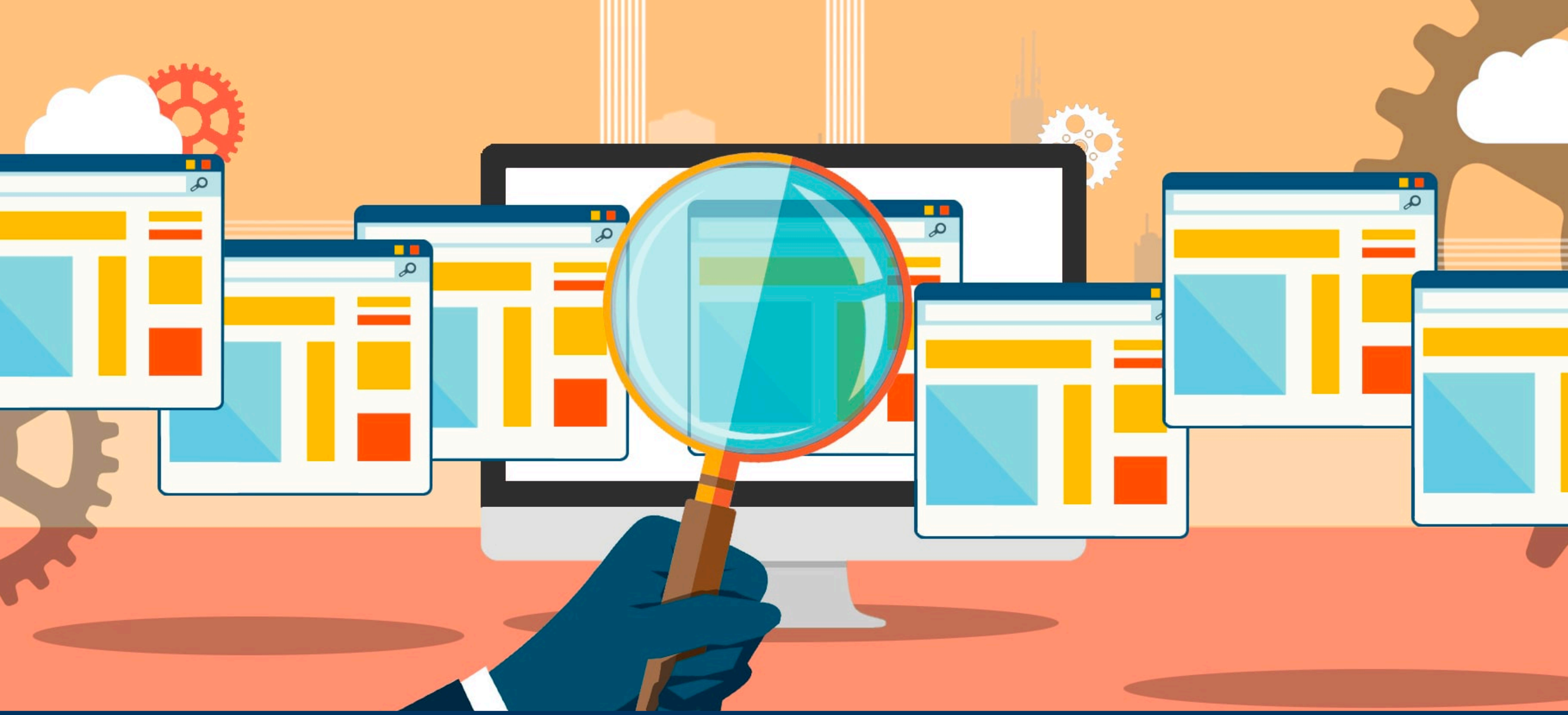


Supervised Learning, pt 2

INST414 - Data Science Techniques

This Module's Learning Objectives

Part 1

- Differentiate between classification and regression in supervised learning
- Describe how voting is used in for k-nearest neighbors classification
- Differentiate binary, multi-class, and multi-label classification
- Define overfitting and describe its impact on generalizability

This Module's Learning Objectives

Part 1

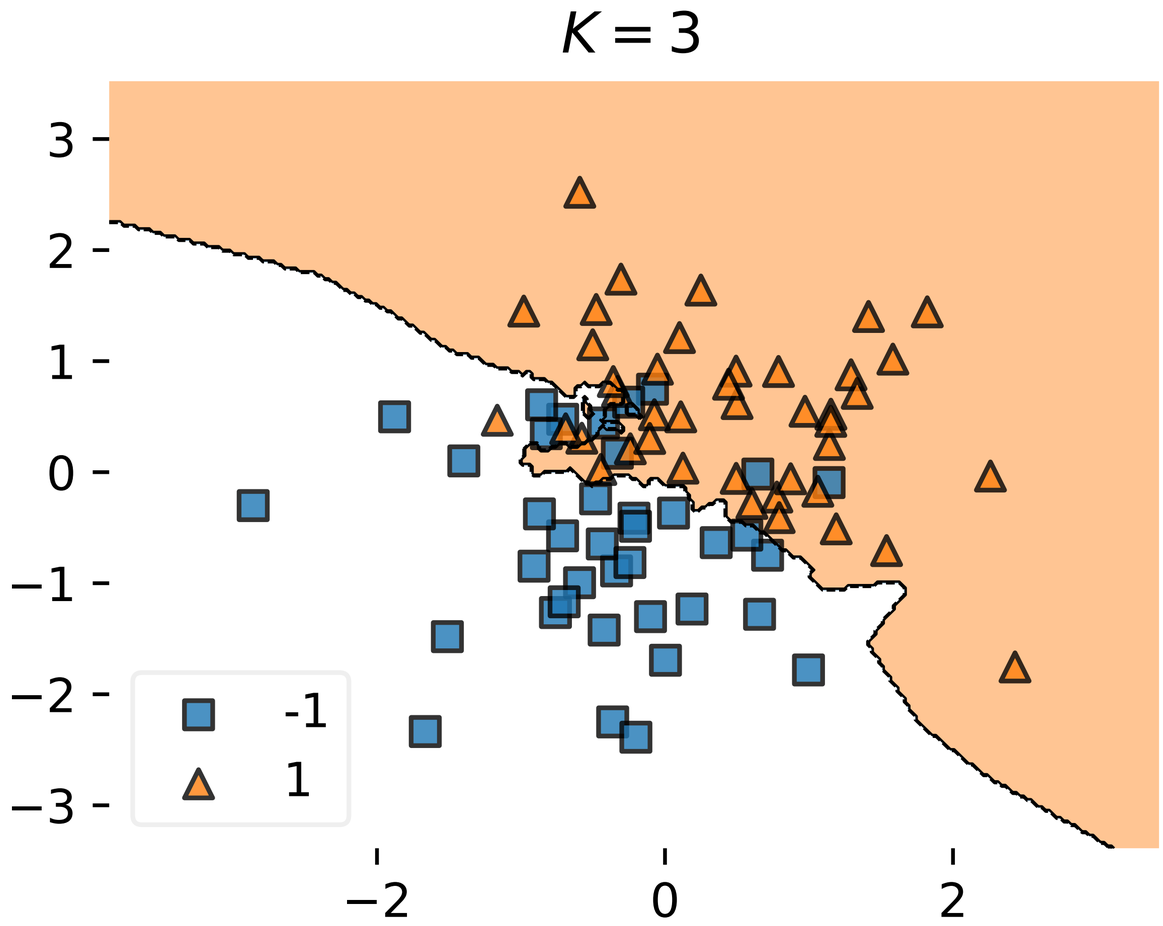
- Differentiate between classification and regression in supervised learning
- Describe how voting is used in for k-nearest neighbors classification
- Differentiate binary, multi-class, and multi-label classification
- Define overfitting and describe its impact on generalizability

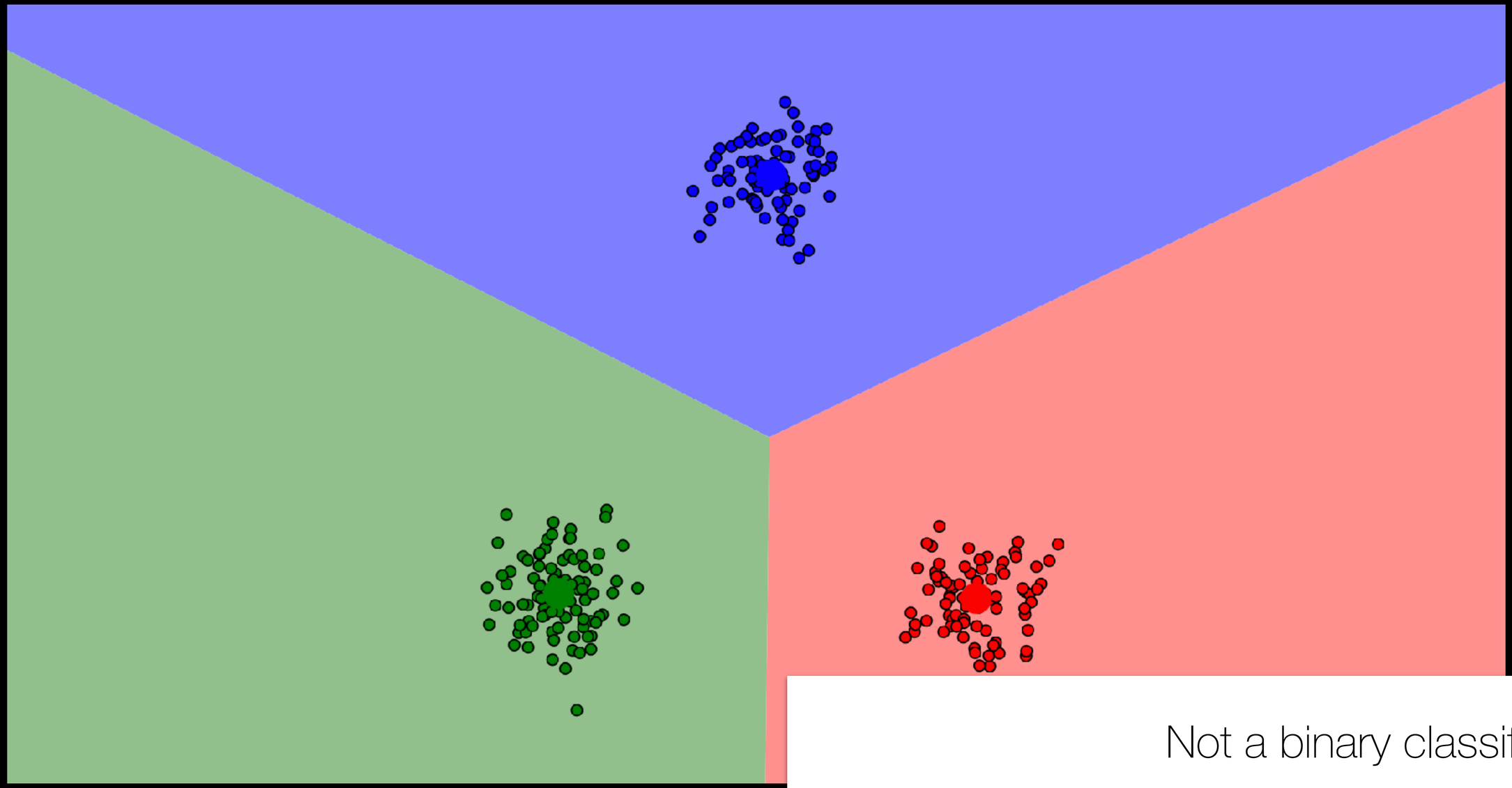
This Module's Learning Objectives

Part 1

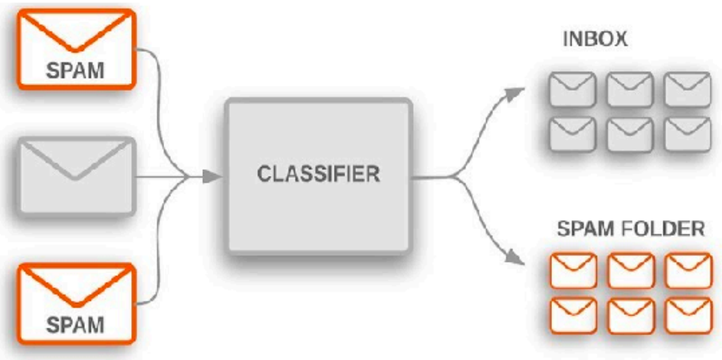
- Differentiate between classification and regression in supervised learning
- Describe how voting is used in for k-nearest neighbors classification
- Differentiate binary, multi-class, and multi-label classification
- Define overfitting and describe its impact on generalizability

Binary Classification

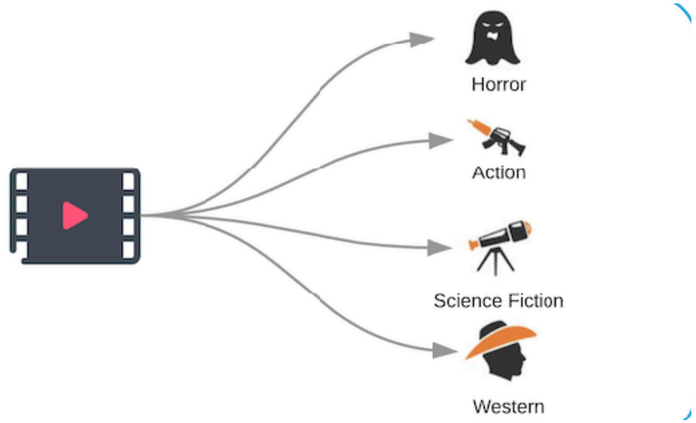




Not a binary classification



Binary Classification
e.g., Spam vs. Not-Spam



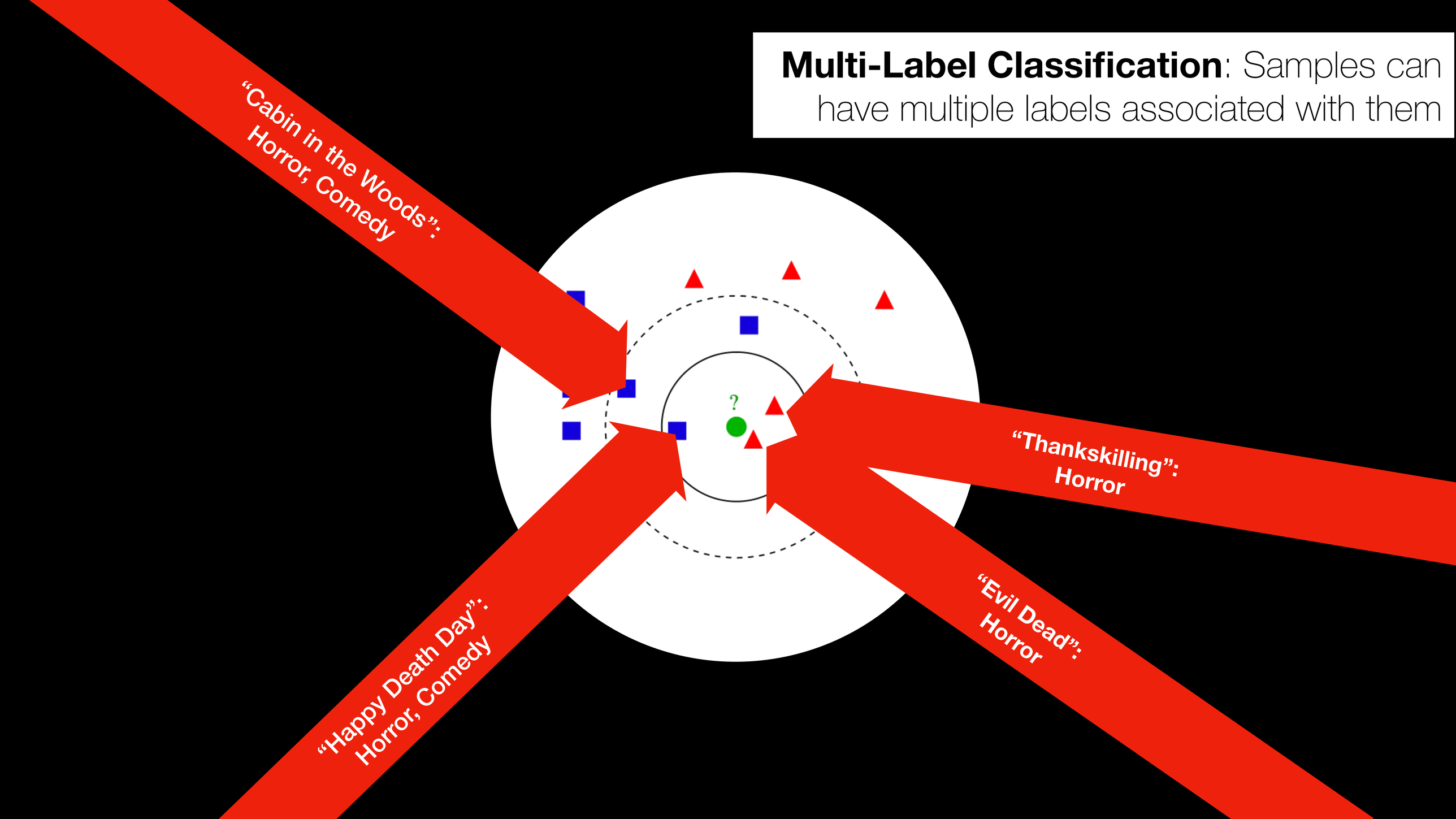
Multi-Class Classification
e.g., Movie Genre

But can’t a movie be in more than one genre?

Can assume only one genre for standard multi-class classification

BUT! We can handle multiple labels

Multi-Label Classification: Samples can have multiple labels associated with them



This Module's Learning Objectives

Part 1

- Differentiate between classification and regression in supervised learning
- Describe how voting is used in for k-nearest neighbors classification
- Differentiate binary, multi-class, and multi-label classification
- Define overfitting and describe its impact on generalizability

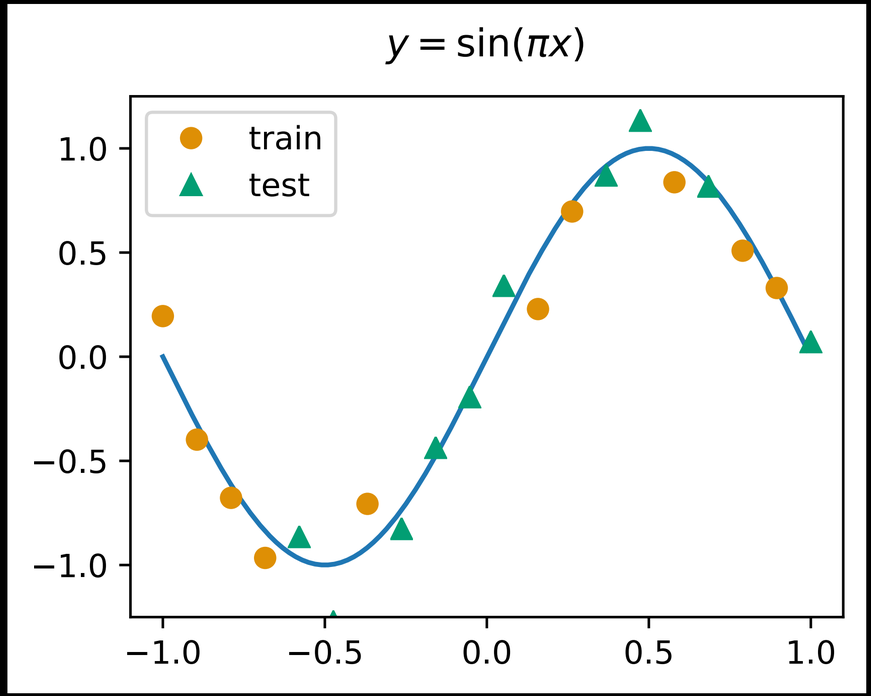
This Module's Learning Objectives

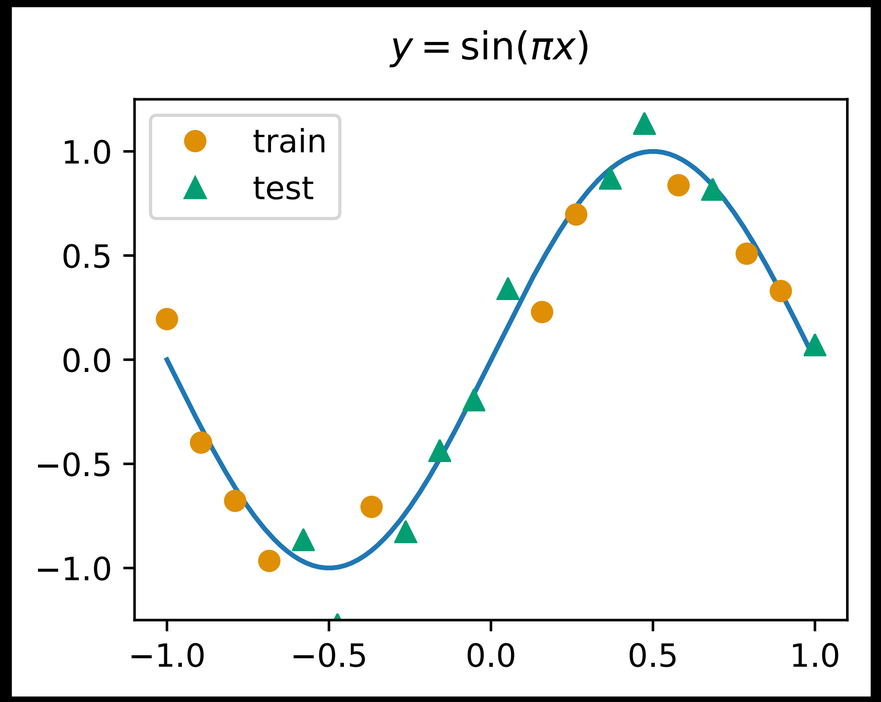
Part 1

- Differentiate between classification and regression in supervised learning
- Describe how voting is used in for k-nearest neighbors classification
- Differentiate binary, multi-class, and multi-label classification
- Define overfitting and describe its impact on generalizability

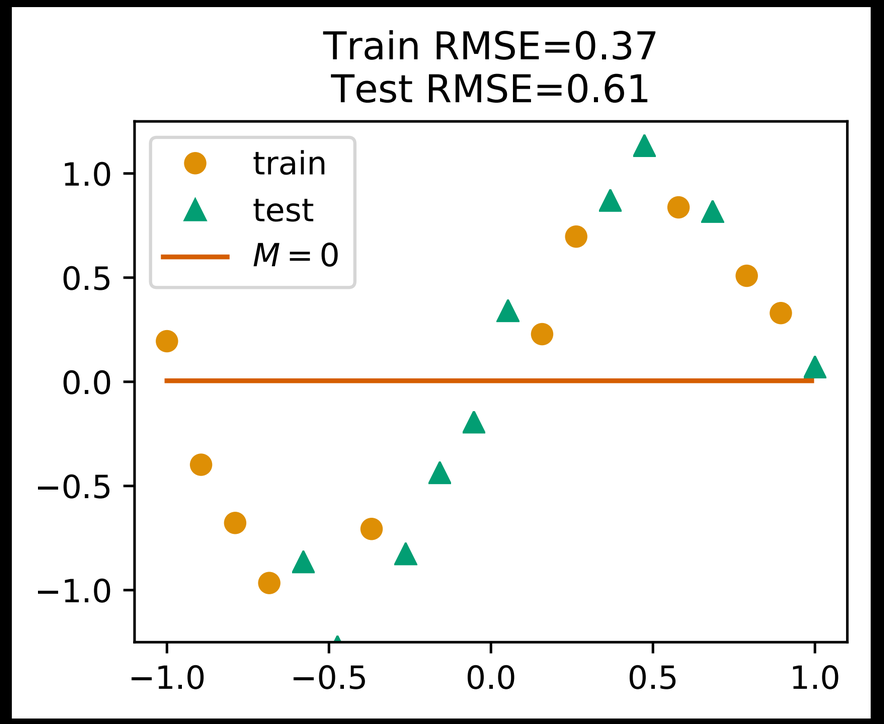
An Example: Fitting Curve

- Suppose we have 20 observations randomly sampled (with noise) from a sin curve
- Fit with kNN regression
  - What is kNN regression?





How does this decision boundary change as we increase/decrease k?

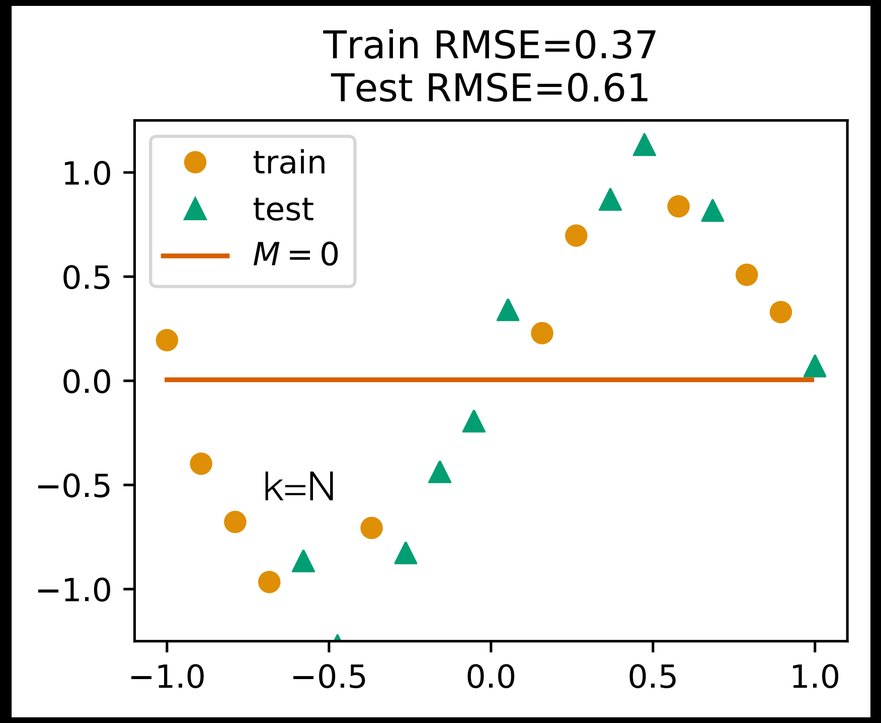


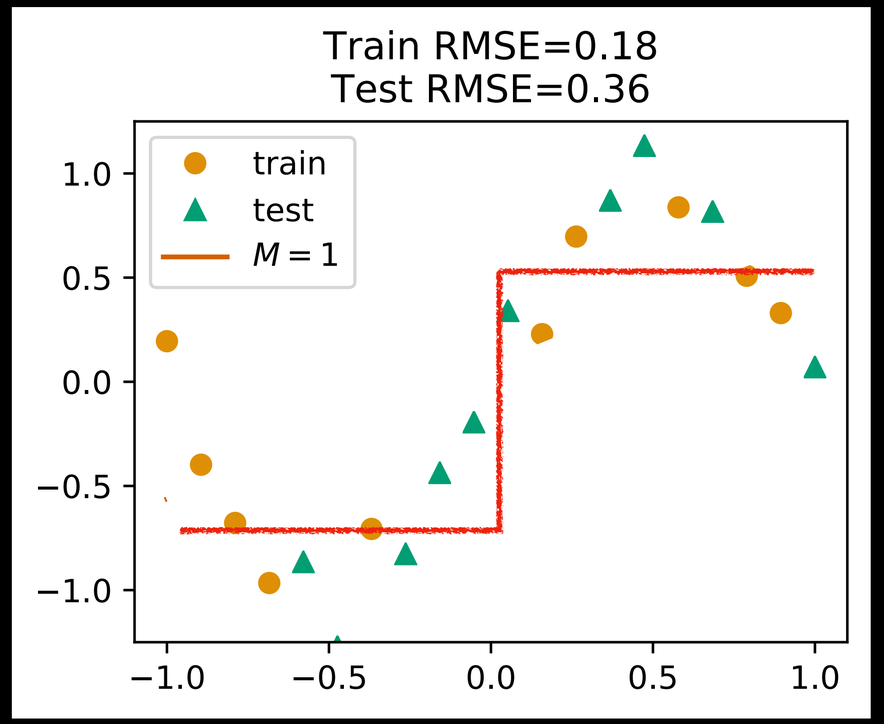
We have n=20 points. What happens if k=20?

Predicts the average over all 20 samples

Fitting Models with Different Complexity

- The model is “underfit”
  - Cannot capture the patterns in the training data
  - May generalize well if data is really imbalanced



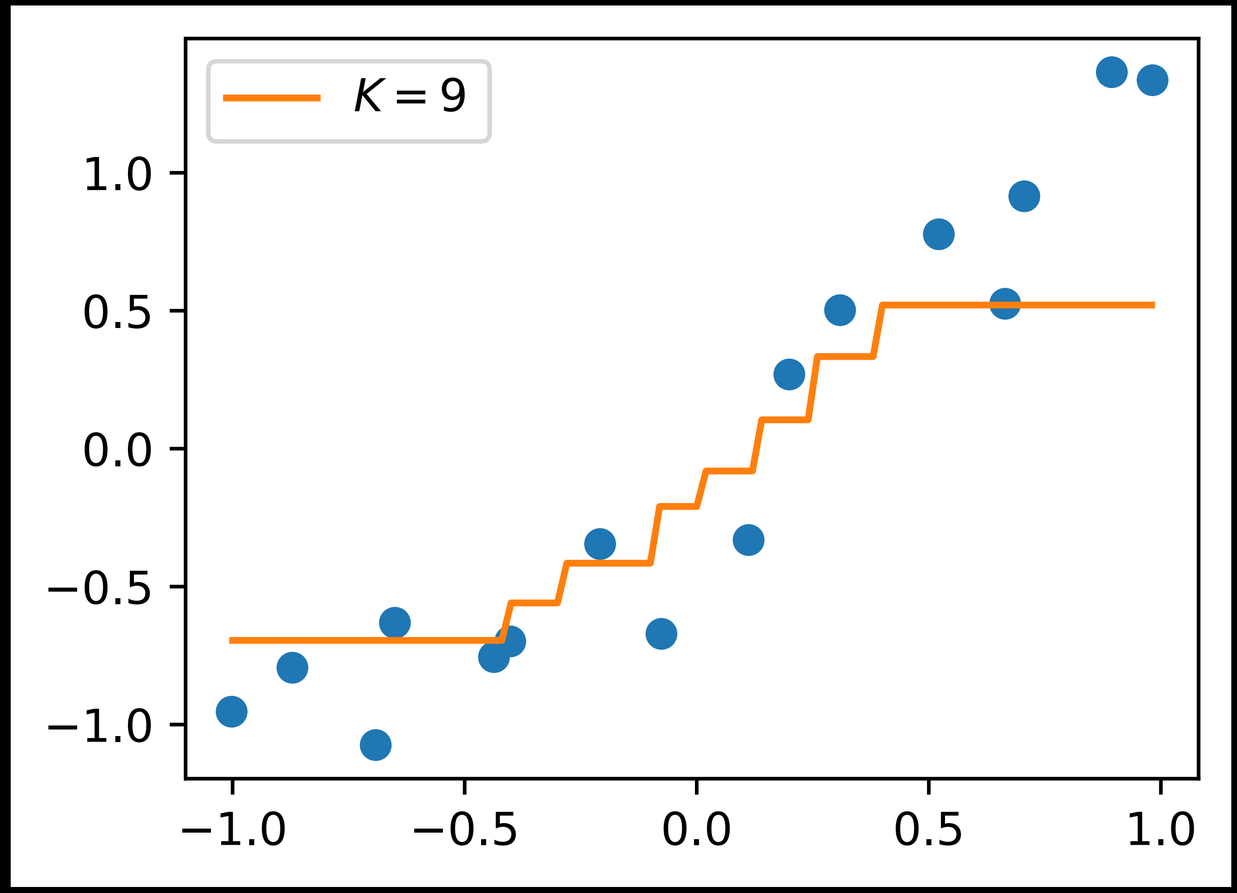


We have n=20 points. What happens if k=n/2, 10?

Is this a better model than k=n?

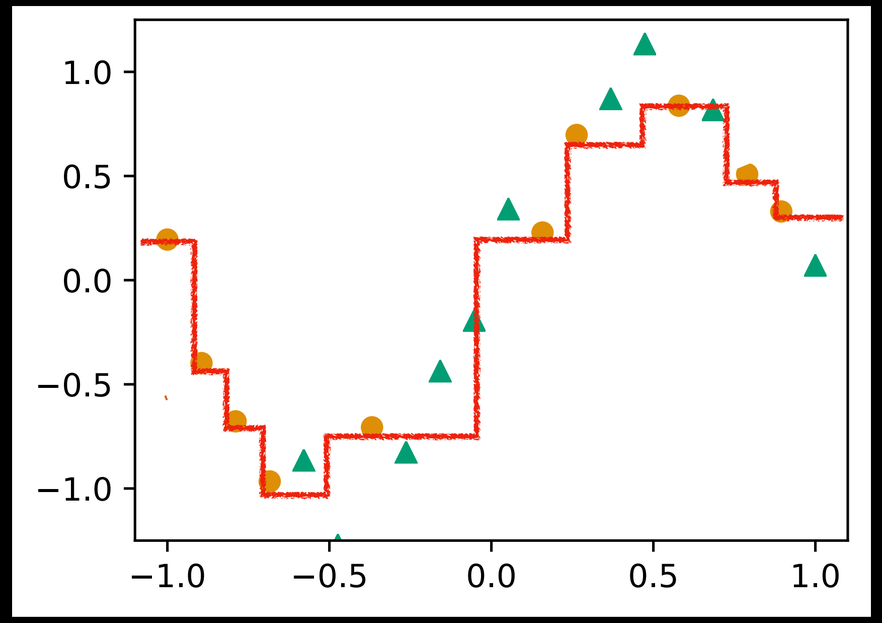
A Better-fit Model

- The model captures the general trend among the training data
- The model “ignores” little variations
  - ignore = not complex enough to capture



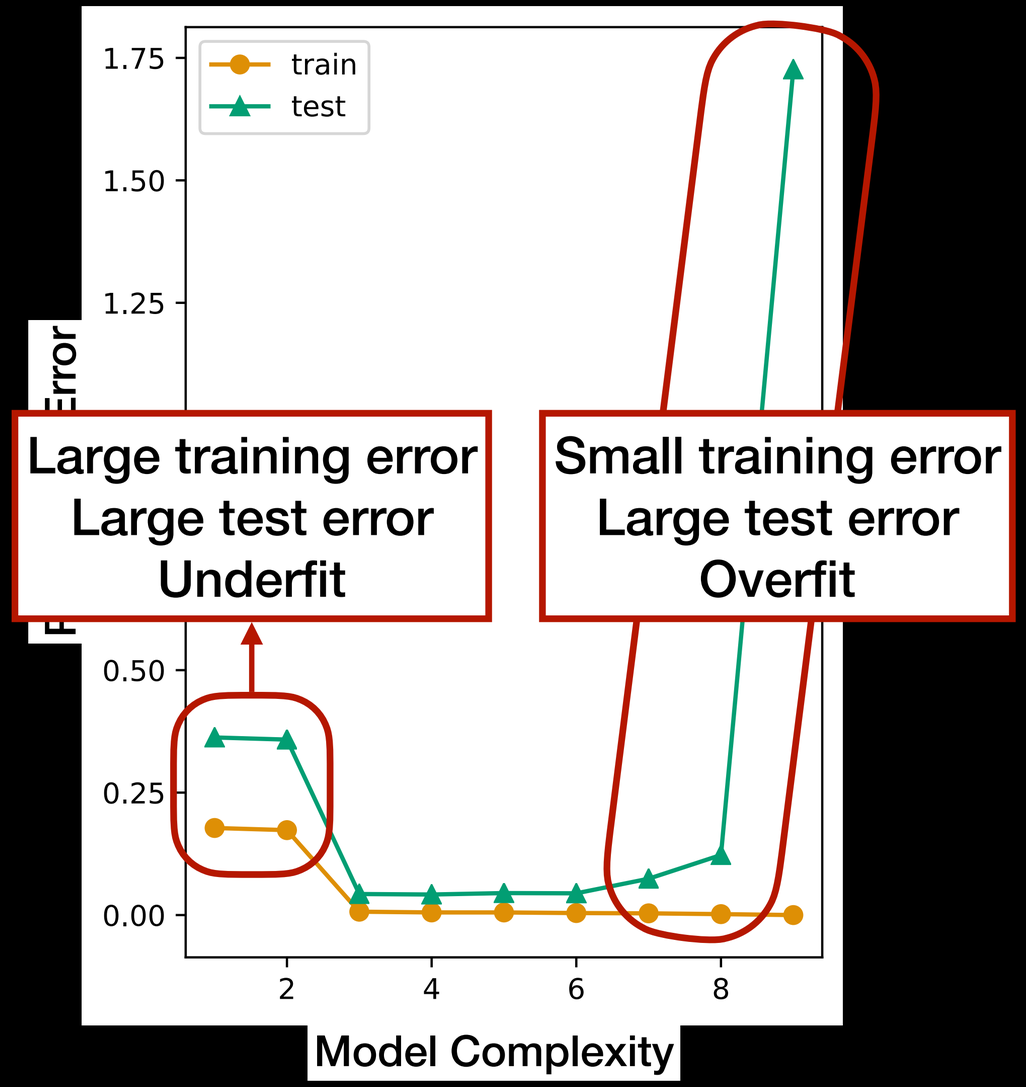
An Overfit Model

- The model is complex enough to capture both the pattern and the noise
- Fails to generalize to unseen data
- The model has been "*overfit*"

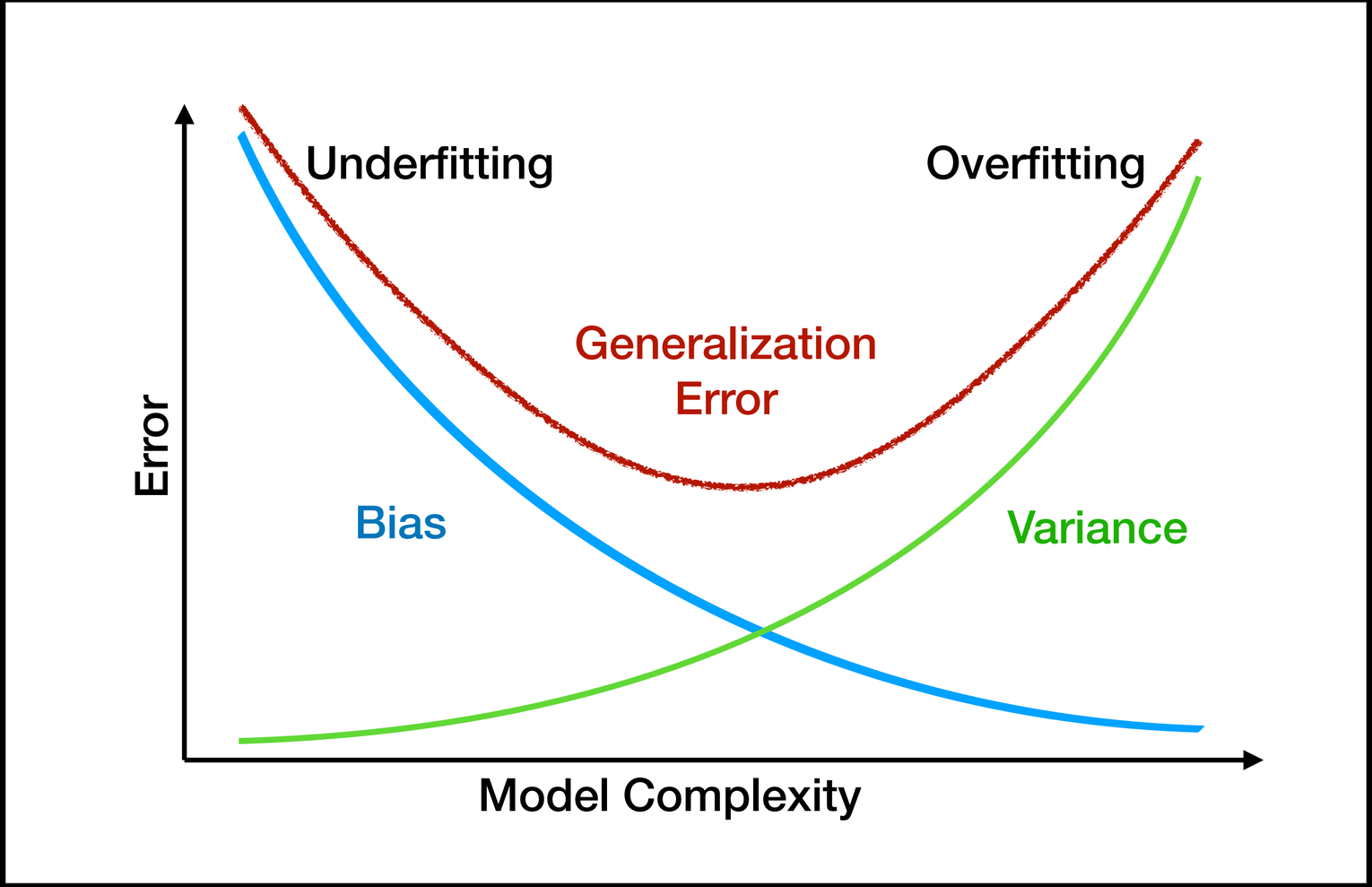


Model Performance on Training and Test Data

- As complexity increases:
  - Training error keeps decreasing
  - Test error decreases then increases



A Bit of Learning Theory



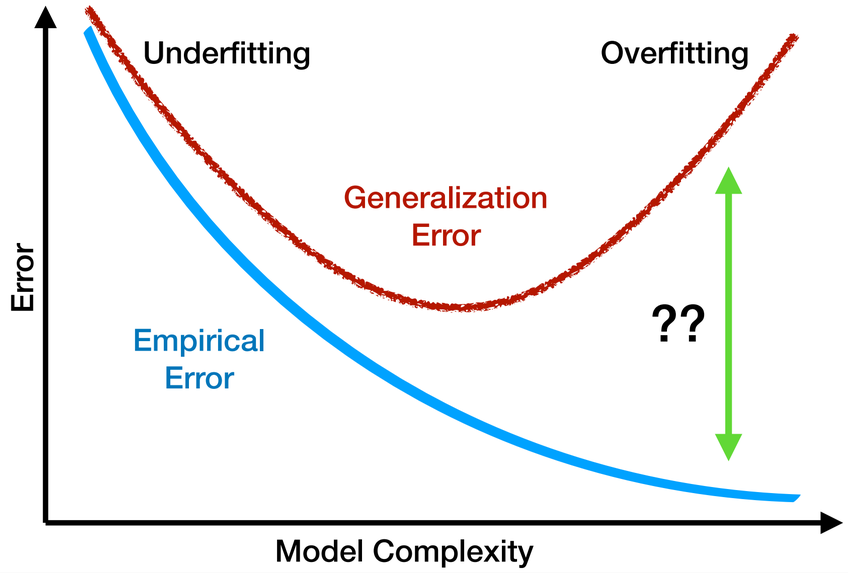
The Bias-Variance Tradeoff

We don’t know due to unseen data
Positively correlated with model complexity
Leads to Overfitting

Generalization Error = Empirical Error + ??

The goal of ML is to minimize Generalization Error

Can be measured using training set (training error)



How to Reduce Overfitting

- More training examples
- Reducing the complexity of the models
  - Large k for KNN models (but not too large)
  - “Regularization terms” to penalize complex models.
    - To discuss next time!

How do we reduce over-fitting?

More training data

Reduce model complexity (e.g., increase k in k-nn)

“Regularization” to penalize over-fitting

This Module's Learning Objectives

Part 1

- Differentiate between classification and regression in supervised learning
- Describe how voting is used in for k-nearest neighbors classification
- Differentiate binary, multi-class, and multi-label classification
- Define overfitting and describe its impact on generalizability

This Module's Learning Objectives

Part 2

Define regularization and its use of improving generalizability

Use linear regression models to predict scores for data elements

Explain how linear regression can be adapted for classification

Extract the class-label probabilities for model outputs

This Module's Learning Objectives

Part 2

Define regularization and its use of improving generalizability

Use linear regression models to predict scores for data elements

Explain how linear regression can be adapted for classification

Extract the class-label probabilities for model outputs

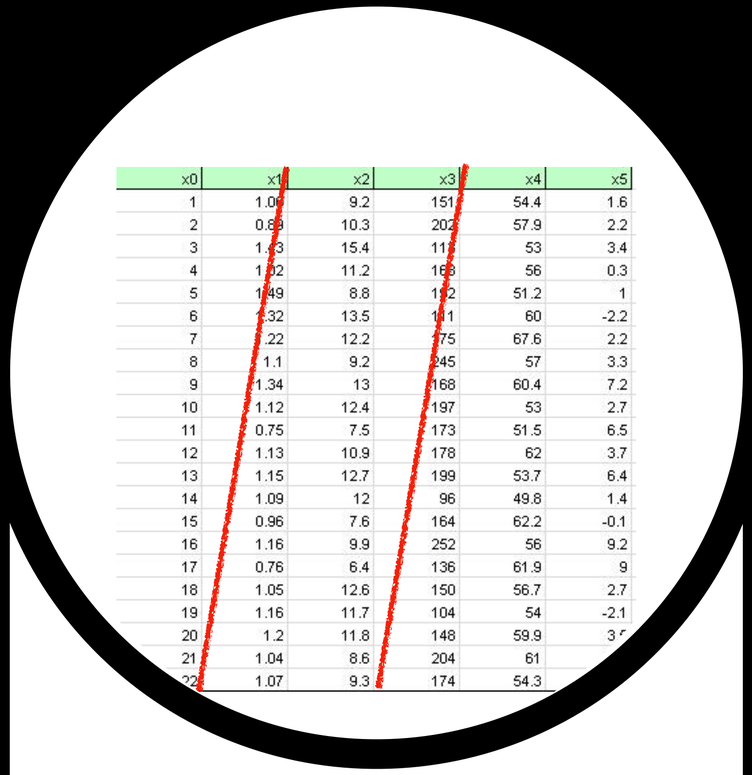
How to Reduce Overfitting

- More training examples
- Reducing the complexity of the models
  - Large k for KNN models (but not too large)
  - “Regularization terms” to penalize complex models

What does “regularization” mean?

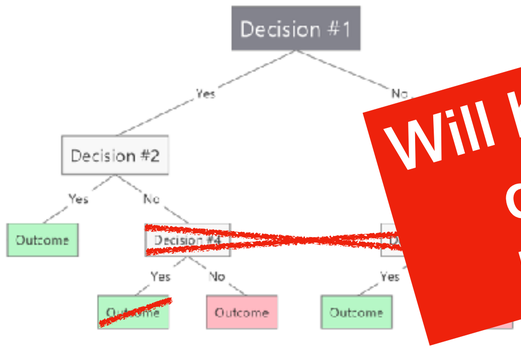
Regularization: Methods for penalizing model complexity

How might you penalize model complexity?

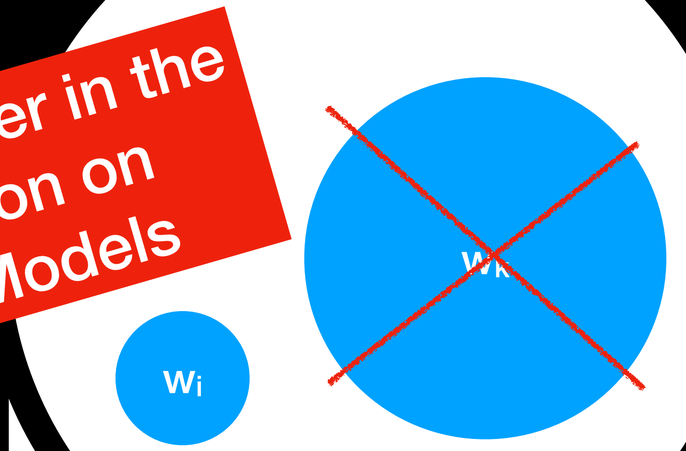


x0	x1	x2	x3	x4	x5
1	1.06	9.2	151	54.4	1.6
2	0.89	10.3	202	57.9	2.2
3	1.43	15.4	113	53	3.4
4	1.02	11.2	168	56	0.3
5	1.49	8.8	192	51.2	1
6	1.32	13.5	111	60	-2.2
7	1.22	12.2	175	67.6	2.2
8	1.1	9.2	245	57	3.3
9	1.34	13	168	60.4	7.2
10	1.12	12.4	197	53	2.7
11	0.75	7.5	173	51.5	6.5
12	1.13	10.9	178	62	3.7
13	1.15	12.7	199	53.7	6.4
14	1.09	12	96	49.8	1.4
15	0.96	7.6	164	62.2	-0.1
16	1.16	9.9	252	56	9.2
17	0.76	6.4	136	61.9	9
18	1.05	12.6	150	56.7	2.7
19	1.16	11.7	104	54	-2.1
20	1.2	11.8	148	59.9	3.5
21	1.04	8.6	204	61	
22	1.07	9.3	174	54.3	

You use fewer dimensions/features

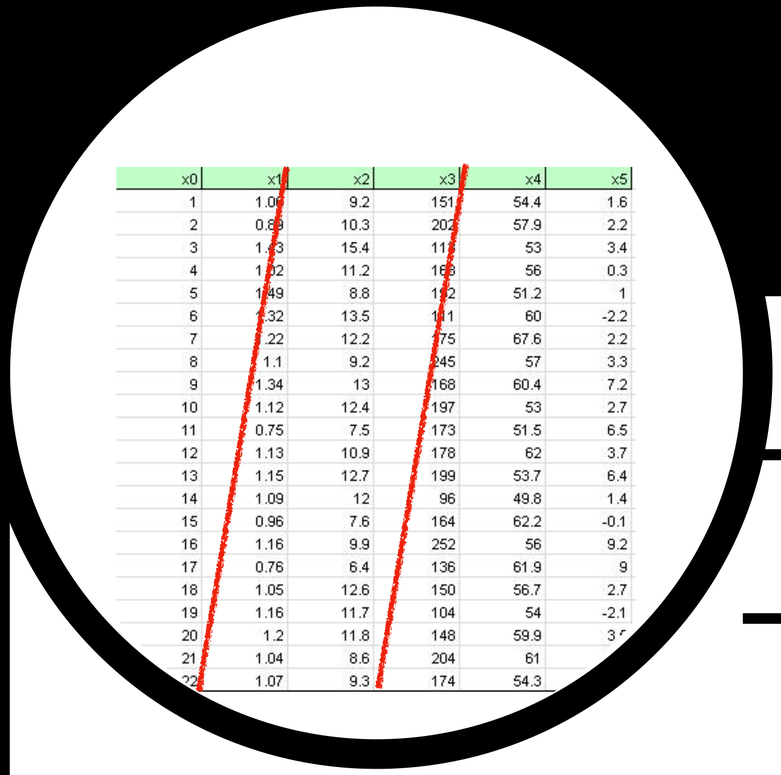


Model enforces sparsity (mostly 0s)



Model forces feature weights to be small

Deciding which features to use is your decision



x0	x1	x2	x3	x4	x5
1	1.06	9.2	151	54.4	1.6
2	0.89	10.3	202	57.9	2.2
3	1.43	15.4	113	53	3.4
4	1.02	11.2	168	56	0.3
5	1.49	8.8	192	51.2	1
6	1.32	13.5	111	60	-2.2
7	1.22	12.2	175	67.6	2.2
8	1.1	9.2	245	57	3.3
9	1.34	13	168	60.4	7.2
10	1.12	12.4	197	53	2.7
11	0.75	7.5	173	51.5	6.5
12	1.13	10.9	178	62	3.7
13	1.15	12.7	199	53.7	6.4
14	1.09	12	96	49.8	1.4
15	0.96	7.6	164	62.2	-0.1
16	1.16	9.9	252	56	9.2
17	0.76	6.4	136	61.9	9
18	1.05	12.6	150	56.7	2.7
19	1.16	11.7	104	54	-2.1
20	1.2	11.8	148	59.9	3.5
21	1.04	8.6	204	61	
22	1.07	9.3	174	54.3	

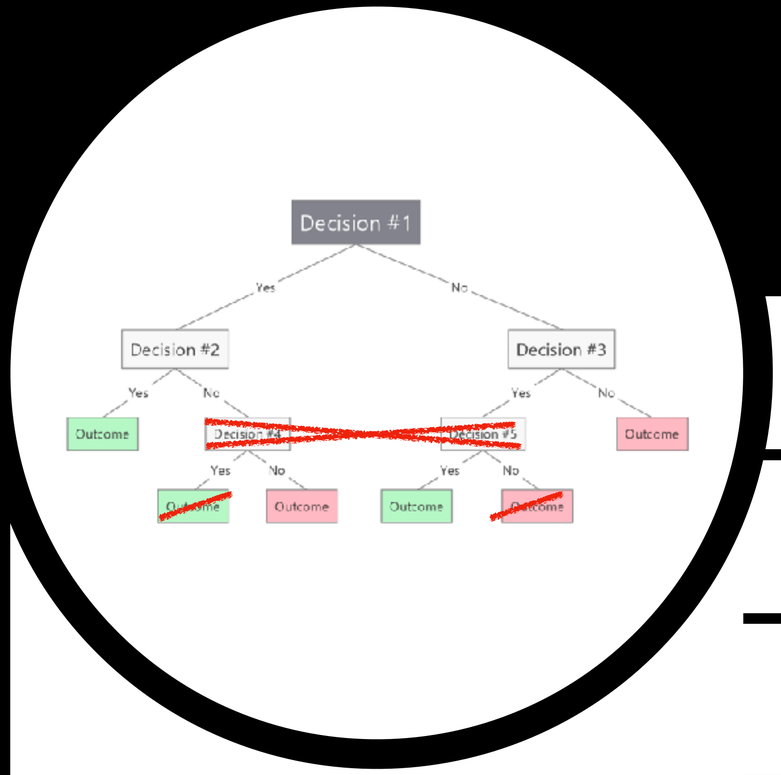
You use fewer dimensions/features

Also called “feature selection”

<u>You</u> decide which features should count

Fewer features —> distances in fewer dimensions

Deciding which features to use is part of the model-fitting process here



When building decision trees, how many features many?

Forcing fewer features == shallower trees

Shallower trees == simpler models

Model enforces sparsity (mostly 0s)

Regularization: Methods for penalizing model complexity

Reducing complexity reduces overfitting

Increases model interpretability

(Often) Decreases computational burden

This Module's Learning Objectives

Part 2

Define regularization and its use of improving generalizability

Use linear regression models to predict scores for data elements

Explain how linear regression can be adapted for classification

Extract the class-label probabilities for model outputs

This Module's Learning Objectives

Part 2

Define regularization and its use of improving generalizability

Use linear regression models to predict scores for data elements

Explain how linear regression can be adapted for classification

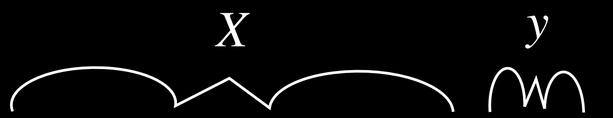
Extract the class-label probabilities for model outputs

Recall from last time…

x_i => i^{th} row in $\boldsymbol{X}$

x_{ij} => j^{th} feature of the i^{th} row in $\boldsymbol{X}$

$\boldsymbol{X}$ is an n x d matrix, where d is the number of features



mass	width	height	color_score	fruit_label
192	8.4	7.3	0.55	1
180	8.0	6.8	0.59	1
176	7.4	7.2	0.60	1
86	6.2	4.7	0.80	2
84	6.0	4.6	0.79	2
80	5.8	4.3	0.77	2
80	5.9	4.3	0.81	2
76	5.8	4.0	0.81	2
178	7.1	7.8	0.92	1
172	7.4	7.0	0.89	1
166	6.9	7.3	0.93	1
172	7.1	7.6	0.92	1
154	7.0	7.1	0.88	1
164	7.3	7.7	0.70	1
152	7.6	7.3	0.69	1
156	7.7	7.1	0.69	1
156	7.6	7.5	0.67	1
168	7.5	7.6	0.73	1
162	7.5	7.1	0.83	1
162	7.4	7.2	0.85	1

Linear Regression in simplest form

Linear regression is the process of finding these weights w_j such that…

Function of weights w times a row in X plus some constant value

$$y_i = wx_i + b | x_i \in X, y_i \in y$$

$$y_i = \sum_{j \in d} w_j x_{i,j} + b$$

Linear Regression in simplest form

Linear regression is the process of finding these weights w_j such that…

Function of weights w times a row in X plus some constant value

$$= wx_i + b | x_i \in X, y_i \in y$$

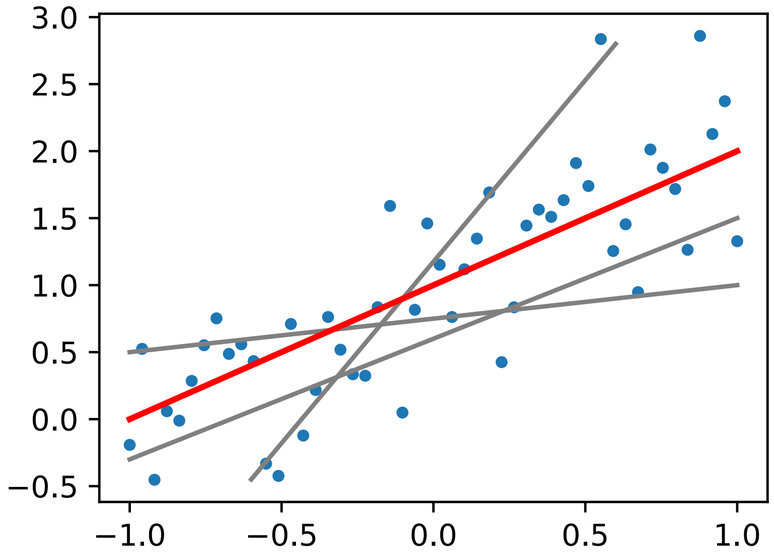
We minimize this error

Actual Value: y_i , Predicted Value: $\hat{y}_i$

$$min(y_i - \hat{y}_i)$$

Linear Regression with One Variable

- Input: x_0
- $y = w_0 x_0 + b$
- Parameters to be estimated:
  - $\hat{w}_0$ (slope)
  - $\hat{b}$: (intercept)
- Predicted outcome:
  $\hat{y} = \hat{w}_0 x_0 + \hat{b}$



How are Parameters Estimated?

- Parameters are estimated by fitting training data
- Many different ways to estimate parameters *w*
  - Often expressed as different “objective functions”
- “Learning” == algorithm to find parameters that optimize the objective function
- Typically means minimizing some loss function

KMeans.fit() learns to minimize distances from points to cluster centers

Examples

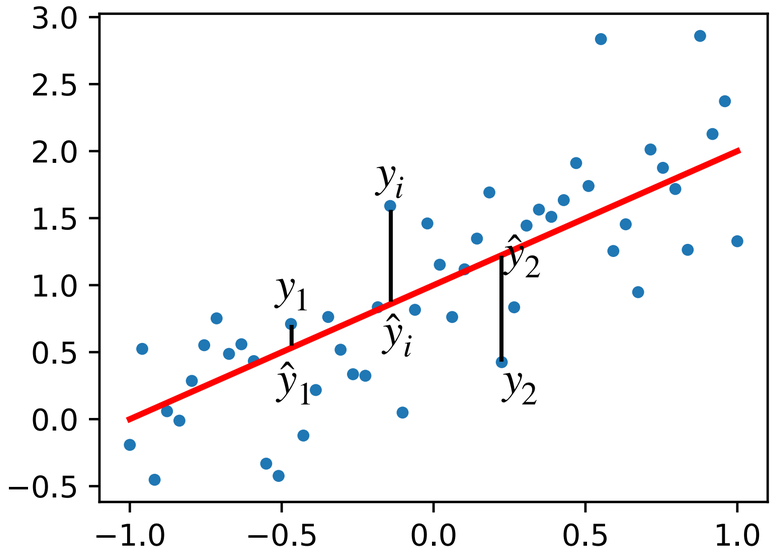
```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

Linear Regression with Ordinary Least Squares (OLS)

- Find parameters that minimize mean squared error:
  - I.e., sum of the squared differences between predicted and actual values

$$L = \sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - (\hat{w}_i x_i + \hat{b}))^2$$

- Goal:
  - Find the w that minimizes L
- Approaches:
  - Stochastic Gradient Descent (SGD)
  - Closed Form Solution



Linear Regression: Assumes relationship between X and y is linear

May (likely) not be true

Prev Up Next

scikit-learn 1.0.2
Other versions

Please cite us if you use the software.

sklearn.linear_model.LinearRegression
Examples using sklearn.linear_model.LinearReg

sklearn.linear_model.LinearRegression

class sklearn.linear_model.LinearRegression(*, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : *bool, default=True*
Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : *bool, default=False*
This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use `StandardScaler` before calling `fit` on an estimator with `normalize=False`.

Deprecated since version 1.0: `normalize` was deprecated in version 1.0 and will be removed in 1.2.

copy_X : *bool, default=True*
If True, X will be copied; else, it may be overwritten.

n_jobs : *int, default=None*
The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly `n_targets > 1` and secondly `X` is sparse or if `positive` is set to `True`. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

positive : *bool, default=False*
When set to `True`, forces t

New in version 0.24.

Attributes:

coef_ : *array of shape (n_fe*
Estimated coefficients for t
this is a 2D array of shape (n_targets, n_features), while if only one target is passed, this is a 1D array of length n_features.

rank_ : *int*

SKLearn.LinearRegression() fits *w* via OLS

Linear Regression is “easy” to interpret

Feature weights w_j tell you how much that feature contributes to the outcome

$$y_i = \sum_{j \in d} w_j x_{i,j} + b$$

sklearn.linear_model.LinearRegression

class sklearn.linear_model.LinearRegression(*, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

Ordinary least squares Linear Regression.

LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Parameters:

fit_intercept : *bool, default=True*
Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).

normalize : *bool, default=False*
This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use `StandardScaler` before calling `fit` on an estimator with `normalize=False`.

Deprecated since version 1.0: `normalize` was deprecated in version 1.0 and will be removed in 1.2.

copy_X : *bool, default=True*
If True, X will be copied; else, it may be overwritten.

n_jobs : *int, default=None*
The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly `n_targets > 1` and secondly `X` is sparse or if `positive` is set to `True`. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

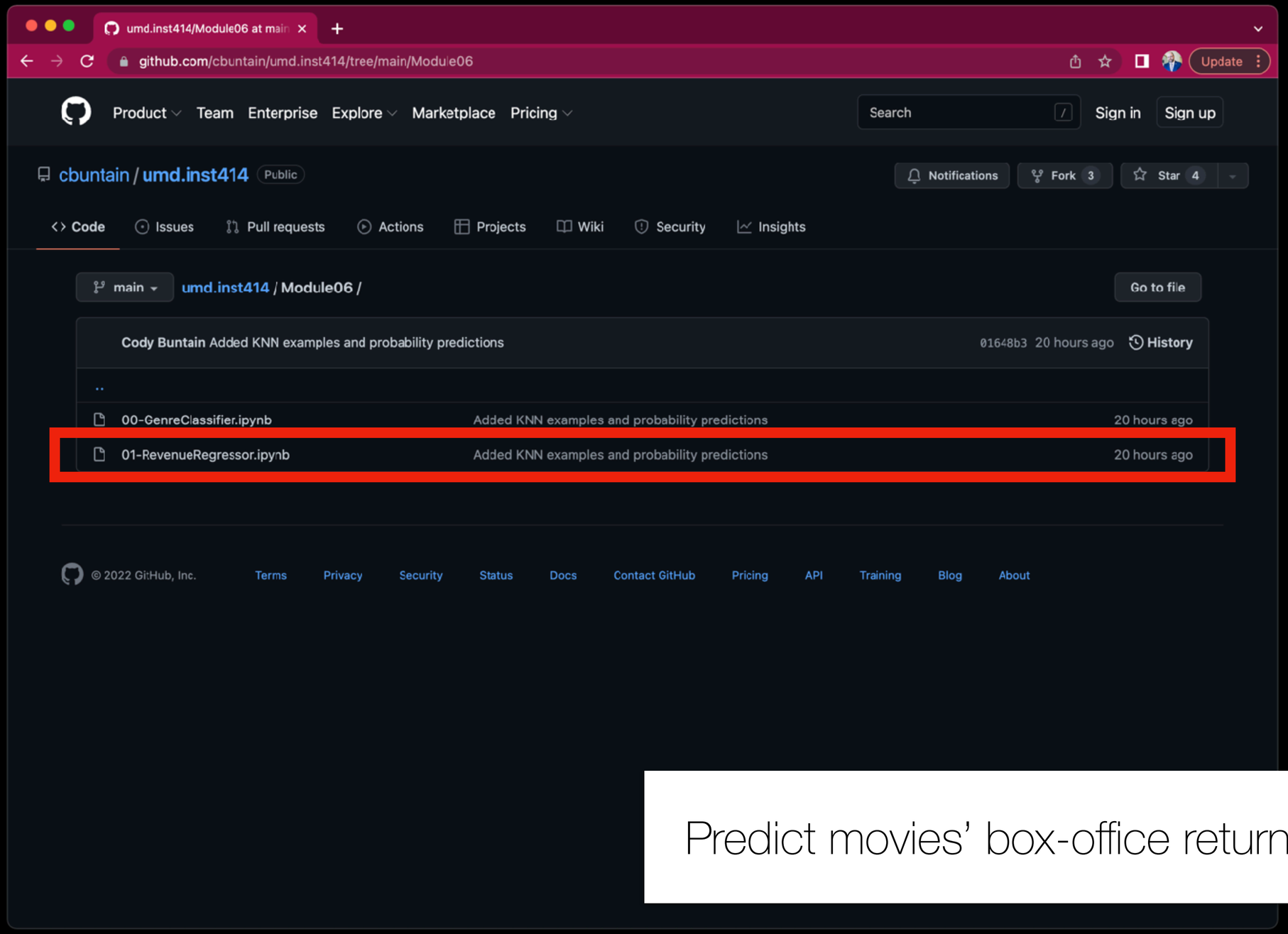
positive : *bool, default=False*
When set to `True`, forces the coefficients to be positive. This option is only supported for dense arrays.

New in version 0.24.

Attributes:

coef_ : *array of shape (n_features,) or (n_targets, n_features)*
Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D), this is a 2D array of shape (n_targets, n_features), while if only one target is passed, this is a 1D array of length n_features.

Feature weights w_j tell you how much that feature contributes to the outcome



Predict movies' box-office returns from actors

jupyter 01-RevenueRegressor Last Checkpoint: Yesterday at 5:23 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=31337)
```

In [ ]:

In [19]:
```
model = LinearRegression()

# Fit our model on train and test data
model.fit(X_train, y_train)
```

Out[19]:
```
LinearRegression()
```

In [ ]:

In [20]:
```
# Inspect the model coefficients
model.coef_
```

Out[20]:
```
array([[ 2.34697851e+00,  8.80858929e-01,  1.68591282e+00,
        -3.21849861e-01, -6.05605476e-01,  2.39691901e+00,
         2.47295217e+00, -7.15601574e-01, -1.33035802e+00,
        -7.06407904e-01, -6.73534376e-01,  9.91663263e-01,
         2.11561749e+00,  1.04363284e-01, -2.79225657e-01,
         8.88611557e-01,  7.17287891e-01, -4.26865348e-01,
        -4.25618067e-02,  7.08680694e-01,  8.44452654e-01,
        -2.65885138e+00, -2.17134032e+00,  6.38664471e-01,
        -1.48323101e+00,  9.10782963e-01, -8.86000395e-01,
        -2.15255603e+00, -5.48220526e-01, -1.24027068e+00,
         1.05869828e-01,  2.33096854e+00,  1.50733817e+00,
         6.27994512e-01, -1.32186432e+00,  1.17909094e+00,
        -3.37768916e-01, -4.99860029e-01,  1.60409212e+00,
         4.19868081e-01, -1.40586136e+00,  8.97436580e-01,
        -3.22769132e+00, -5.17746436e-01, -3.97260013e-02,
        -1.44863637e+00, -7.56482743e-01,  2.51675297e-01,
         1.56750293e+00,  1.51939589e+00,  1.02784074e+00,
         1.28728963e+00, -8.82431657e-01,  2.37814459e+00,
        -2.42706886e+00, -1.39577127e+00,  2.24178372e+00,
```

In [21]:
```
# Model intercept, or close to
# the average of the transformed gross
model.intercept_
```

Out[21]:
```
array([15.69495412])
```

In [ ]:

In [22]:
```
# Zip the actors with their model coefficients, so we can inspect actors' roles on gross revenue
actor_weights = list(zip([actor_id_to_name_map[i] for i in X_train.columns], model.coef_[0,:]))
```

jupyter 01-RevenueRegressor Last Checkpoint: Yesterday at 5:23 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run | Markdown

Out[21]:
```
array([15.69495412])
```

In [ ]:

In [22]:
```
# Zip the actors with their model coefficients, so we can inspect actors' roles on gross revenue
actor_weights = list(zip([actor_id_to_name_map[i] for i in X_train.columns], model.coef_[0,:]))

print("Actors with most positive influence on gross:")
for tup in sorted(actor_weights, key=lambda t: t[1], reverse=True)[:10]:
    print(tup)

print("\nActors with most negative influence on gross:")
for tup in sorted(actor_weights, key=lambda t: t[1], reverse=True)[-10:]:
    print(tup)
```

```
Actors with most positive influence on gross:
('Jason Lee', 3.4032622773493206)
('Rupert Grint', 3.3450674434871086)
('Bill Hader', 3.2736763486145395)
('Tom Hanks', 3.2241697804190808)
('Tobey Maguire', 3.1130334919510148)
('Will Smith', 3.002037998759295)
('T.J. Miller', 2.9608238084799345)
('Michael Keaton', 2.9598477986375253)
('Benedict Cumberbatch', 2.8673377581056703)
('Tobin Bell', 2.8474388183876966)

Actors with most negative influence on gross:
('Thomas Jane', -2.4478803329361556)
('Tom Sizemore', -2.6588513787467085)
('Mark Webber', -2.940567373835491)
('Udo Kier', -2.9415251368281505)
('Michael Angarano', -3.225905200513357)
('Malcolm McDowell', -3.2276913167828414)
('Shea Whigham', -3.503492474753421)
('Kevin Corrigan', -3.518483064494919)
('Wes Bentley', -3.942561670950922)
('Adam Beach', -4.2875070636910895)
```

In [ ]:

In [23]:
```
predicted_gross = model.predict(X_test)
```

In [24]:
```
sqr_errors = []

for predicted,actual in zip(predicted_gross[:,0], y_test["gross"]):
    error = predicted - actual
```

A movie starring Jason Lee, Rupert Grint, Tom Hanks, etc. receives a higher box office return

jupyter 01-RevenueRegressor Last Checkpoint: Yesterday at 5:23 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3

Run Markdown

Out[21]:
```
array([15.69495412])
```

In [ ]:

In [22]:
```
# Zip the actors with their model coefficients, so we can inspect actors' roles on gross revenue
actor_weights = list(zip([actor_id_to_name_map[i] for i in X_train.columns], model.coef_[0,:]))

print("Actors with most positive influence on gross:")
for tup in sorted(actor_weights, key=lambda t: t[1], reverse=True)[:10]:
    print(tup)

print("\nActors with most negative influence on gross:")
for tup in sorted(actor_weights, key=lambda t: t[1], reverse=True)[-10:]:
    print(tup)
```

```
Actors with most positive influence on gross:
('Jason Lee', 3.4032622773493206)
('Rupert Grint', 3.3450674434871086)
('Bill Hader', 3.2736763486145395)
('Tom Hanks', 3.2241697804190808)
('Tobey Maguire', 3.1130334919510148)
('Will Smith', 3.002037998759295)
('T.J. Miller', 2.9608238084799345)
('Michael Keaton', 2.9598477986375253)
('Benedict Cumberbatch', 2.8673377581056703)
('Tobin Bell', 2.8474388183876966)

Actors with most negative influence on gross:
('Thomas Jane', -2.4478803329361556)
('Tom Sizemore', -2.6588513787467085)
('Mark Webber', -2.940567373835491)
('Udo Kier', -2.9415251368281505)
('Michael Angarano', -3.225905200513357)
('Malcolm McDowell', -3.2276913167828414)
('Shea Whigham', -3.503492474753421)
('Kevin Corrigan', -3.518483064494919)
('Wes Bentley', -3.942561670950922)
('Adam Beach', -4.2875070636910895)
```

In [ ]:

In [23]:
```
predicted_gross = model.predict(X_test)
```

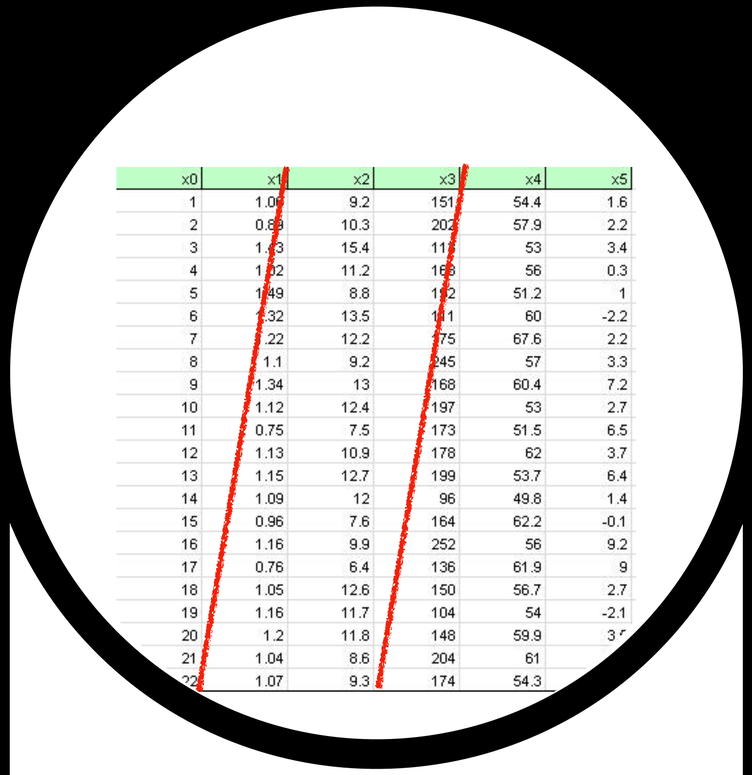
In [24]:
```
sqr_errors = []

for predicted,actual in zip(predicted_gross[:,0], y_test["gross"]):
    error = predicted - actual
```

Movies starring Adam Beach or Malcom McDowell do worse

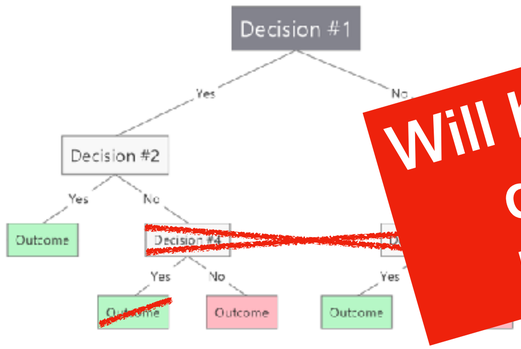
Regularization: Methods for penalizing model complexity

How might you penalize model complexity?

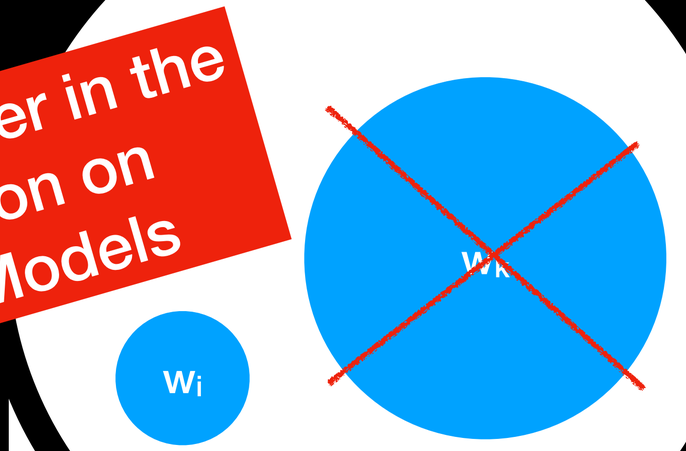


x0	x1	x2	x3	x4	x5
1	1.06	9.2	151	54.4	1.6
2	0.89	10.3	202	57.9	2.2
3	1.43	15.4	113	53	3.4
4	1.02	11.2	168	56	0.3
5	1.49	8.8	192	51.2	1
6	1.32	13.5	111	60	-2.2
7	1.22	12.2	175	67.6	2.2
8	1.1	9.2	245	57	3.3
9	1.34	13	168	60.4	7.2
10	1.12	12.4	197	53	2.7
11	0.75	7.5	173	51.5	6.5
12	1.13	10.9	178	62	3.7
13	1.15	12.7	199	53.7	6.4
14	1.09	12	96	49.8	1.4
15	0.96	7.6	164	62.2	-0.1
16	1.16	9.9	252	56	9.2
17	0.76	6.4	136	61.9	9
18	1.05	12.6	150	56.7	2.7
19	1.16	11.7	104	54	-2.1
20	1.2	11.8	148	59.9	3.5
21	1.04	8.6	204	61	
22	1.07	9.3	174	54.3	

You use fewer dimensions/features

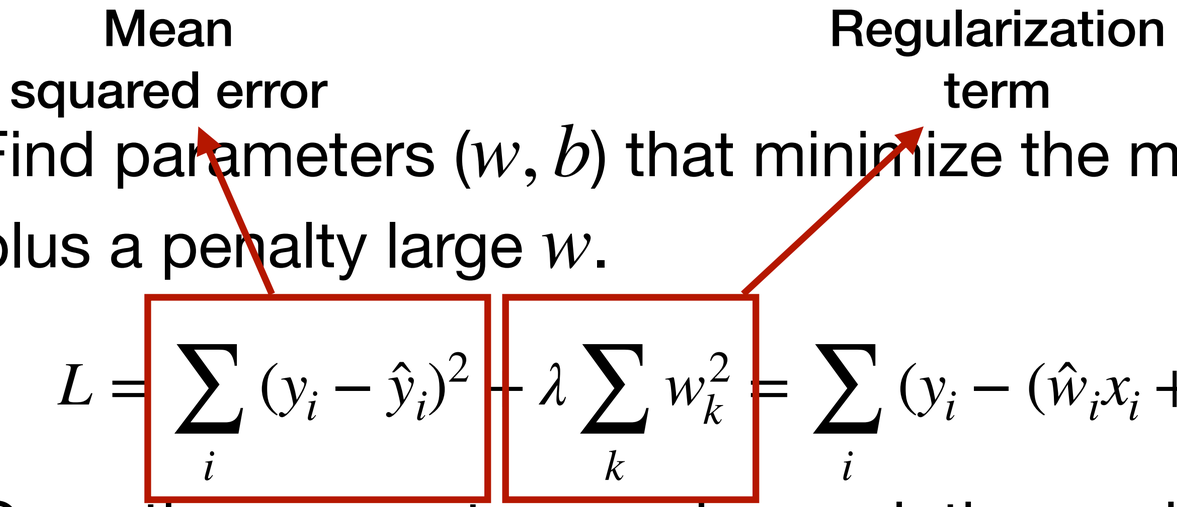


Model enforces sparsity (mostly 0s)



Model forces feature weights to be small

Ridge Regression (L2 Regularization)



- Find parameters (w, b) that minimize the mean squared error plus a penalty large w.

$$L = \sum_i (y_i - \hat{y}_i)^2 + \lambda \sum_k w_k^2 = \sum_i (y_i - (\hat{w}_i x_i + \hat{b}))^2 + \lambda \sum_k w_k^2$$

- Once the parameters are learned, the prediction is the same.
  - $\hat{y} = \hat{w}x + \hat{b}$ (No Penalty term!)
- Penalty enforces regularization
  - L2 regularization — minimizing the sum of squares of w
  - λ controls the strength of the regularization.
  - Larger λ pushes w to 0

Lasso Regression (L1 Regularization)

- Find parameters (w, b) that minimize the mean squared error plus a penalty large w

$$L = \sum_i (y_i - \hat{y}_i)^2 + \lambda \sum_k |w_k| = \sum_i (y_i - (\hat{w}_i x_i + \hat{b}))^2 + \lambda \sum_k |w_k|$$

- Lasso tends to shrink many w to zero, which reduces the number of features

This Module's Learning Objectives

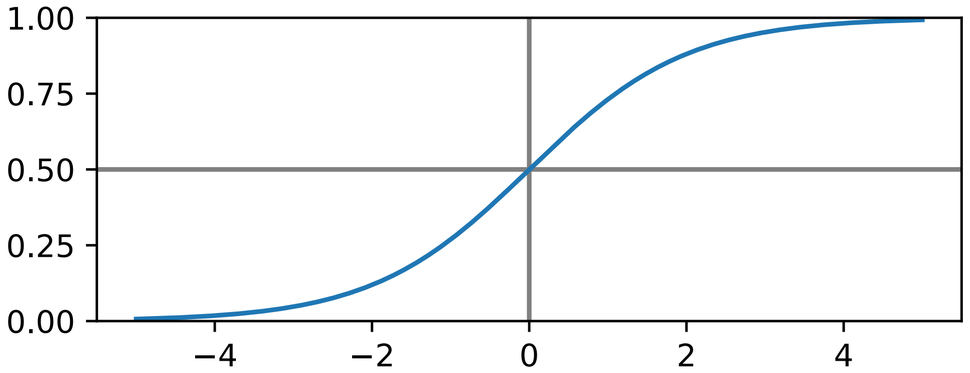
Part 2

- Define regularization and its use of improving generalizability
- Use linear regression models to predict scores for data elements
- Explain how linear regression can be adapted for classification
- Extract the class-label probabilities for model outputs

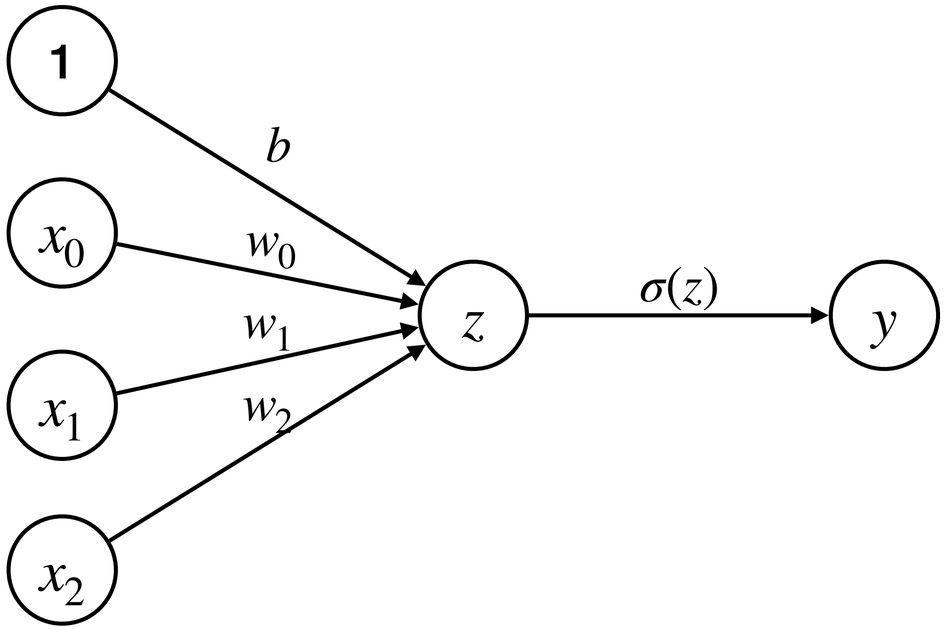
“Transform” Linear Regression for Classification

- Regression: $X \rightarrow y, y \in (-\infty, +\infty)$
- Binary classification: $X \rightarrow y, y \in \{0,1\}$

- What if we can mapping $(-\infty, +\infty) \rightarrow \{0,1\}$
- The *logistic sigmoid* function: $\sigma(x) = \dfrac{1}{1+e^{-x}}$

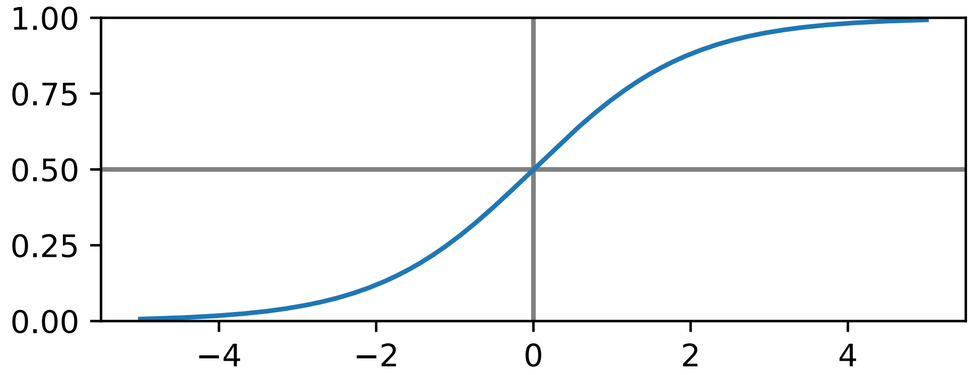


Logistic Regression



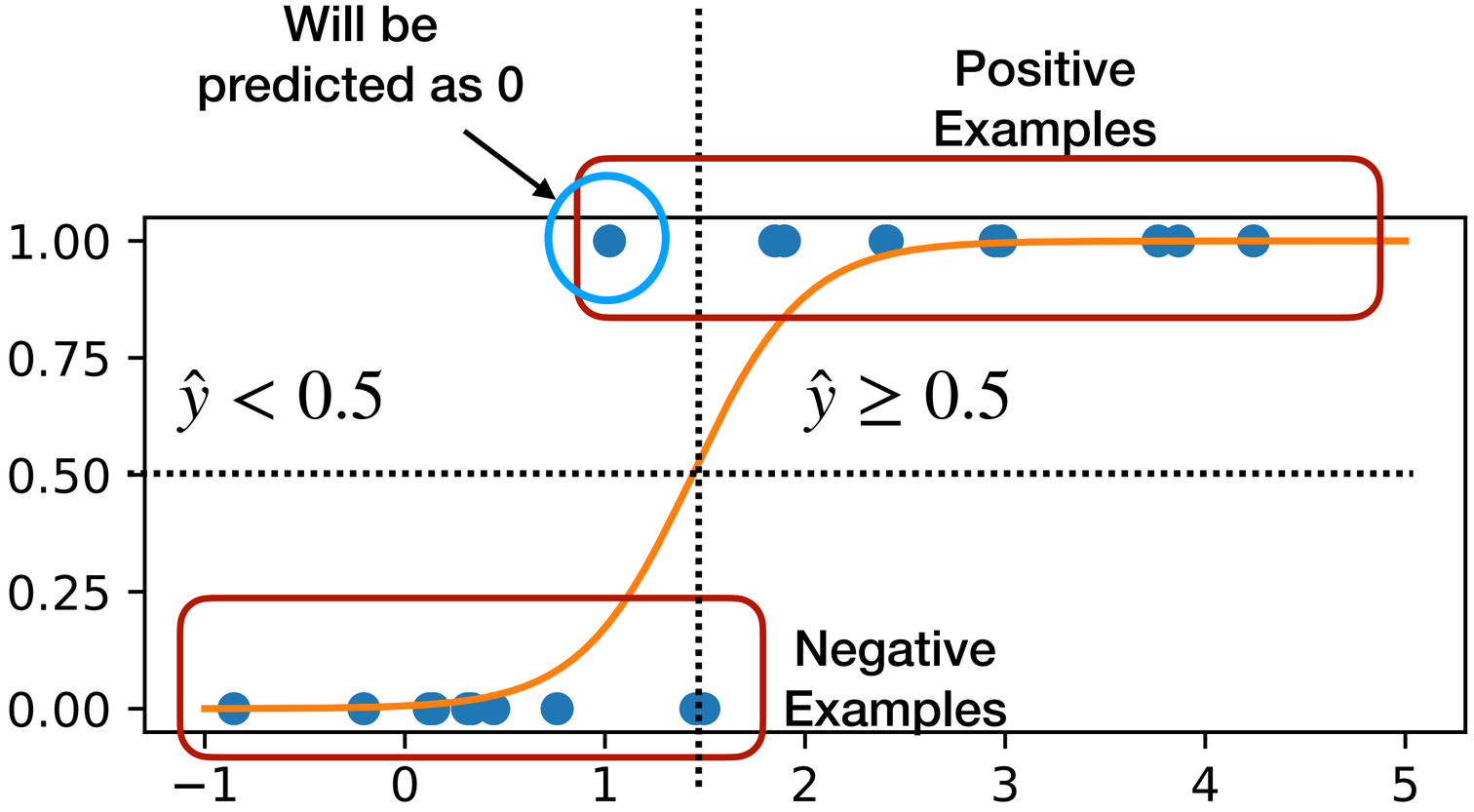
$$1, \text{ if } y > 0.5$$
$$0, \text{ if } y < 0.5$$

$$\hat{z} = \hat{b} + \hat{w}_0 x_0 + \cdots + \hat{w}_k x_k$$
$$\hat{y} = \sigma(\hat{z})$$
$$= \sigma(\hat{b} + \hat{w}_0 x_0 + \cdots + \hat{w}_k x_k)$$
$$= \frac{1}{1 + \exp[-(\hat{b} + \hat{w}_0 x_0 + \cdots + \hat{w}_k x_k)]}$$

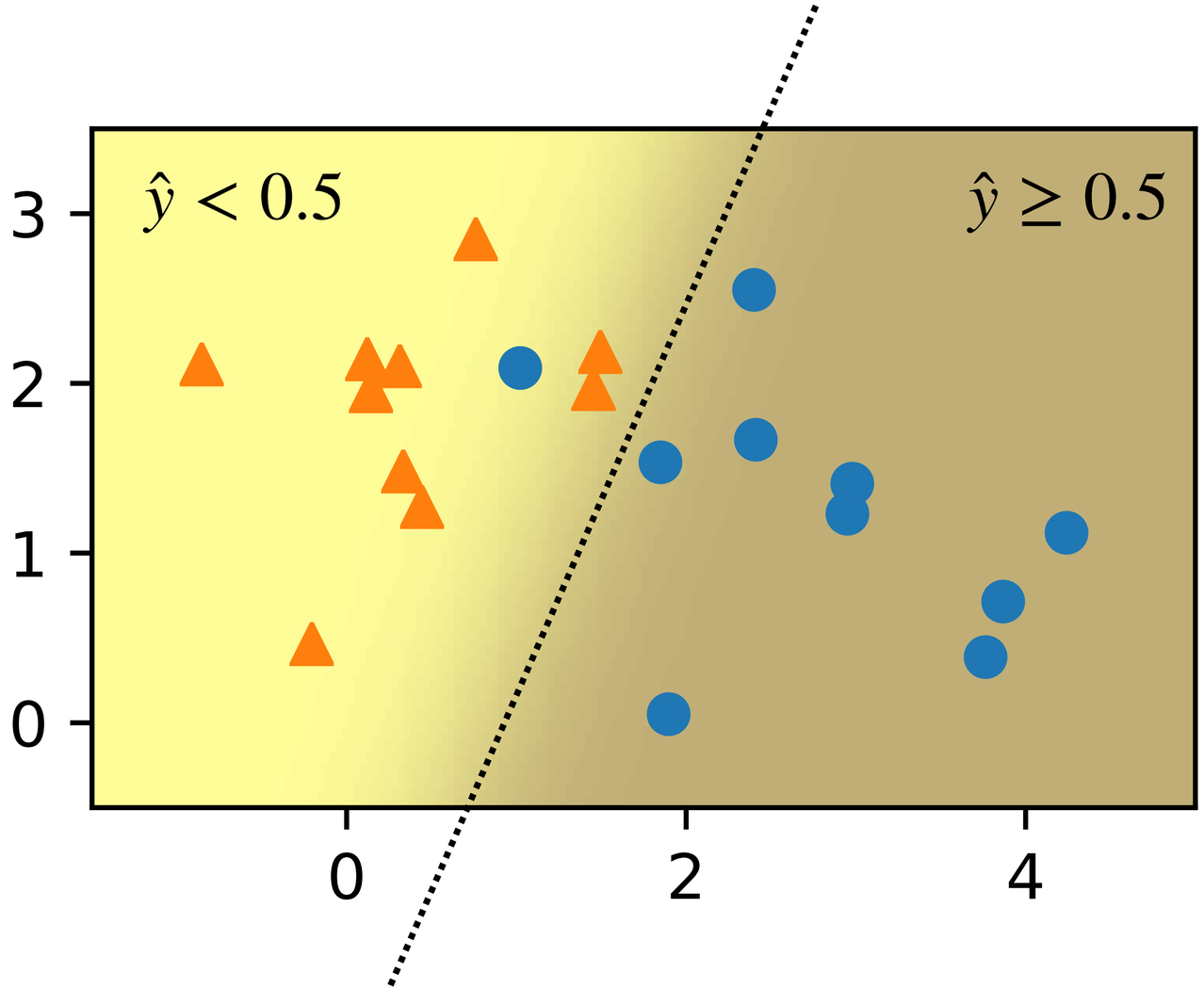


The Logistic function transforms real-value input to an output number y between 0 and 1, which can be interpreted as the *probability* that the input object belongs to class 1, given its input features $(x_0, x_1, \cdots, x_k)$

Logistic Regression: 1-D Example

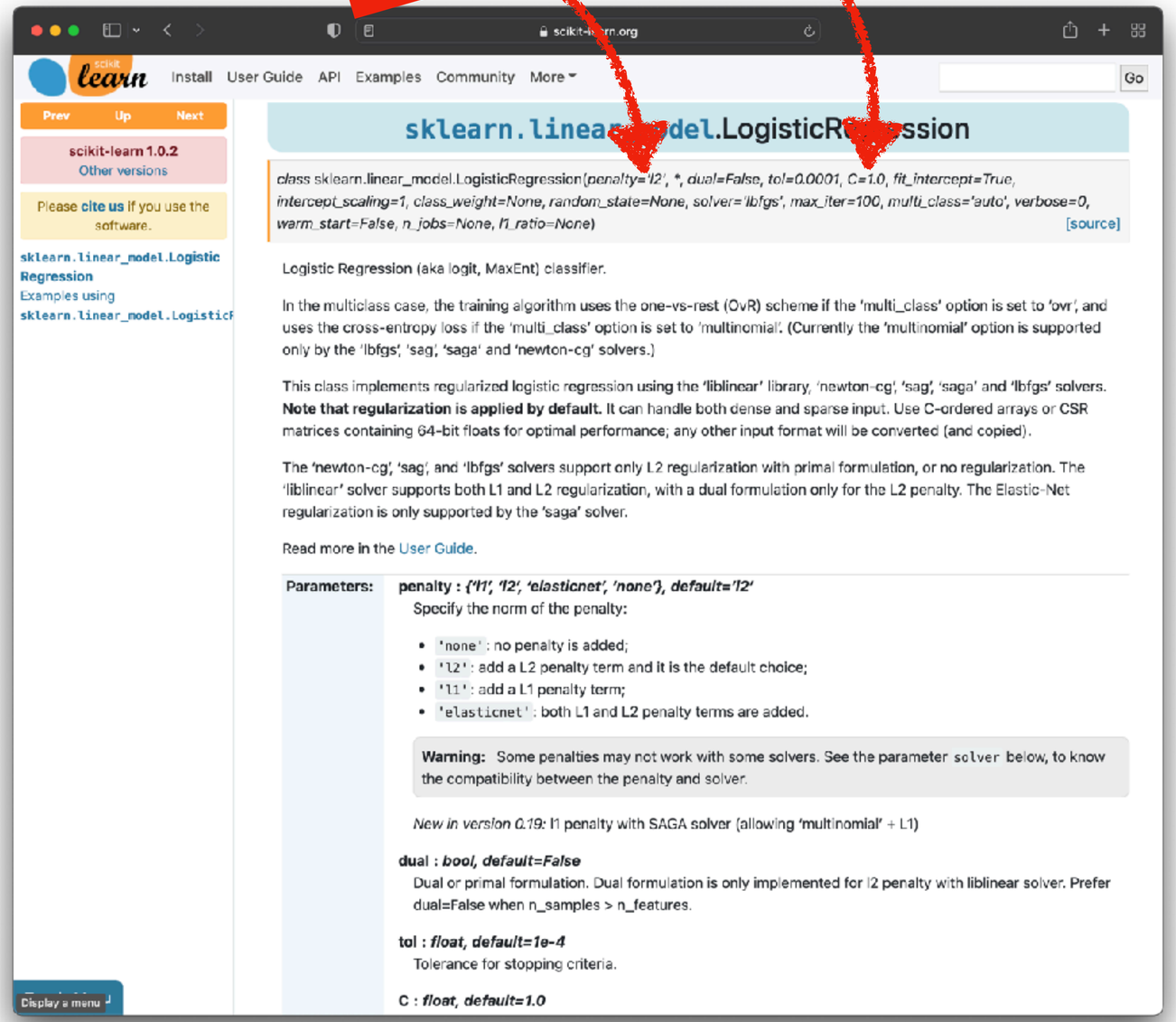


Logistic Regression: 2-D example



Logistic Regression: Regularizatio

- L2 regularization is “on” by default in scikit-learn
  - Parameter C controls the strength of the regularization.



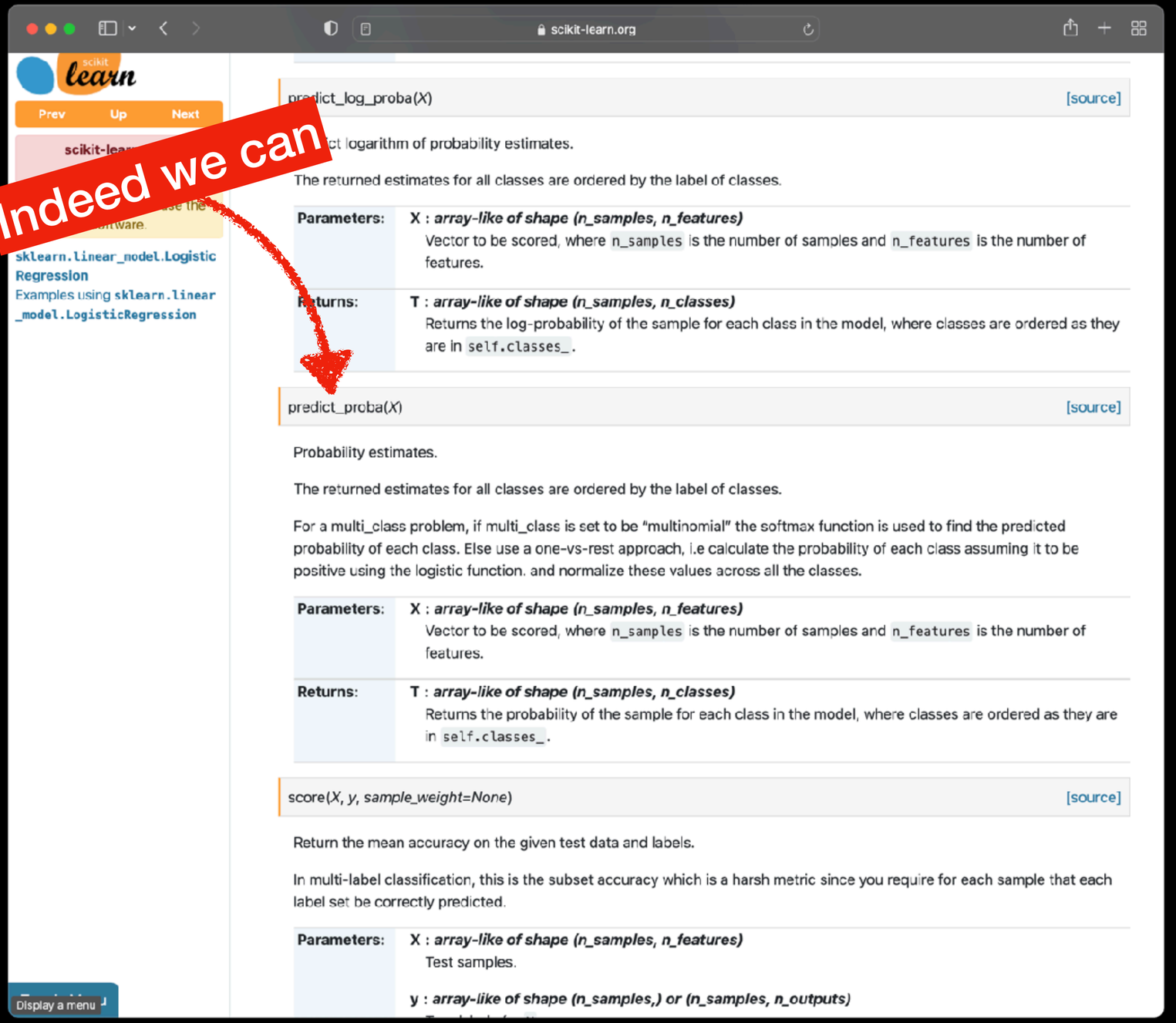
This Module's Learning Objectives

Part 2

- Define regularization and its use of improving generalizability
- Use linear regression models to predict scores for data elements
- Explain how linear regression can be adapted for classification
- Extract the class-label probabilities for model outputs

If Logistic Regression classifies labels via probability thresholds…

Can we extract these probabilities directly?


scikit-learn.org
scikit learn
Prev
Up
Next
sklearn.linear_model.LogisticRegression
Examples using sklearn.linear_model.LogisticRegression
Indeed we can
predict_log_proba(X) [source]
ct logarithm of probability estimates.
The returned estimates for all classes are ordered by the label of classes.
Parameters: X : array-like of shape (n_samples, n_features)
Vector to be scored, where n_samples is the number of samples and n_features is the number of features.
Returns: T : array-like of shape (n_samples, n_classes)
Returns the log-probability of the sample for each class in the model, where classes are ordered as they are in self.classes_.
predict_proba(X) [source]
Probability estimates.
The returned estimates for all classes are ordered by the label of classes.
For a multi_class problem, if multi_class is set to be "multinomial" the softmax function is used to find the predicted probability of each class. Else use a one-vs-rest approach, i.e calculate the probability of each class assuming it to be positive using the logistic function. and normalize these values across all the classes.
Parameters: X : array-like of shape (n_samples, n_features)
Vector to be scored, where n_samples is the number of samples and n_features is the number of features.
Returns: T : array-like of shape (n_samples, n_classes)
Returns the probability of the sample for each class in the model, where classes are ordered as they are in self.classes_.
score(X, y, sample_weight=None) [source]
Return the mean accuracy on the given test data and labels.
In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.
Parameters: X : array-like of shape (n_samples, n_features)
Test samples.
y : array-like of shape (n_samples,) or (n_samples, n_outputs)
Display a menu

Why would you want these probabilities?

Perhaps you only care about instances with high confidence

Instances with low confidence (prob. ~ 0.5) are interesting/hard samples

Questions

Prof. Cody Buntain | @codybuntain | cbuntain@umd.edu
Director, Information Ecosystems Lab