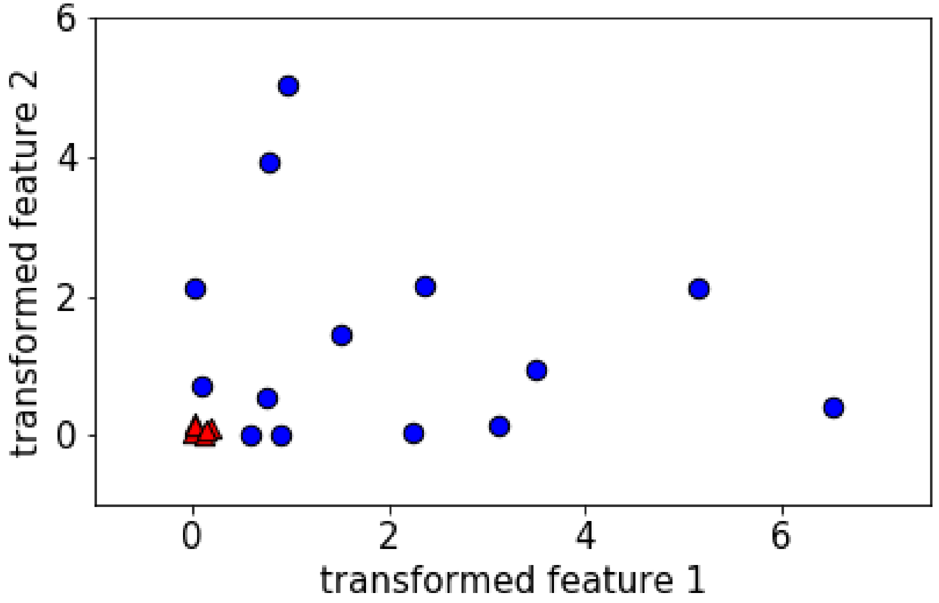


# Transforming your features



# Transforming your features

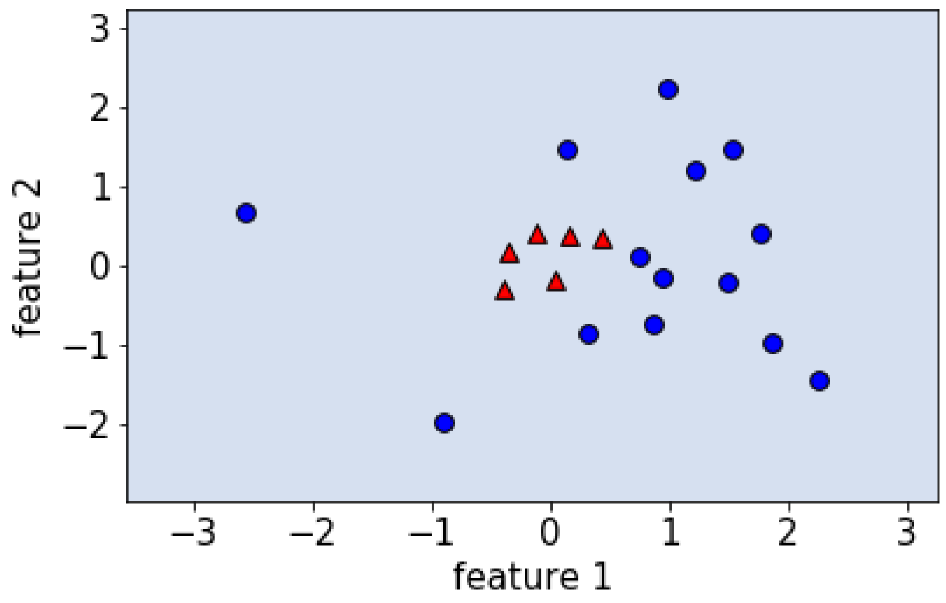


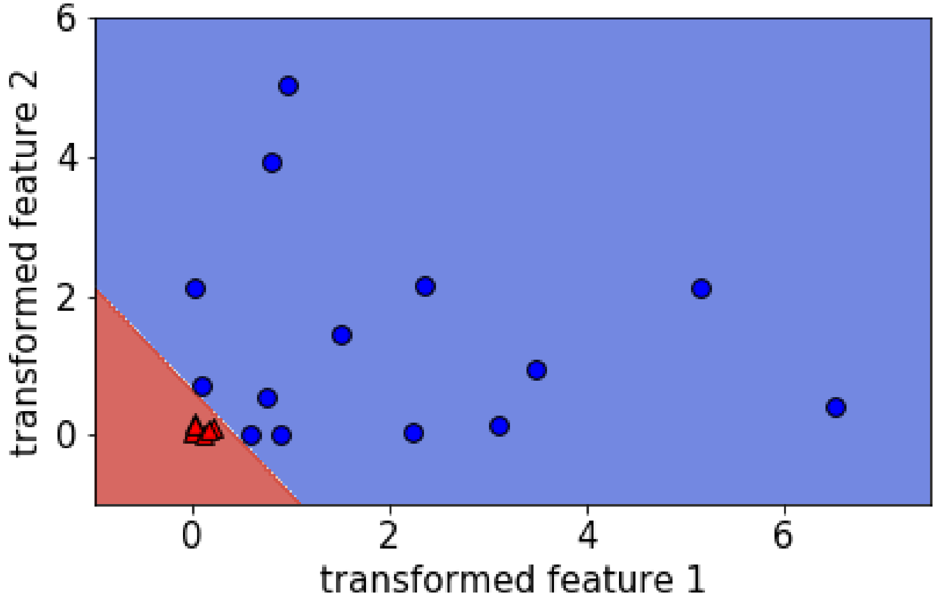




$$\text{transformed feature} =$$

$$(\text{original feature})^2$$

# Transforming your features

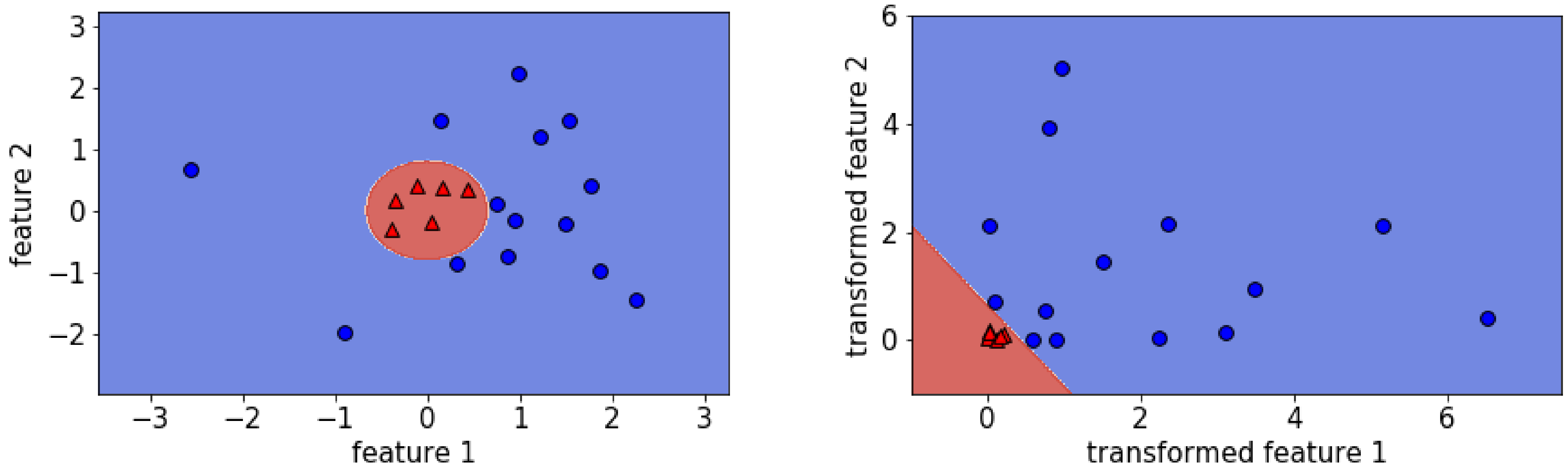



$$\text{transformed feature} =$$

$$(\text{original feature})^2$$

# Transforming your features

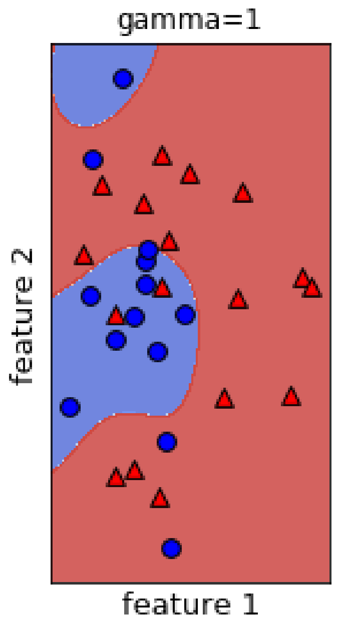



$$\text{transformed feature} =$$

$$(\text{original feature})^2$$

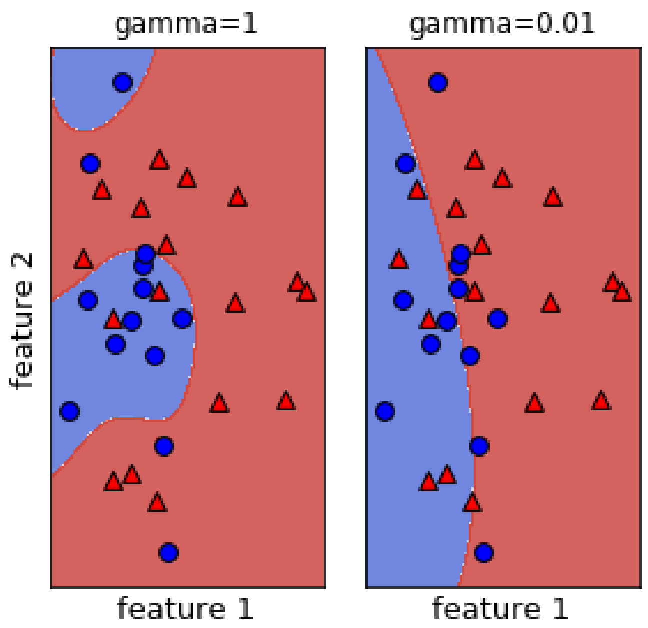
# Kernel SVMs

```
from sklearn.svm import SVC

svm = SVC(gamma=1)    # default is kernel="rbf"
```



# Kernel SVMs

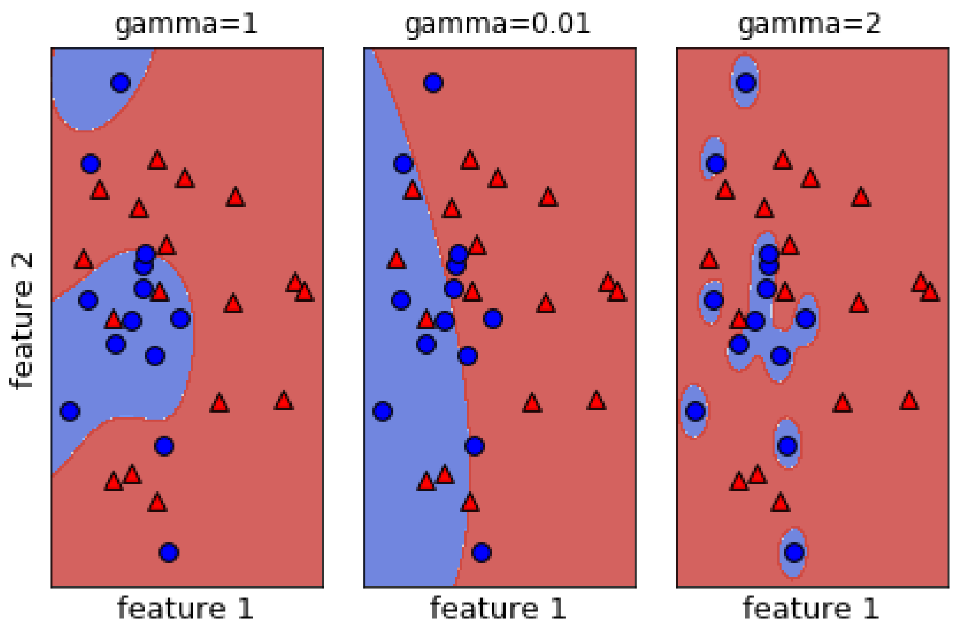
```
from sklearn.svm import SVC

svm = SVC(gamma=0.01) # default is kernel="rbf"
```




- smaller `gamma` leads to smoother boundaries

# Kernel SVMs

```
from sklearn.svm import SVC

svm = SVC(gamma=2)    # default is kernel="rbf"
```




- larger `gamma` leads to more complex boundaries

# Let's practice!

LINEAR CLASSIFIERS IN PYTHON

# Comparing logistic regression and SVM

LINEAR CLASSIFIERS IN PYTHON

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

**Logistic regression:**

- Is a linear classifier
- Can use with kernels, but slow
- Outputs meaningful probabilities
- Can be extended to multi-class
- All data points affect fit
- L2 or L1 regularization

**Support vector machine (SVM):**

- Is a linear classifier
- Can use with kernels, and fast
- Does not naturally output probabilities
- Can be extended to multi-class
- Only "support vectors" affect fit
- Conventionally just L2 regularization

# Use in scikit-learn

Logistic regression in sklearn:

- `linear_model.LogisticRegression`

Key hyperparameters in sklearn:

- `C` (inverse regularization strength)
- `penalty` (type of regularization)
- `multi_class` (type of multi-class)

SVM in sklearn:

- `svm.LinearSVC` and `svm.SVC`

# Use in scikit-learn (cont.)

Key hyperparameters in sklearn:

- `C` (inverse regularization strength)
- `kernel` (type of kernel)
- `gamma` (inverse RBF smoothness)

# SGDClassifier

`SGDClassifier` : scales well to large datasets

```
from sklearn.linear_model import SGDClassifier

logreg = SGDClassifier(loss='log')

linsvm = SGDClassifier(loss='hinge')
```

- `SGDClassifier` hyperparameter `alpha` is like `1/C`

# Let's practice!

LINEAR CLASSIFIERS IN PYTHON

# Conclusion

LINEAR CLASSIFIERS IN PYTHON

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

# How does this course fit into data science?

- Data science
- $\rightarrow$ Machine learning
- $\rightarrow\rightarrow$ Supervised learning
- $\rightarrow\rightarrow\rightarrow$ Classification
- $\rightarrow\rightarrow\rightarrow\rightarrow$ Linear classifiers (this course)

# Congratulations & thanks!

LINEAR CLASSIFIERS IN PYTHON