

Supervised learning

SUPERVISED LEARNING WITH SCIKIT-LEARN



Andreas Müller

Core developer, scikit-learn

What is machine learning?

- The art and science of:
 - Giving computers the ability to learn to make decisions from data
 - without being explicitly programmed!
- Examples:
 - Learning to predict whether an email is spam or not
 - Clustering wikipedia entries into different categories
- Supervised learning: Uses labeled data
- Unsupervised learning: Uses unlabeled data

Unsupervised learning

- Uncovering hidden patterns from unlabeled data
- Example:
 - Grouping customers into distinct categories (Clustering)

Reinforcement learning

- Software agents interact with an environment
 - Learn how to optimize their behavior
 - Given a system of rewards and punishments
 - Draws inspiration from behavioral psychology
- Applications
 - Economics
 - Genetics
 - Game playing
- AlphaGo: First computer to defeat the world champion in Go

Supervised learning

- Predictor variables/features and a target variable
- Aim: Predict the target variable, given the predictor variables
 - Classification: Target variable consists of categories
 - Regression: Target variable is continuous

Predictor variables

Target variable

The diagram illustrates a dataset for supervised learning. On the left, a table titled "Predictor variables" contains five rows of data with four columns: sepal length (cm), sepal width (cm), petal length (cm), and petal width (cm). The rows are indexed from 0 to 4. A blue arrow points downwards from the header "Predictor variables" to the top of the table. On the right, a separate column titled "Target variable" contains five entries: species, setosa, setosa, setosa, setosa, and setosa. A blue arrow points downwards from the header "Target variable" to the top of this column.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

species

setosa

setosa

setosa

setosa

Naming conventions

- Features = predictor variables = independent variables
- Target variable = dependent variable = response variable

Supervised learning

- Automate time-consuming or expensive manual tasks
 - Example: Doctor's diagnosis
- Make predictions about the future
 - Example: Will a customer click on an ad or not?
- Need labeled data
 - Historical data with labels
 - Experiments to get labeled data
 - Crowd-sourcing labeled data

Supervised learning in Python

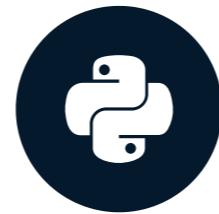
- We will use scikit-learn/sklearn
 - Integrates well with the SciPy stack
- Other libraries
 - TensorFlow
 - keras

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Exploratory data analysis

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

The Iris dataset



Features:

- Petal length
- Petal width
- Sepal length
- Sepal width

Target variable: Species

- Versicolor
- Virginica
- Setosa

The Iris dataset in scikit-learn

```
from sklearn import datasets  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
plt.style.use('ggplot')  
iris = datasets.load_iris()  
type(iris)
```

```
sklearn.datasets.base.Bunch
```

```
print(iris.keys())
```

```
dict_keys(['data', 'target_names', 'DESCR', 'feature_names', 'target'])
```

The Iris dataset in scikit-learn

```
type(iris.data), type(iris.target)
```

```
(numpy.ndarray, numpy.ndarray)
```

```
iris.data.shape
```

```
(150, 4)
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Exploratory data analysis (EDA)

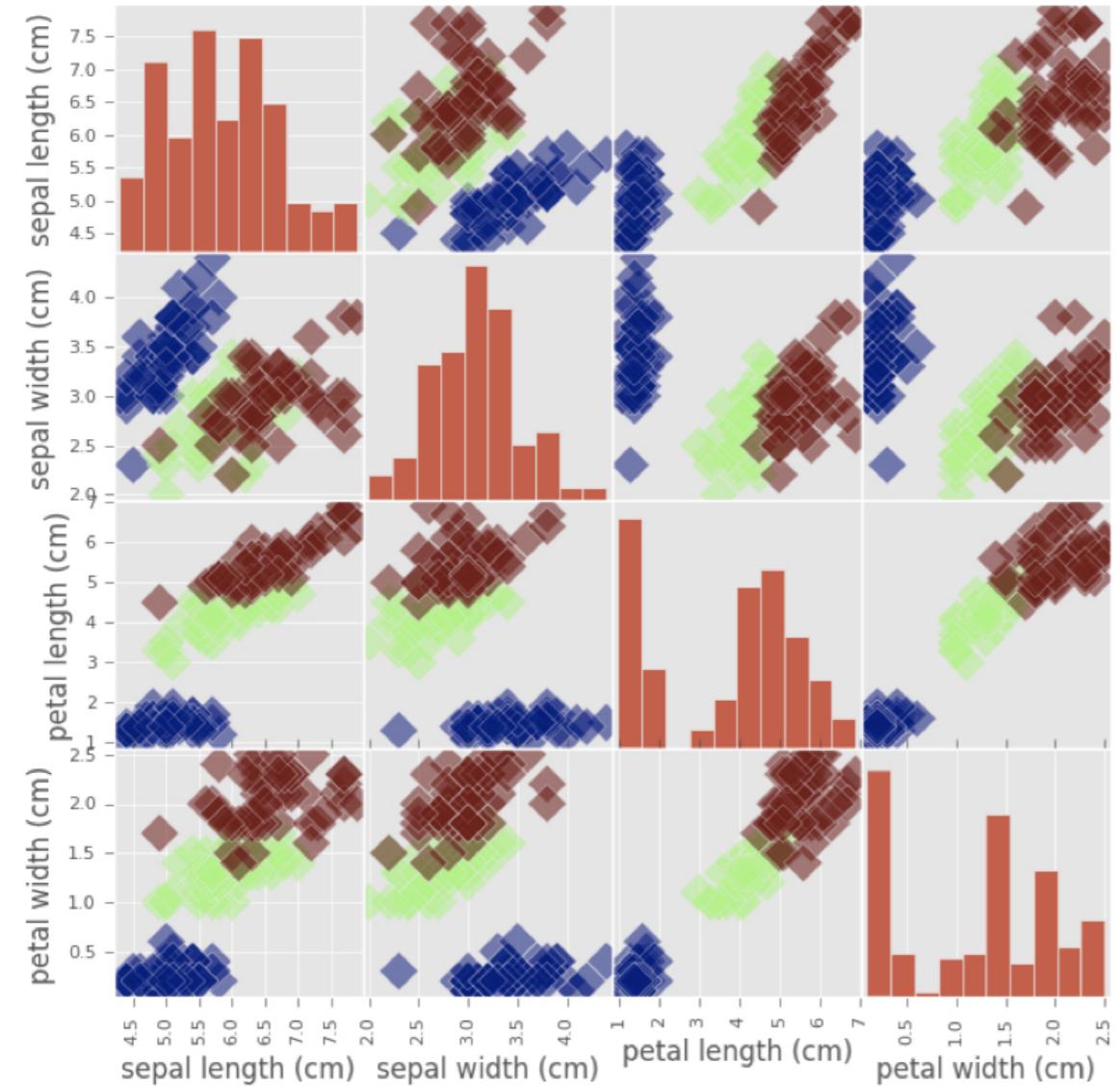
```
X = iris.data  
y = iris.target  
df = pd.DataFrame(X, columns=iris.feature_names)  
print(df.head())
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

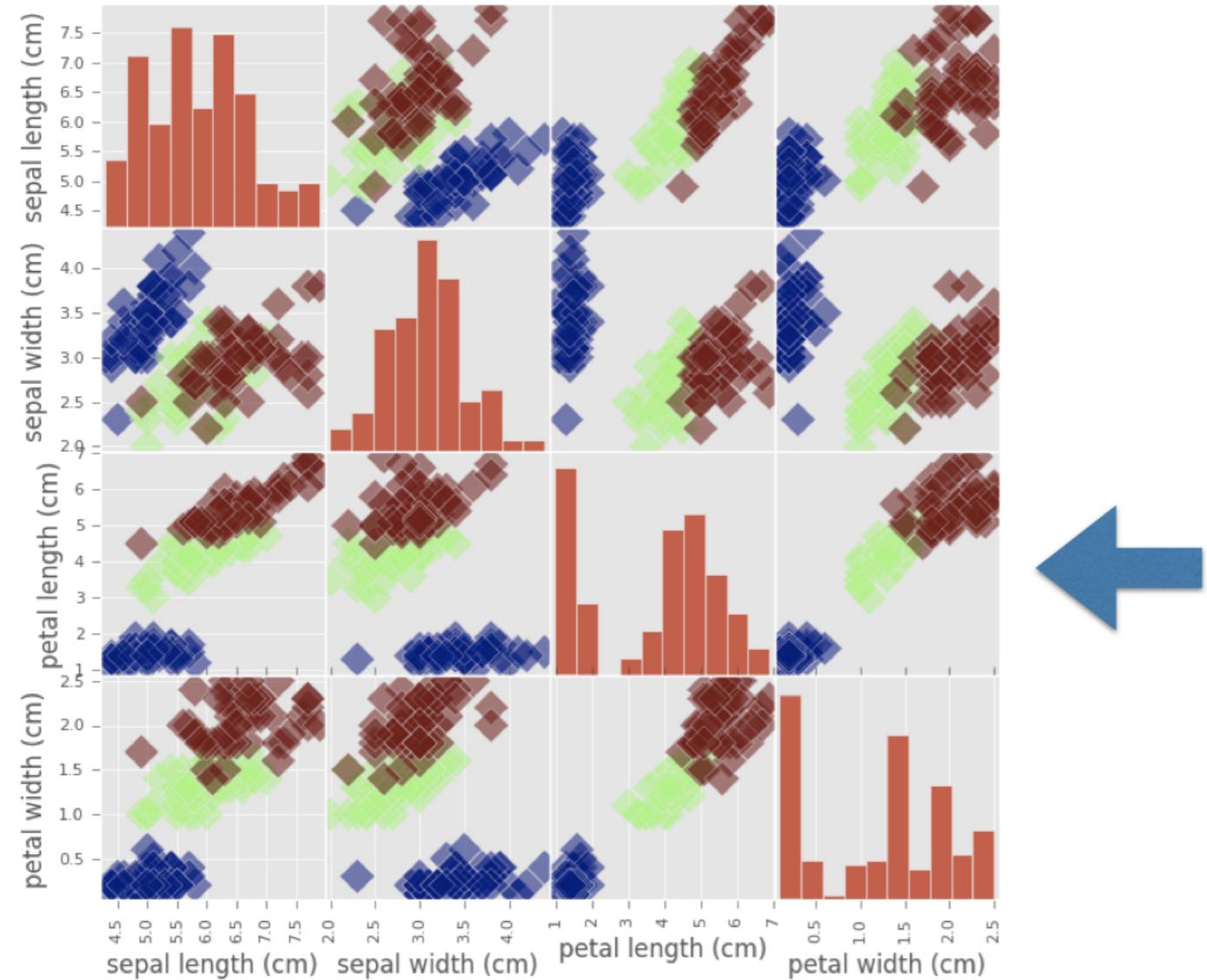
Visual EDA

```
_ = pd.plotting.scatter_matrix(df, c = y, figsize = [8, 8],  
                               s=150, marker = 'D')
```

Visual EDA



Visual EDA

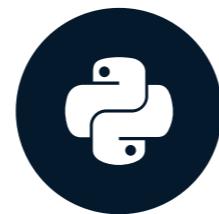


Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

The classification challenge

SUPERVISED LEARNING WITH SCIKIT-LEARN

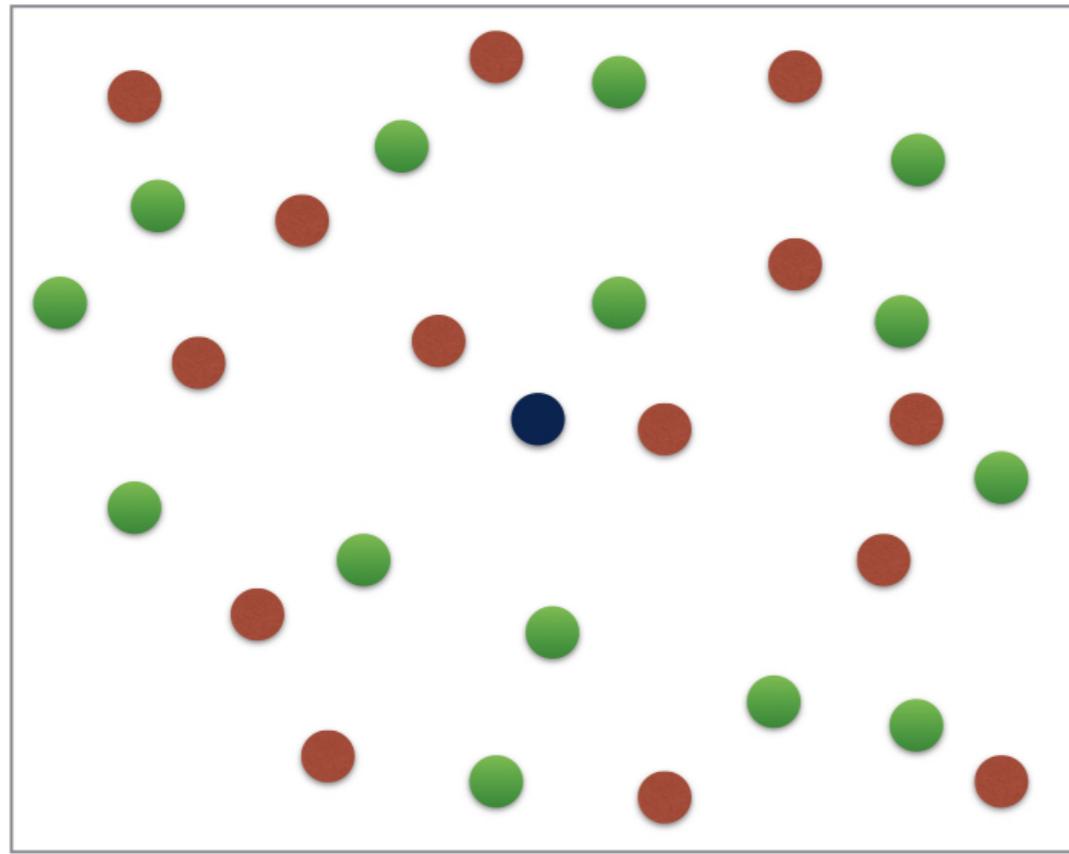


Hugo Bowne-Anderson
Data Scientist, DataCamp

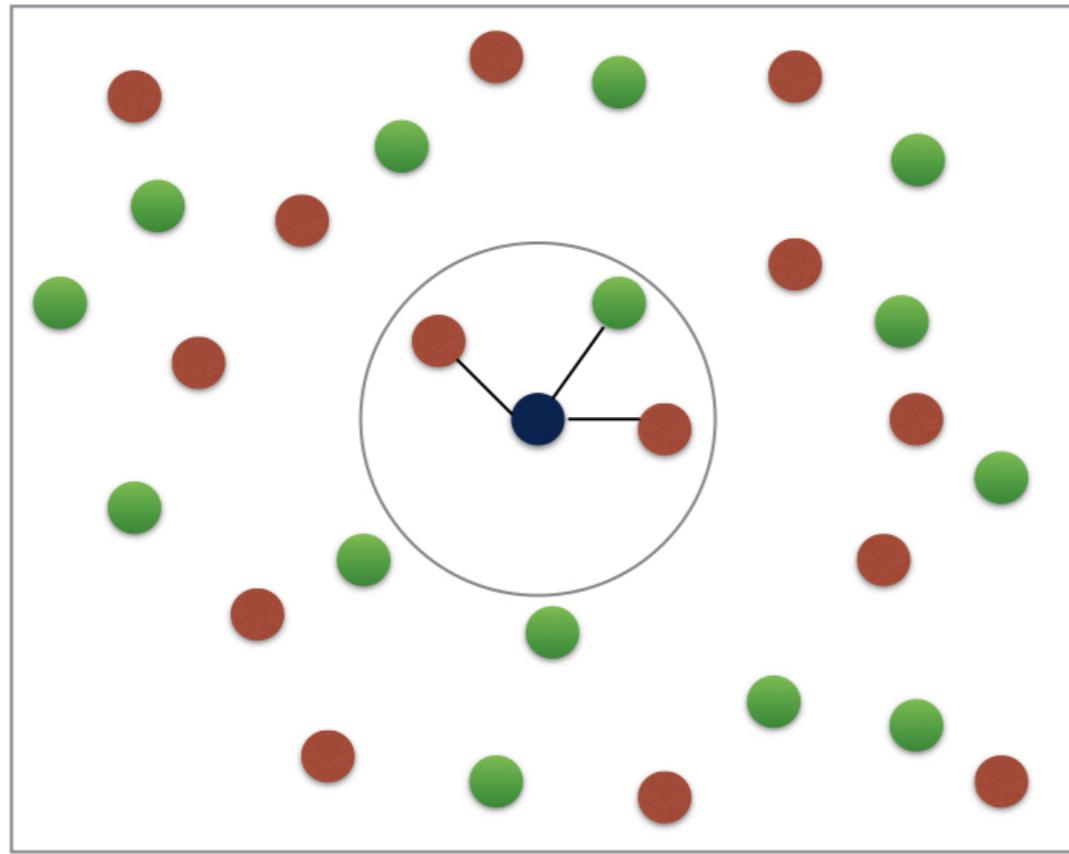
k-Nearest Neighbors

- Basic idea: Predict the label of a data point by
 - Looking at the ‘k’ closest labeled data points
 - Taking a majority vote

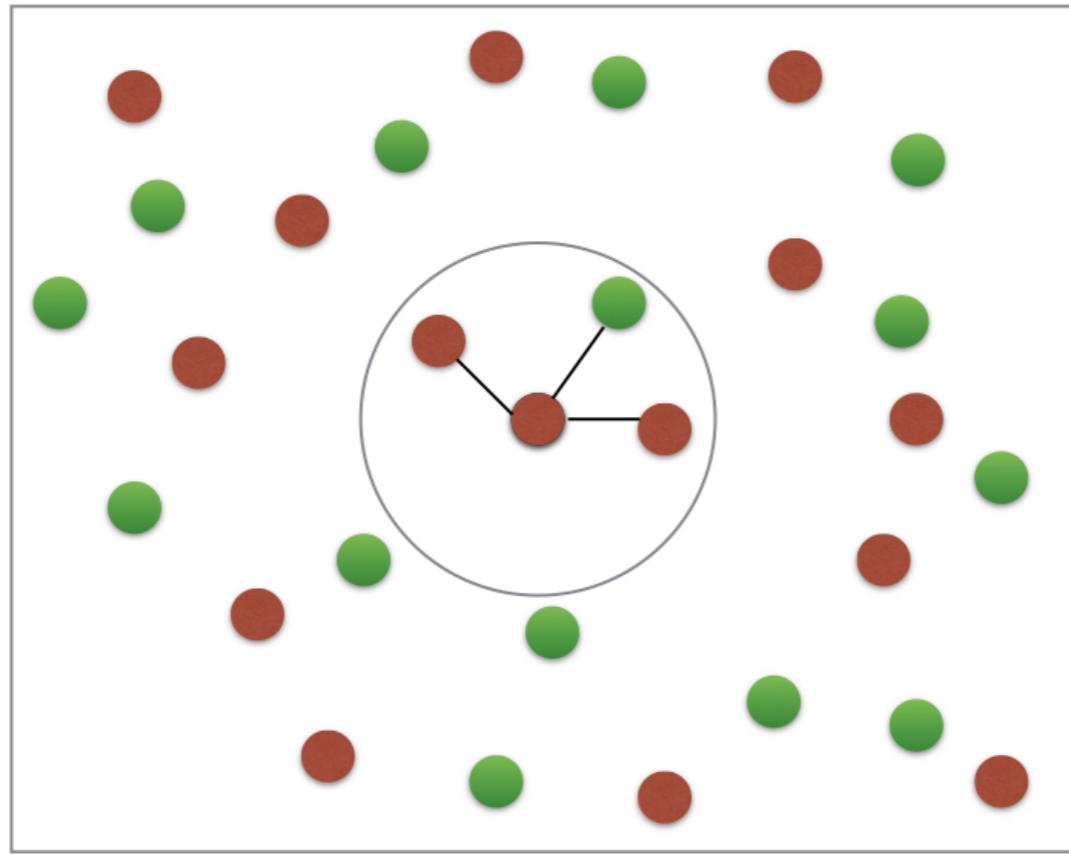
k-Nearest Neighbors



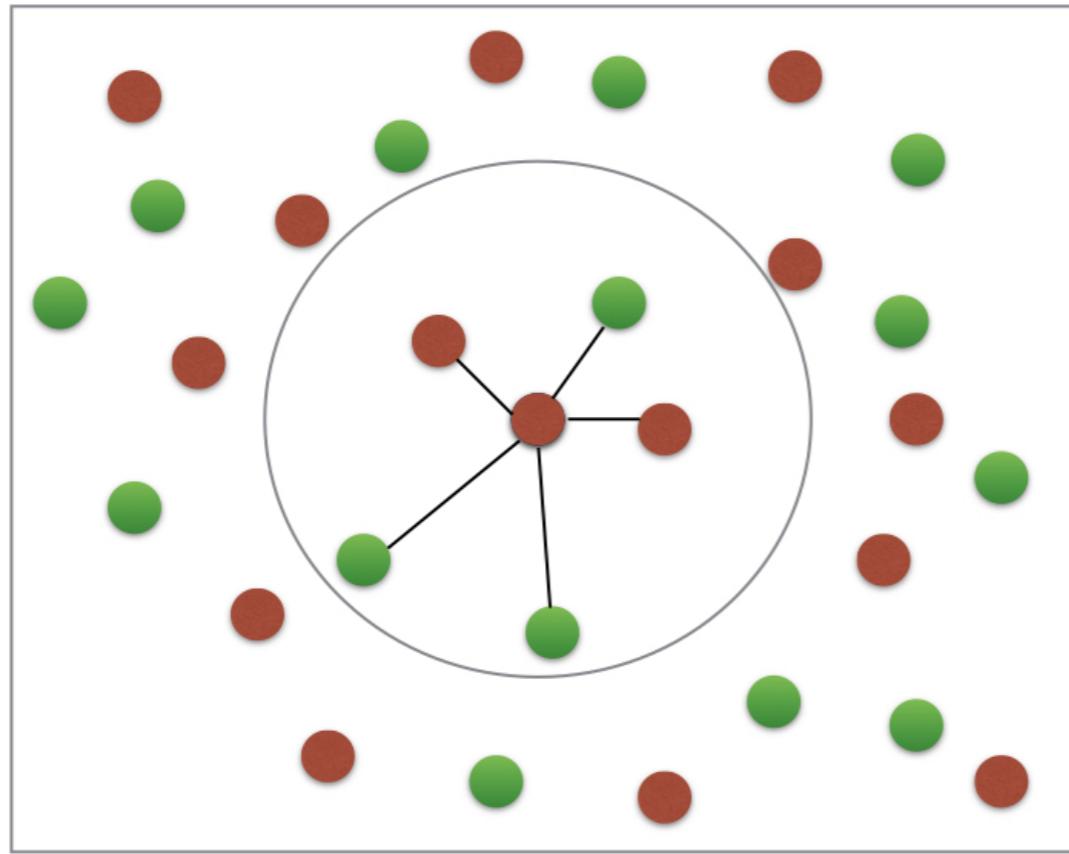
k-Nearest Neighbors



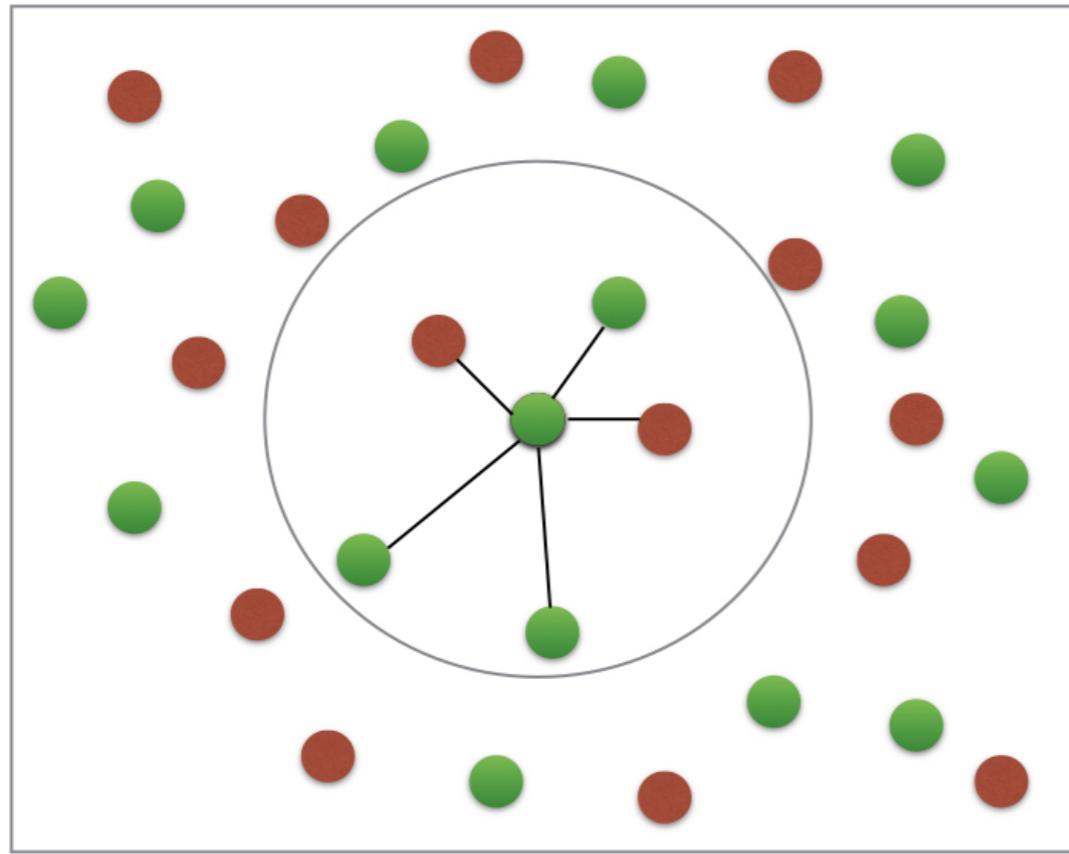
k-Nearest Neighbors



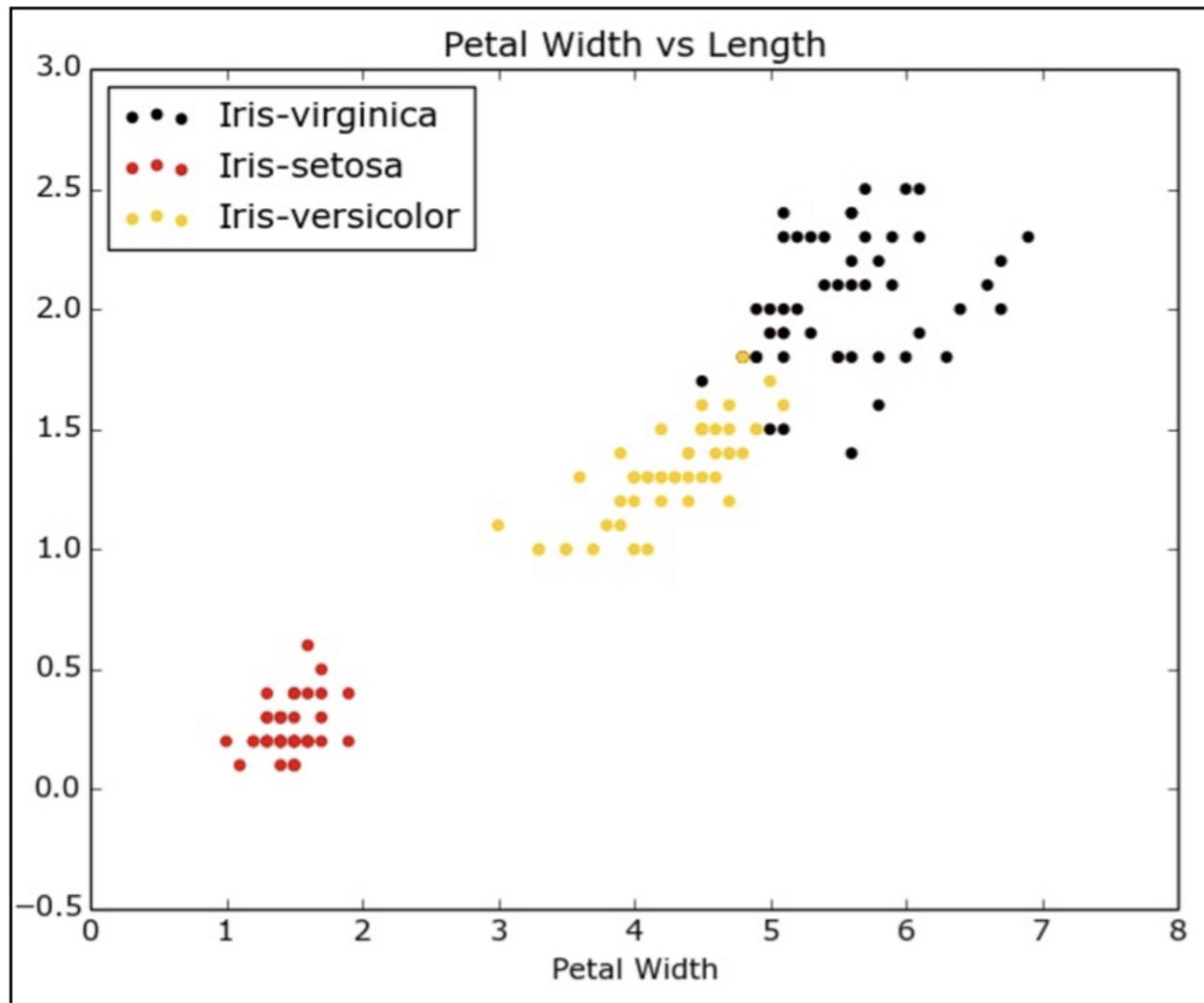
k-Nearest Neighbors



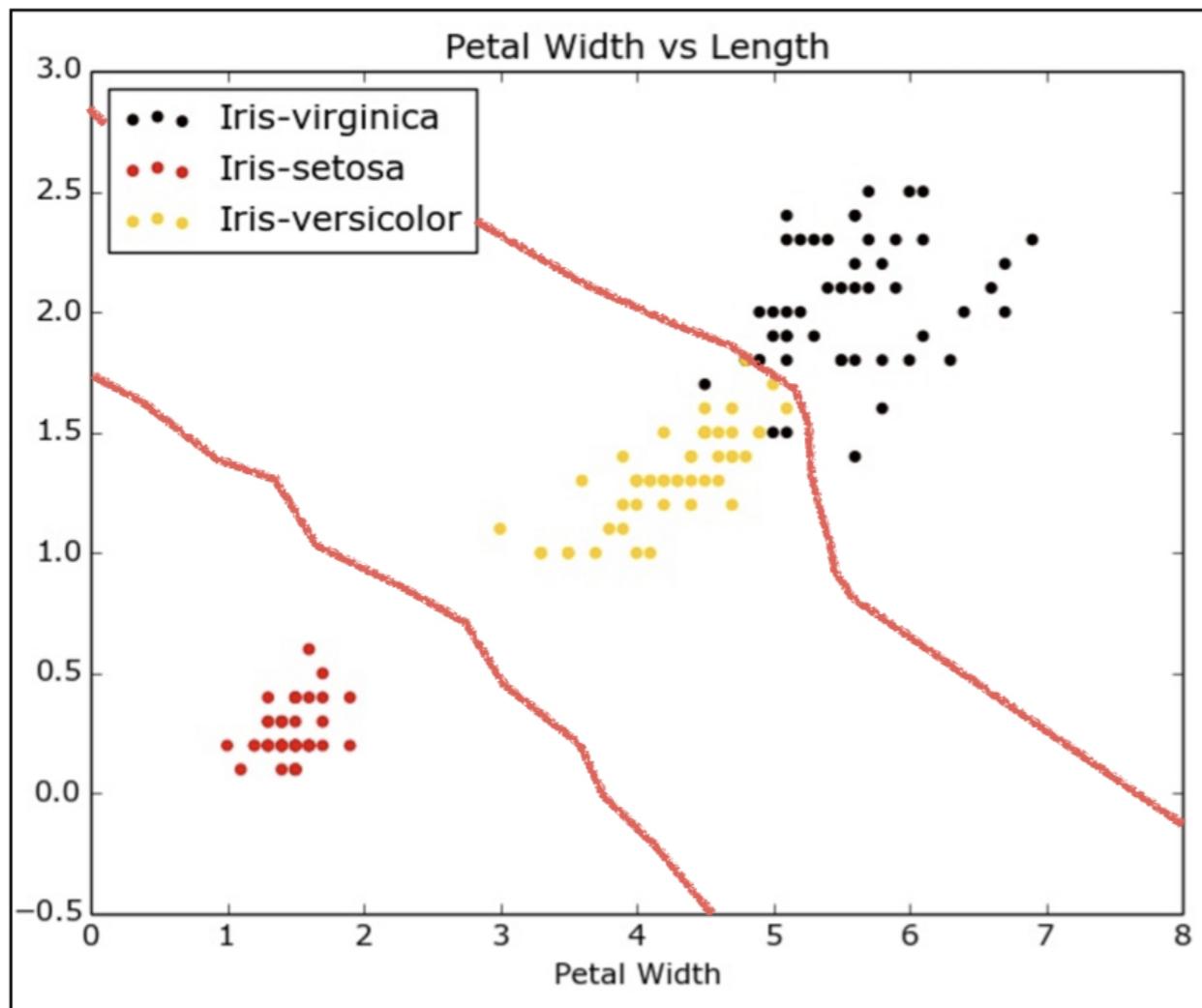
k-Nearest Neighbors



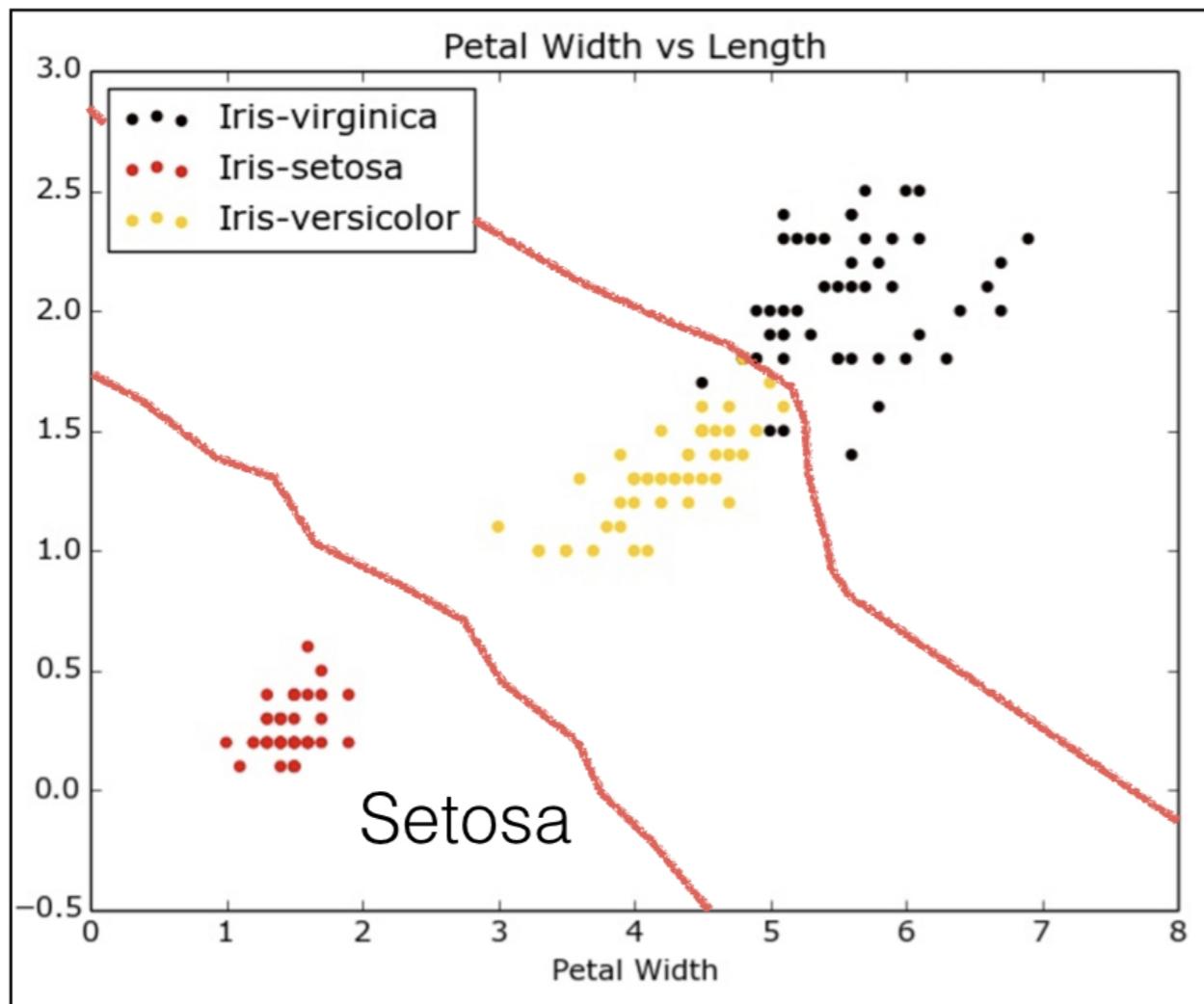
k-NN: Intuition



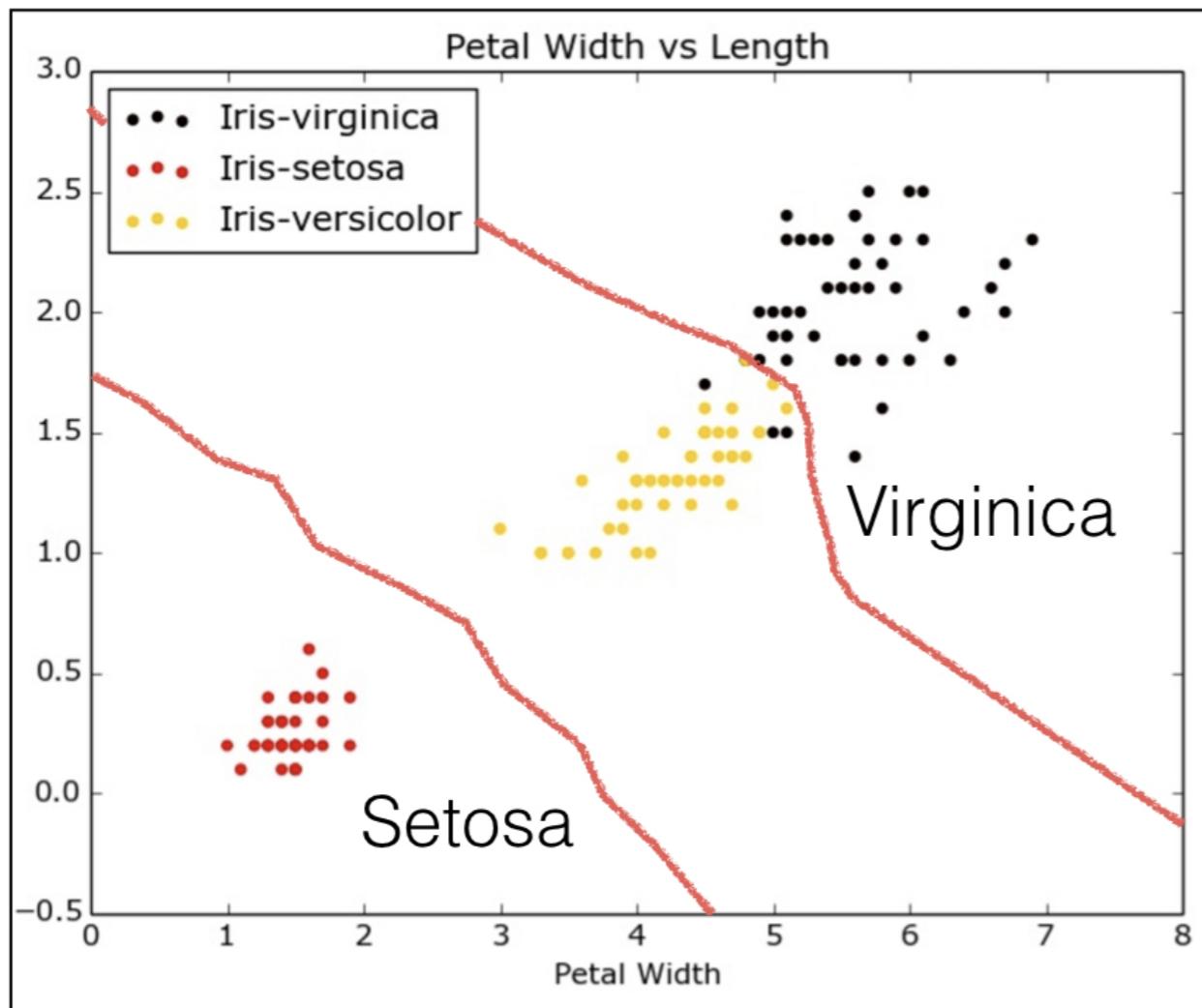
k-NN: Intuition



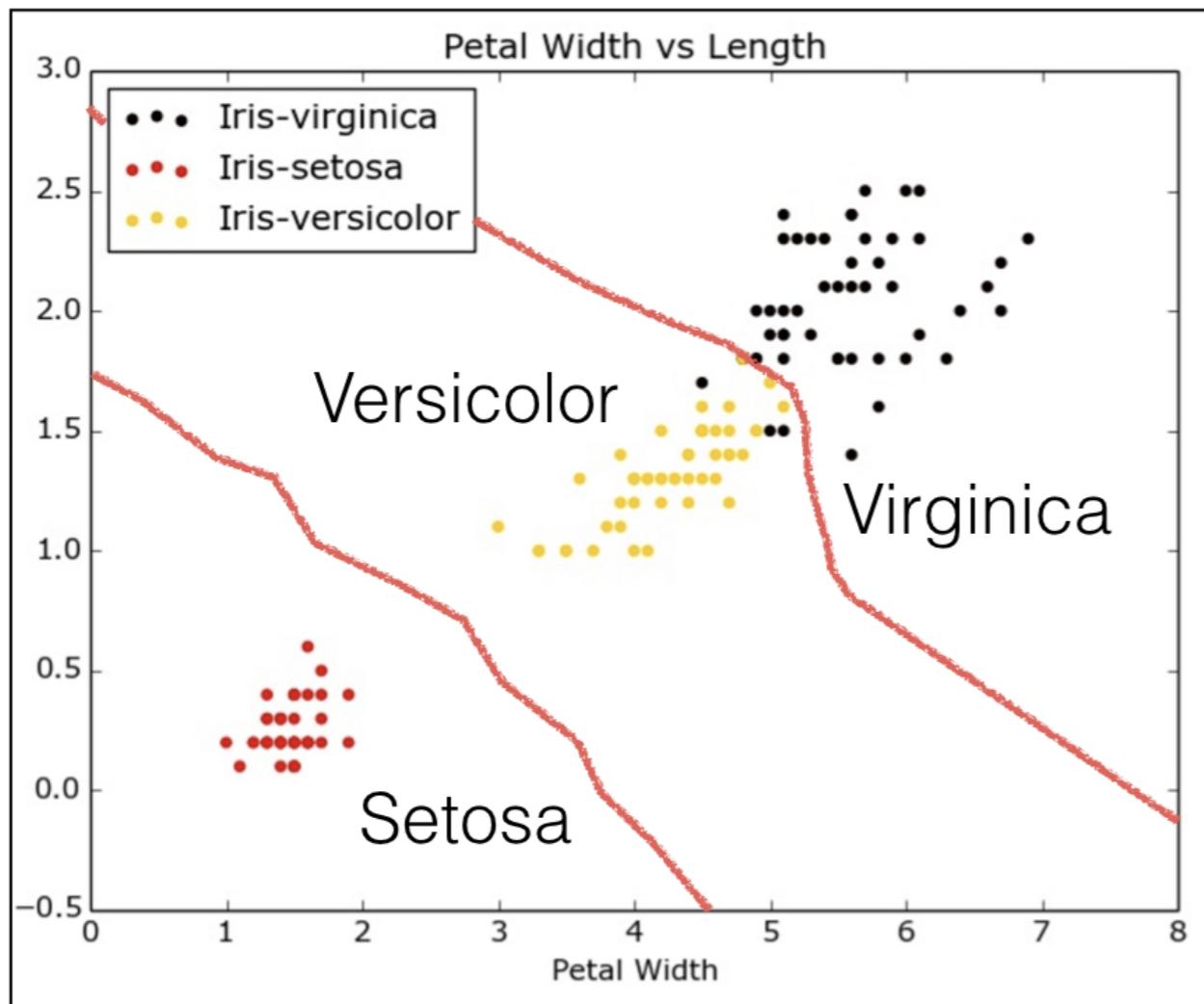
k-NN: Intuition



k-NN: Intuition



k-NN: Intuition



Scikit-learn fit and predict

- All machine learning models implemented as Python classes
 - They implement the algorithms for learning and predicting
 - Store the information learned from the data
- Training a model on the data = ‘fitting’ a model to the data
 - `.fit()` method
- To predict the labels of new data: `.predict()` method

Using scikit-learn to fit a classifier

```
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n_neighbors=6)  
knn.fit(iris['data'], iris['target'])
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30,  
metric='minkowski', metric_params=None, n_jobs=1,  
n_neighbors=6, p=2, weights='uniform')
```

```
iris['data'].shape
```

```
(150, 4)
```

```
iris['target'].shape
```

```
(150,)
```

Predicting on unlabeled data

```
X_new = np.array([[5.6, 2.8, 3.9, 1.1],  
                 [5.7, 2.6, 3.8, 1.3],  
                 [4.7, 3.2, 1.3, 0.2]])
```

```
prediction = knn.predict(X_new)  
X_new.shape
```

```
(3, 4)
```

```
print('Prediction: {}'.format(prediction))
```

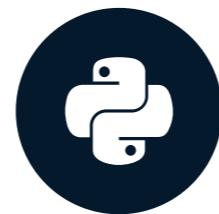
```
Prediction: [1 1 0]
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Measuring model performance

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

Measuring model performance

- In classification, accuracy is a commonly used metric
- Accuracy = Fraction of correct predictions
- Which data should be used to compute accuracy?
- How well will the model perform on new data?

Measuring model performance

- Could compute accuracy on data used to fit classifier
- NOT indicative of ability to generalize
- Split data into training and test set
- Fit/train the classifier on the training set
- Make predictions on test set
- Compare predictions with the known labels

Train/test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.3,
                     random_state=21, stratify=y)
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
```

Test set predictions:

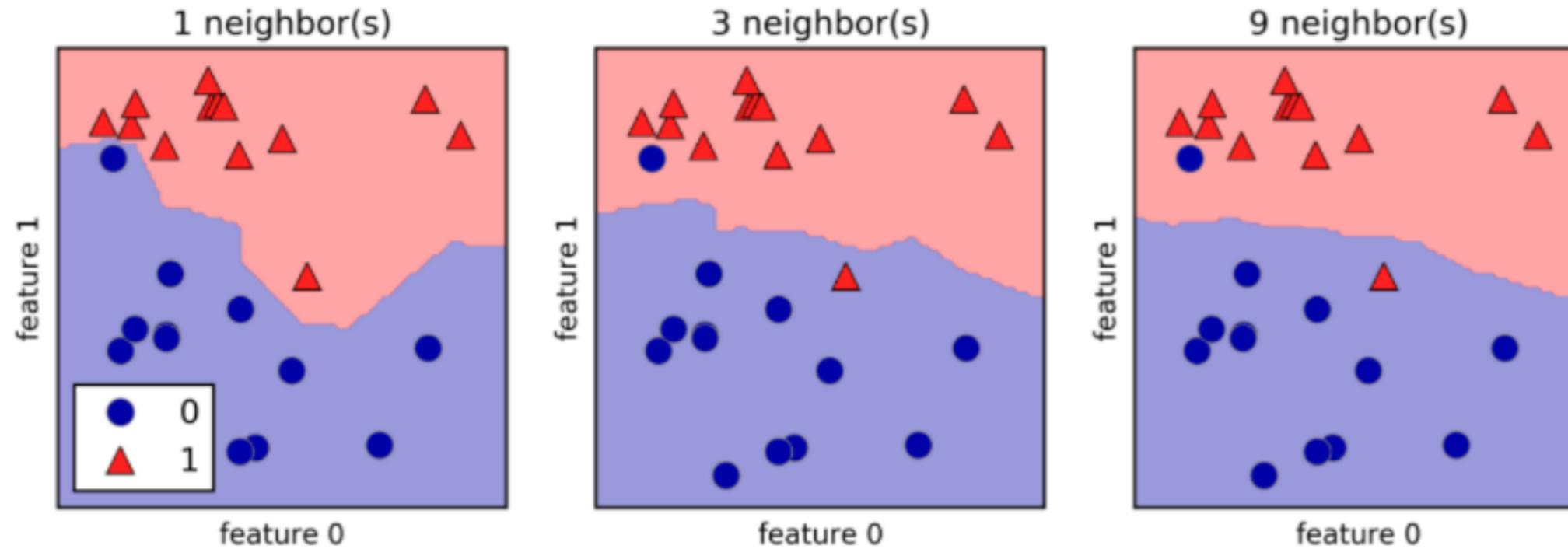
```
[2 1 2 2 1 0 1 0 0 1 0 2 0 2 2 0 0 0 1 0 2 2 2 0 1 1 1 0 0
 1 2 2 0 0 2 2 1 1 2 1 1 0 2 1]
```

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

```
0.9555555555555556
```

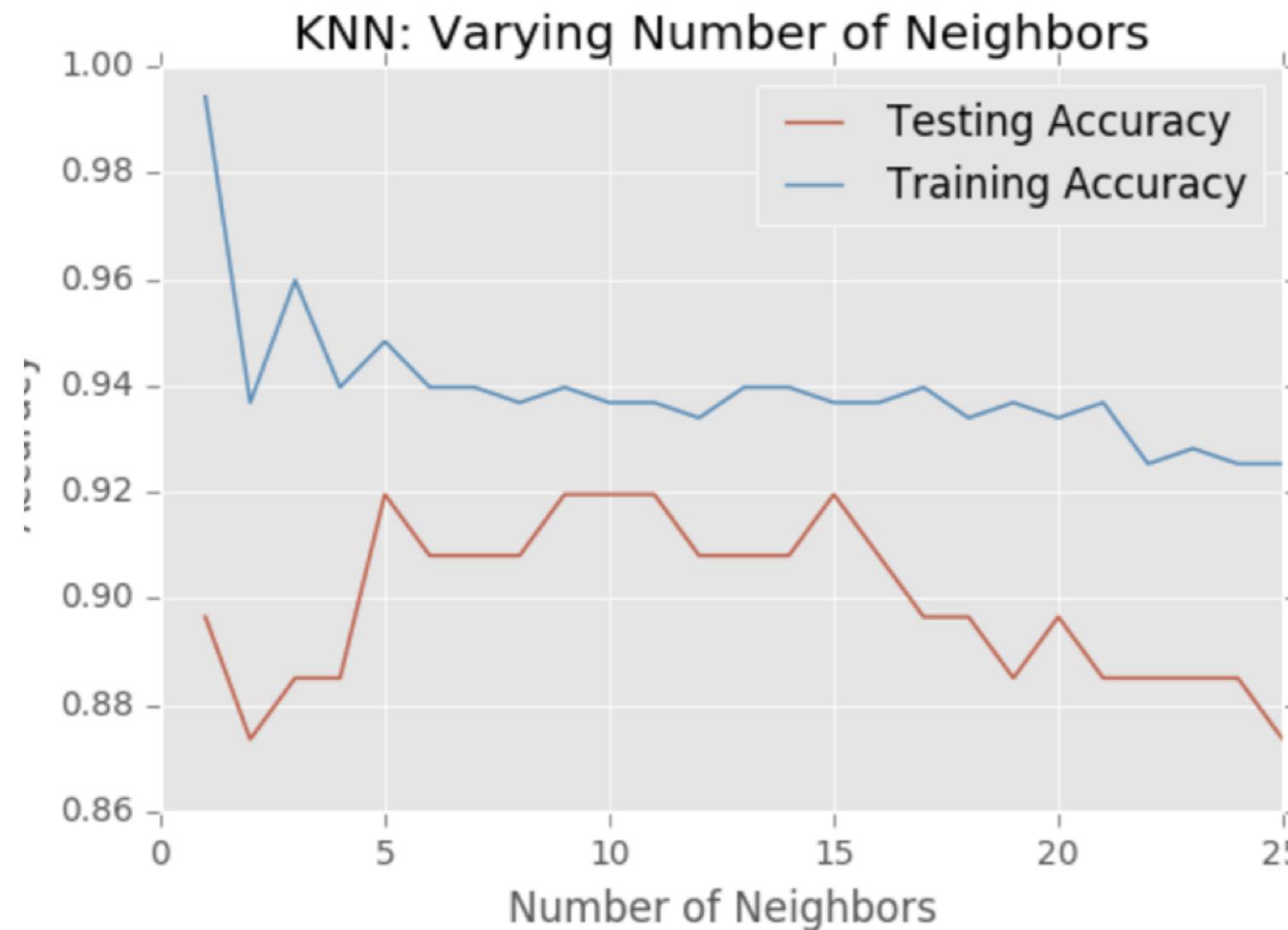
Model complexity

- Larger k = smoother decision boundary = less complex model
- Smaller k = more complex model = can lead to overfitting

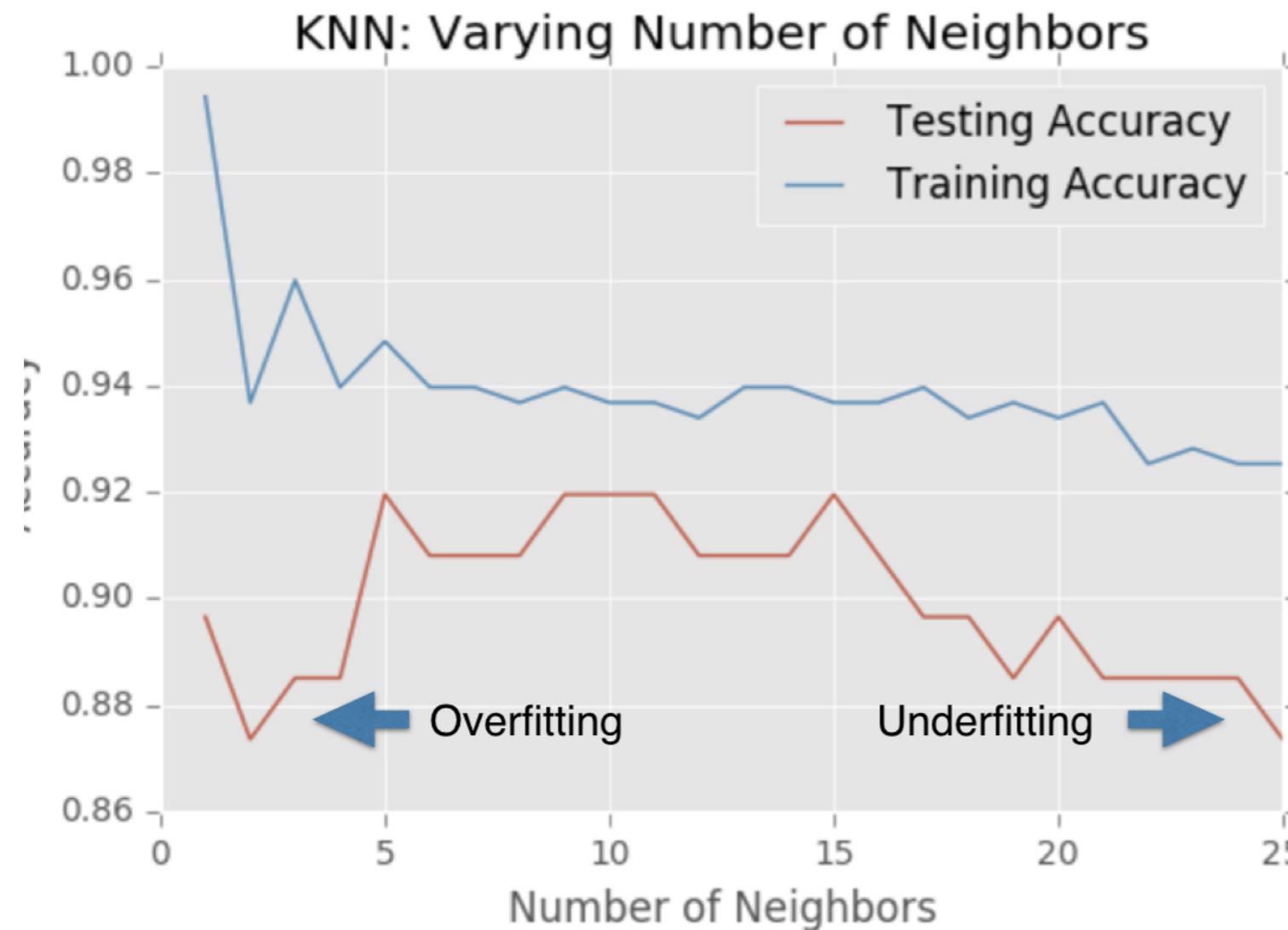


¹ Source: Andreas Müller & Sarah Guido, Introduction to Machine Learning with Python

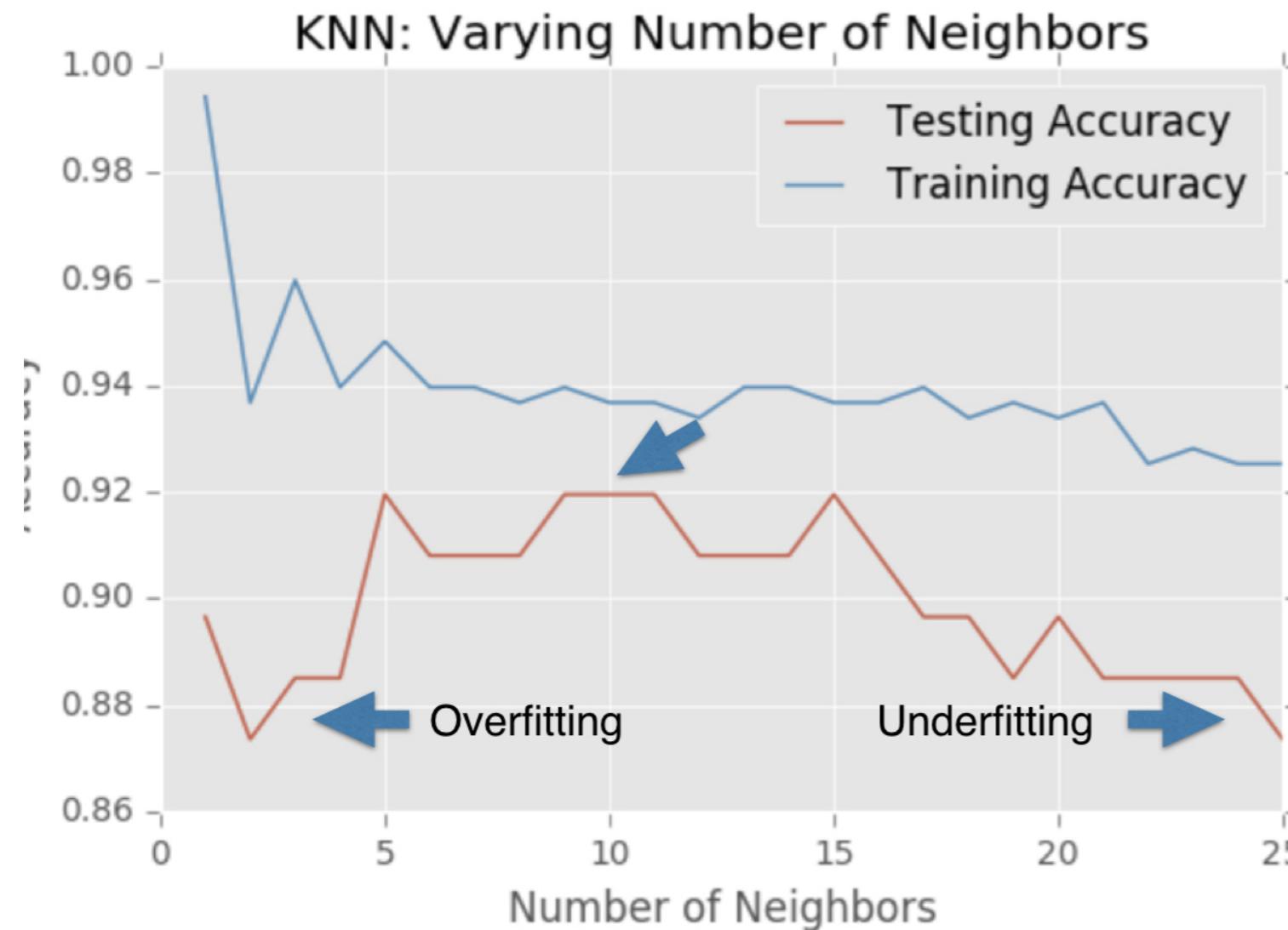
Model complexity and over/underfitting



Model complexity and over/underfitting



Model complexity and over/underfitting

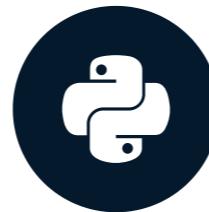


Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Introduction to regression

SUPERVISED LEARNING WITH SCIKIT-LEARN



Andreas Müller

Core developer, scikit-learn

Boston housing data

```
boston = pd.read_csv('boston.csv')
print(boston.head())
```

```
CRIM      ZN   INDUS  CHAS      NX      RM     AGE      DIS      RAD      TAX  \\
0  0.00632  18.0    2.31      0  0.538  6.575  65.2  4.0900      1  296.0
1  0.02731    0.0    7.07      0  0.469  6.421  78.9  4.9671      2  242.0
2  0.02729    0.0    7.07      0  0.469  7.185  61.1  4.9671      2  242.0
3  0.03237    0.0    2.18      0  0.458  6.998  45.8  6.0622      3  222.0
4  0.06905    0.0    2.18      0  0.458  7.147  54.2  6.0622      3  222.0
PTRATIO        B   LSTAT  MEDV
0    15.3  396.90    4.98  24.0
1    17.8  396.90    9.14  21.6
2    17.8  392.83    4.03  34.7
3    18.7  394.63    2.94  33.4
4    18.7  396.90    5.33  36.2
```

Creating feature and target arrays

```
X = boston.drop('MEDV', axis=1).values  
y = boston['MEDV'].values
```

Predicting house value from a single feature

```
X_rooms = X[:,5]  
type(X_rooms), type(y)
```

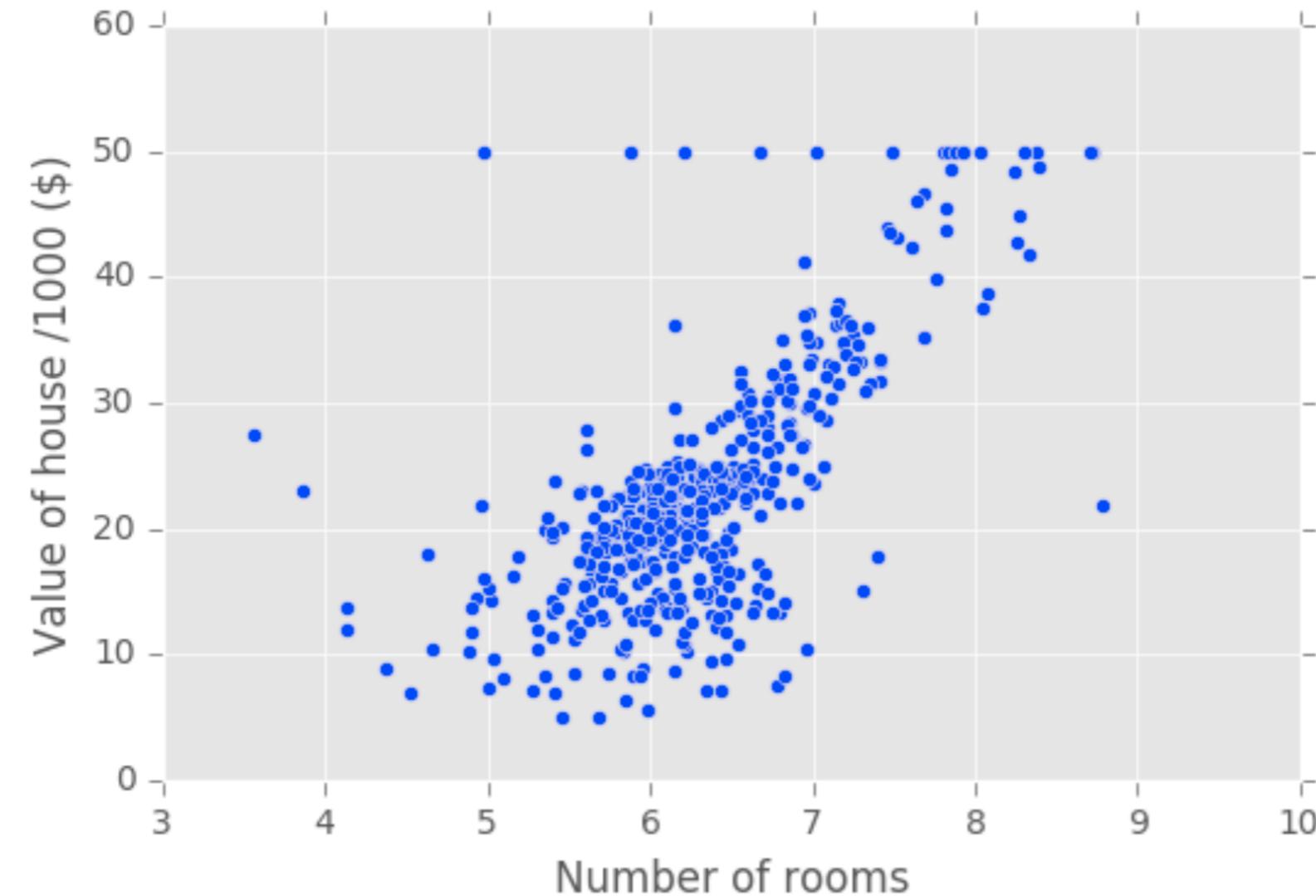
```
(numpy.ndarray, numpy.ndarray)
```

```
y = y.reshape(-1, 1)  
X_rooms = X_rooms.reshape(-1, 1)
```

Plotting house value vs. number of rooms

```
plt.scatter(X_rooms, y)  
plt.ylabel('Value of house /1000 ($)')  
plt.xlabel('Number of rooms')  
plt.show();
```

Plotting house value vs. number of rooms

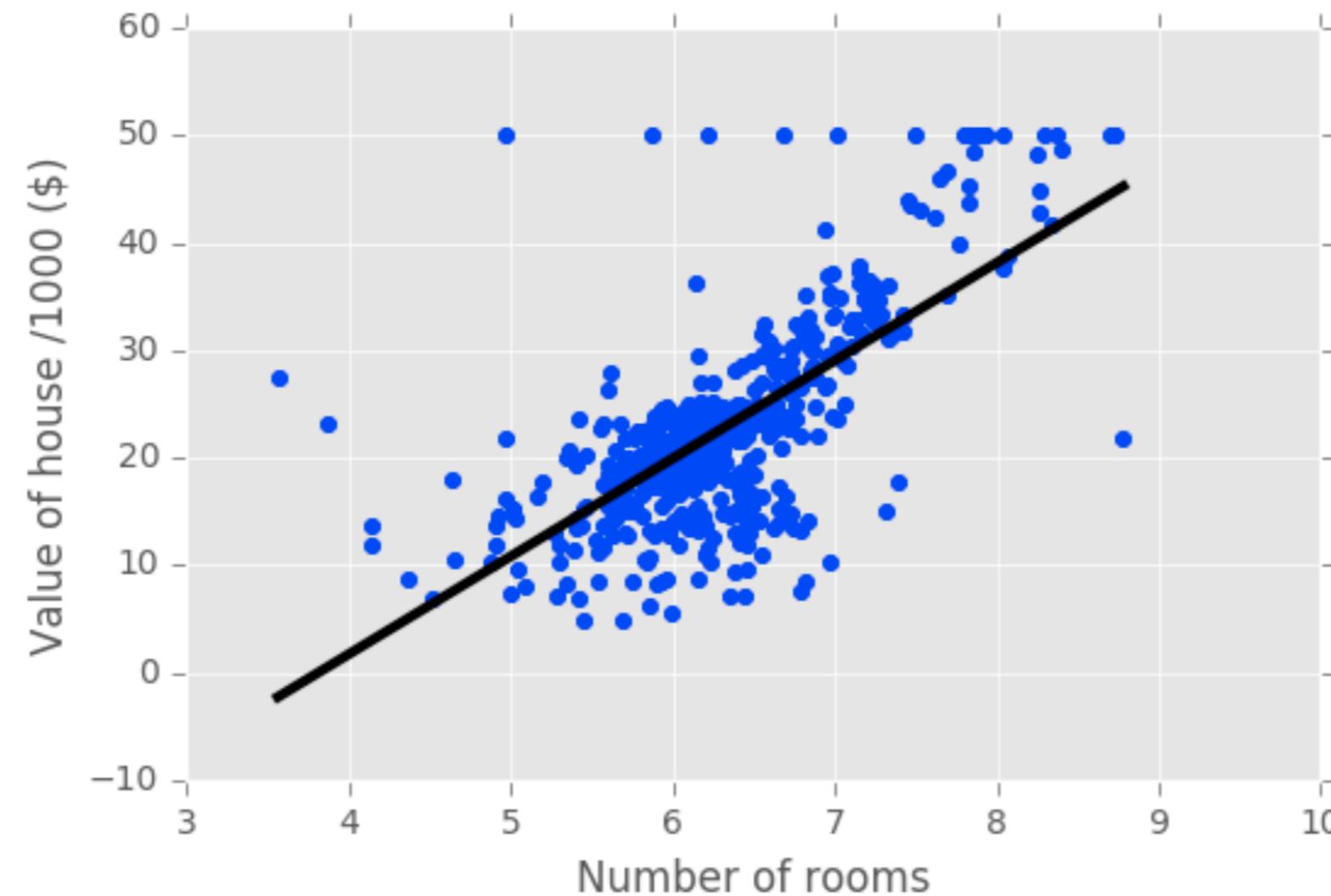


Fitting a regression model

```
import numpy as np  
from sklearn.linear_model import LinearRegression  
  
reg = LinearRegression()  
reg.fit(X_rooms, y)  
prediction_space = np.linspace(min(X_rooms),  
                               max(X_rooms)).reshape(-1, 1)
```

```
plt.scatter(X_rooms, y, color='blue')  
plt.plot(prediction_space, reg.predict(prediction_space),  
          color='black', linewidth=3)  
plt.show()
```

Fitting a regression model

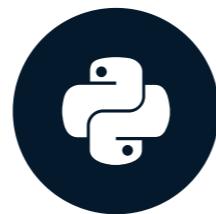


Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

The basics of linear regression

SUPERVISED LEARNING WITH SCIKIT-LEARN

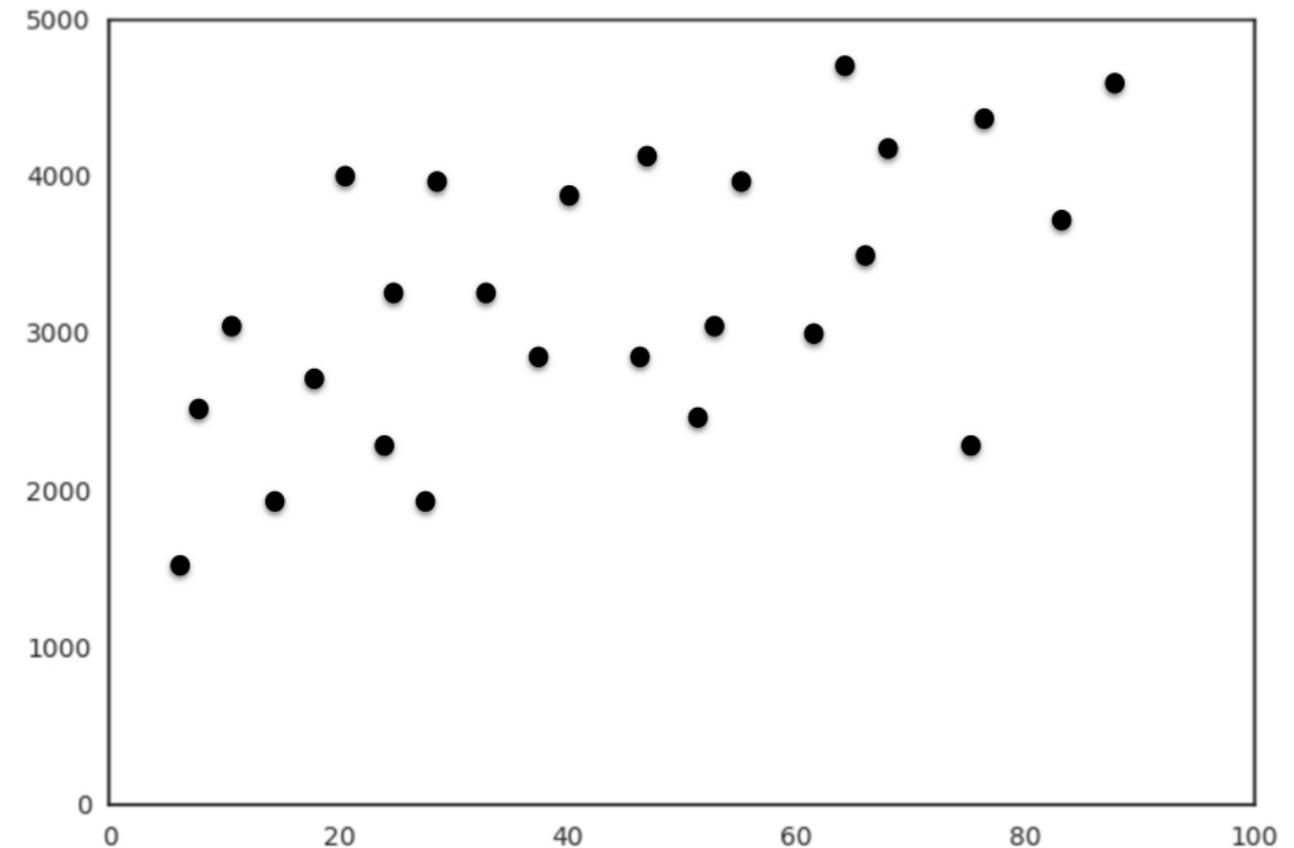


Hugo Bowne-Anderson
Data Scientist, DataCamp

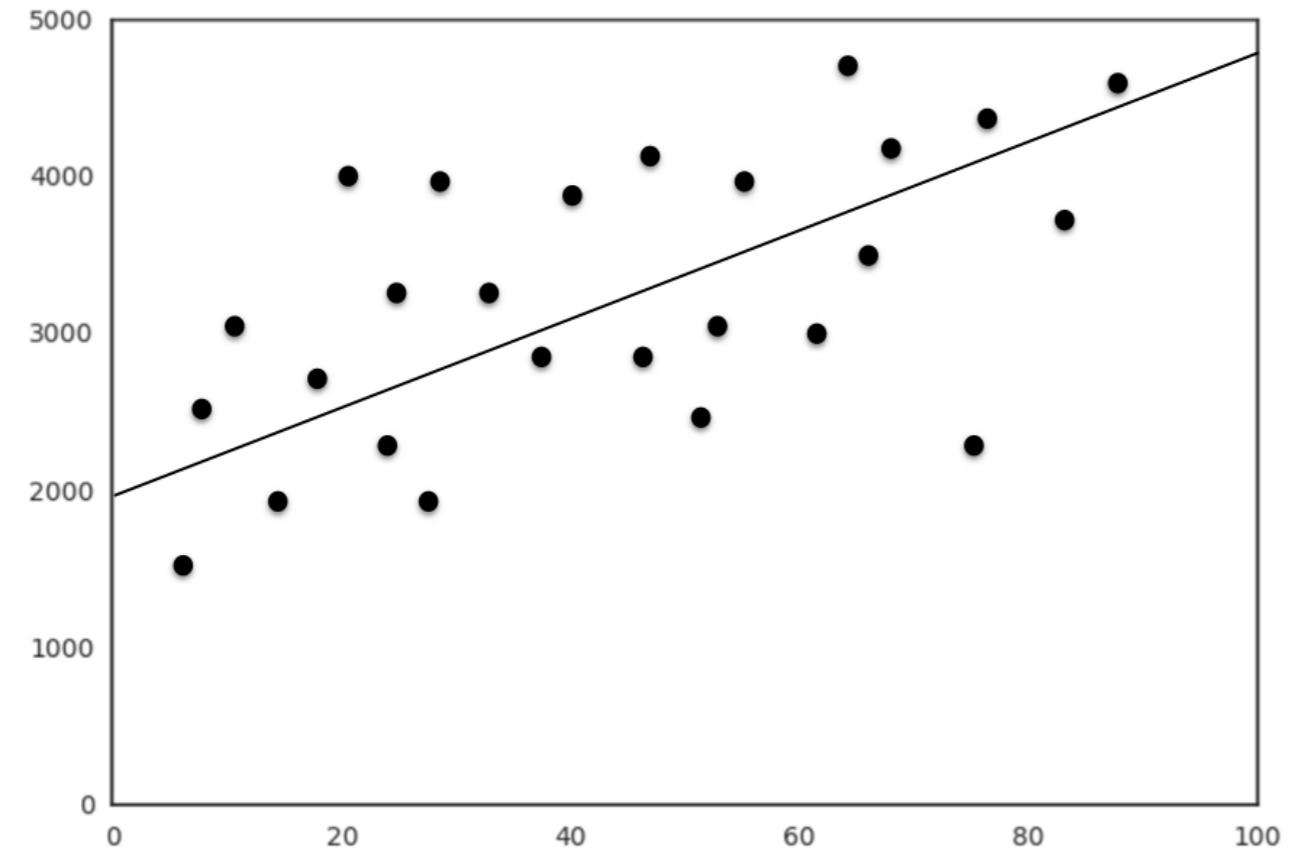
Regression mechanics

- $y = ax + b$
 - y = target
 - x = single feature
 - a, b = parameters of model
- How do we choose a and b ?
- Define an error functions for any given line
 - Choose the line that minimizes the error function

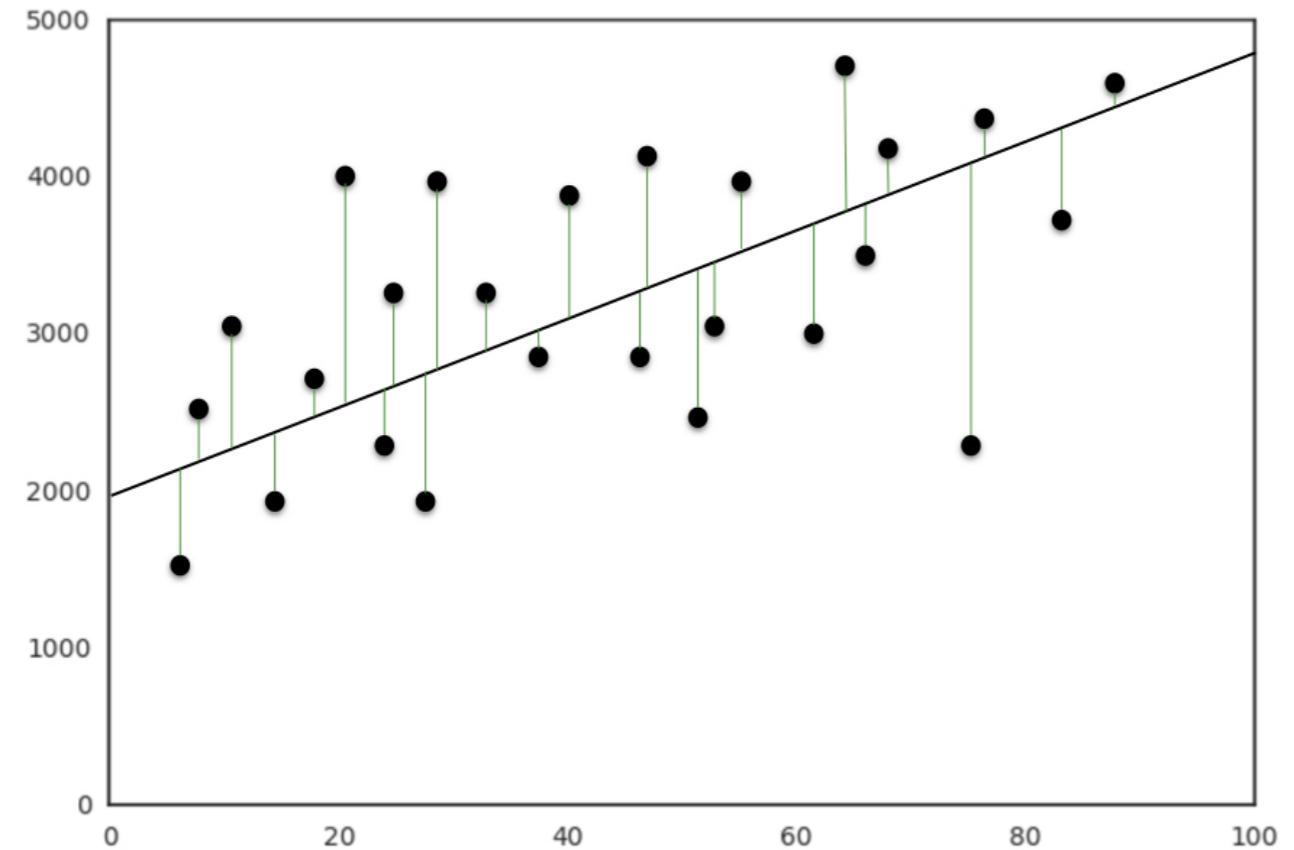
The loss function



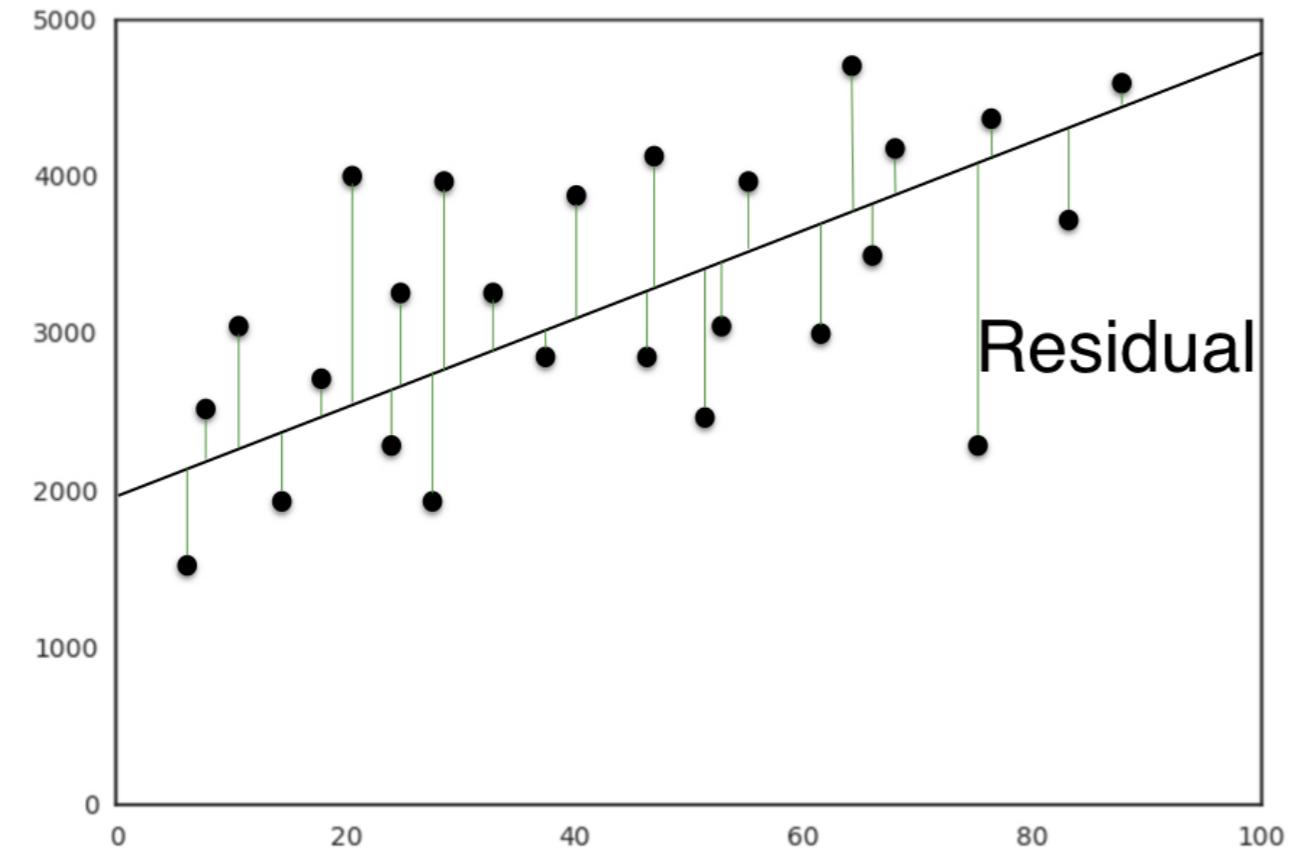
The loss function



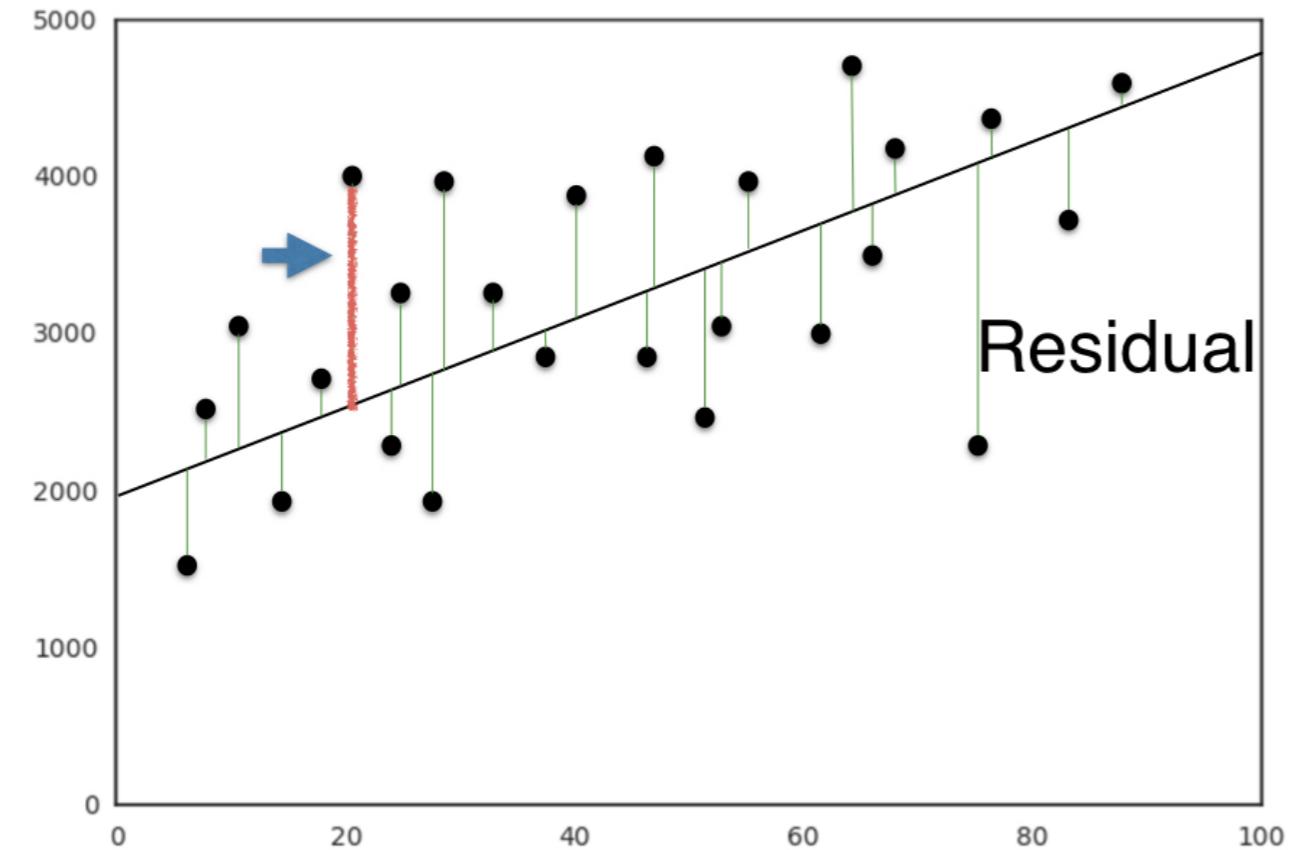
The loss function



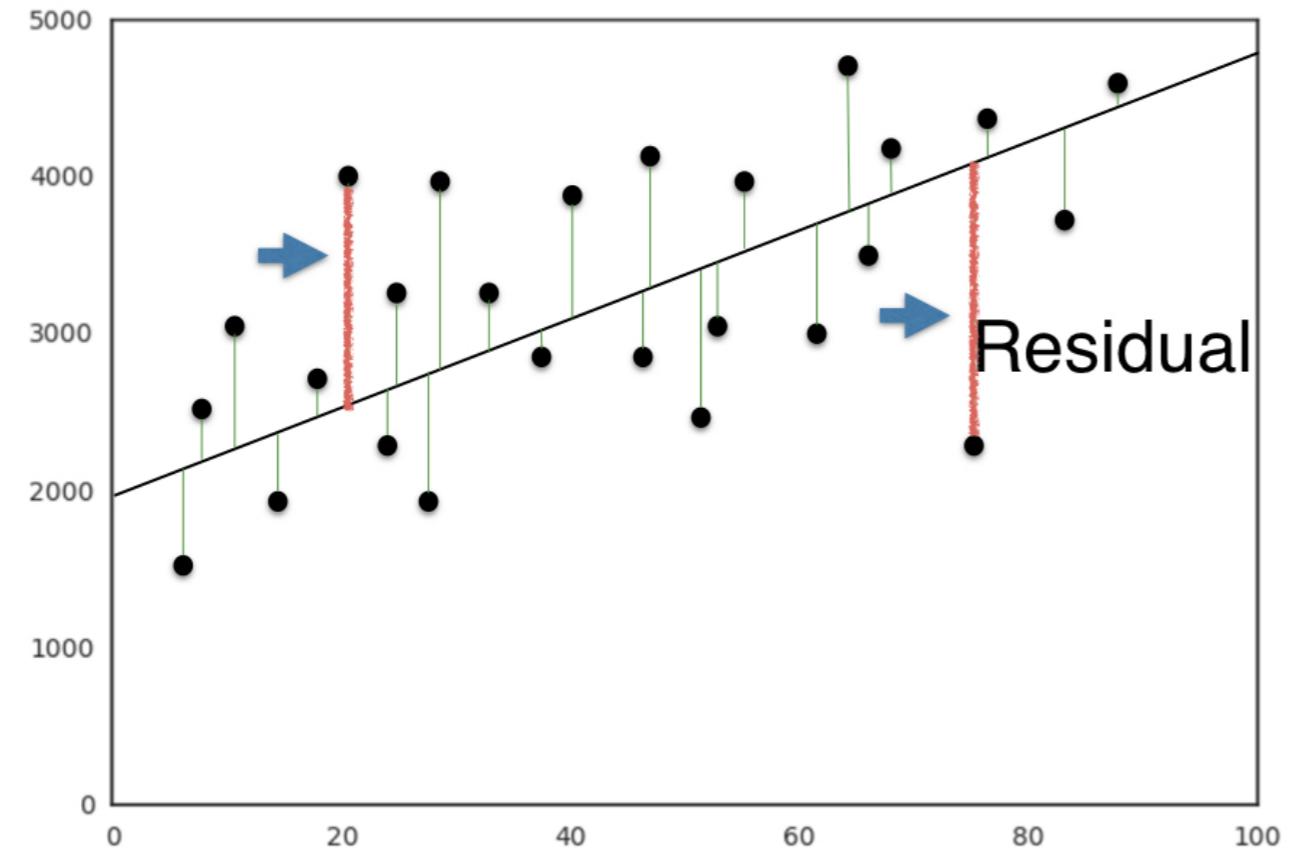
The loss function



The loss function



The loss function



Ordinary least squares(OLS): Minimize sum of squares of residuals

Linear regression in higher dimensions

$$y = a_1x_1 + a_2x_2 + b$$

- To fit a linear regression model here:
 - Need to specify 3 variables
- In higher dimensions:
 - Must specify coefficient for each feature and the variable
b

$$y = a_1x_1 + a_2x_2 + a_3x_3 + \dots + a_nx_n + b$$

- Scikit-learn API works exactly the same way:
 - Pass two arrays: Features, and target

Linear regression on all features

```
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import LinearRegression  
  
X_train, X_test, y_train, y_test = train_test_split(X, y,  
    test_size = 0.3, random_state=42)  
reg_all = LinearRegression()  
reg_all.fit(X_train, y_train)  
y_pred = reg_all.predict(X_test)  
reg_all.score(X_test, y_test)
```

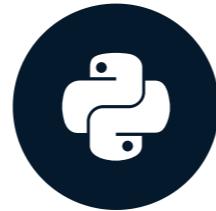
0.71122600574849526

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Cross-validation

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

Cross-validation motivation

- Model performance is dependent on way the data is split
- Not representative of the model's ability to generalize
- Solution: Cross-validation!

Cross-validation basics

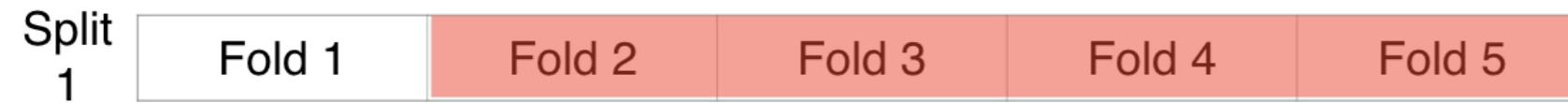
Split	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
1					

Cross-validation basics



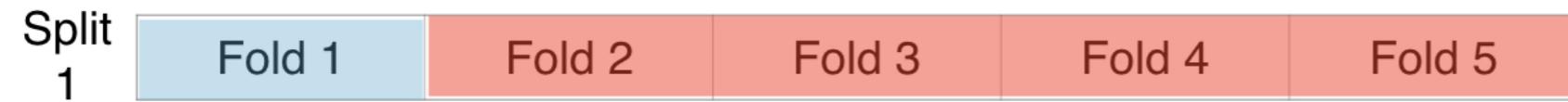
Test data

Cross-validation basics



Training data Test data

Cross-validation basics



Training data Test data

Cross-validation basics



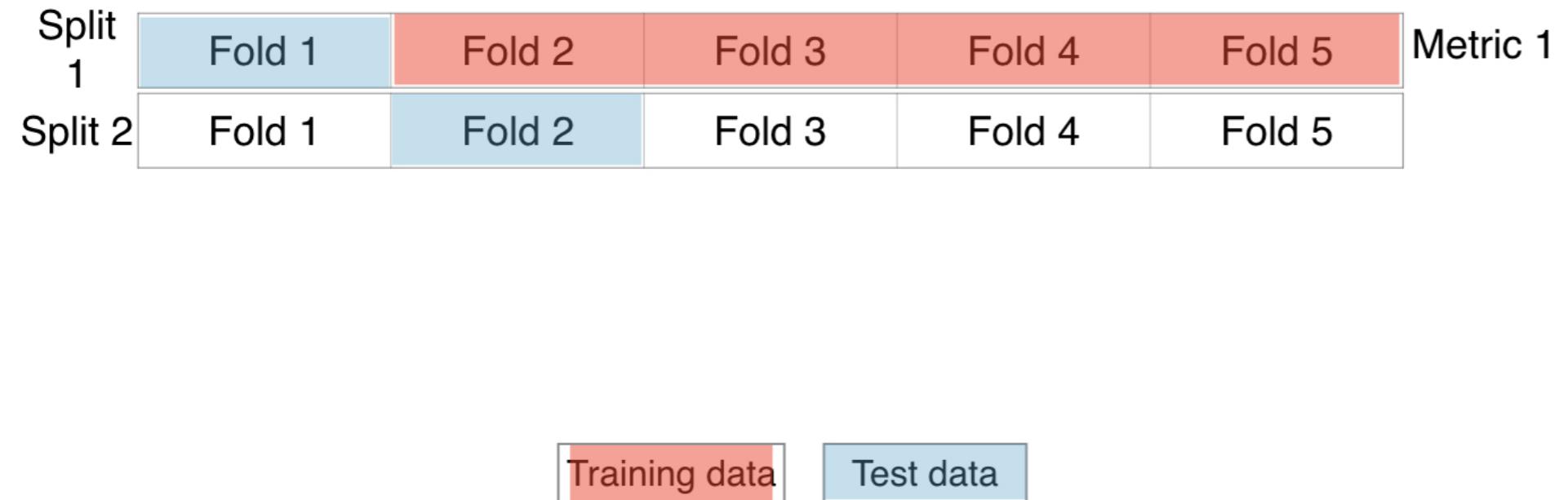
Training data Test data

Cross-validation basics

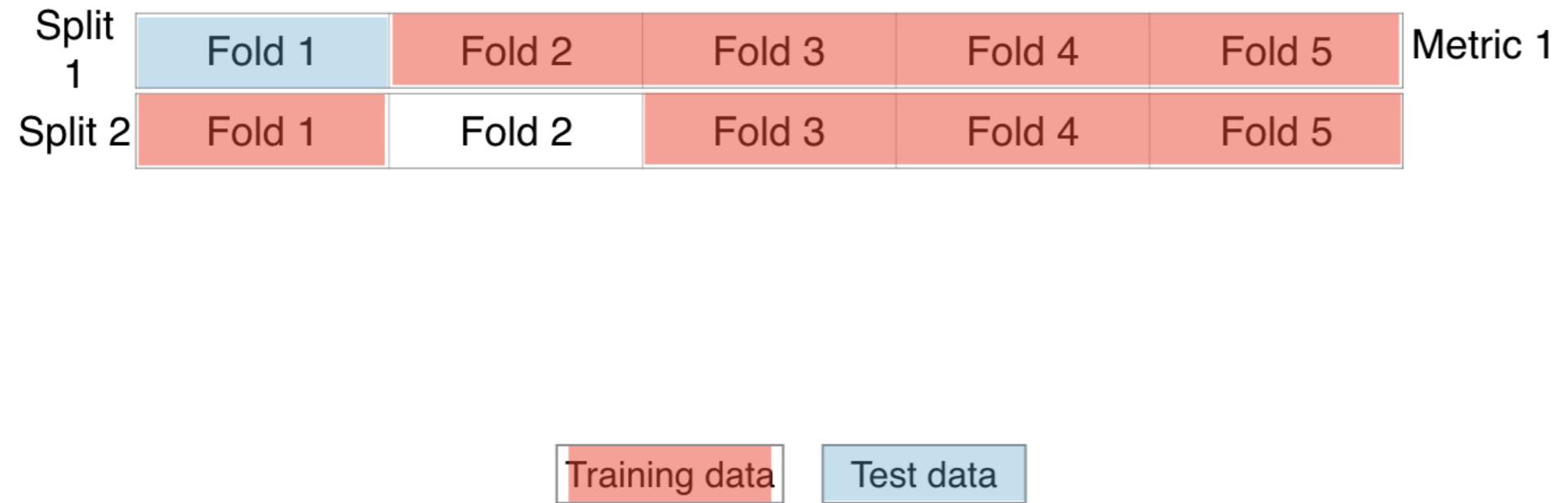
Split	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	
2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	

Training data Test data

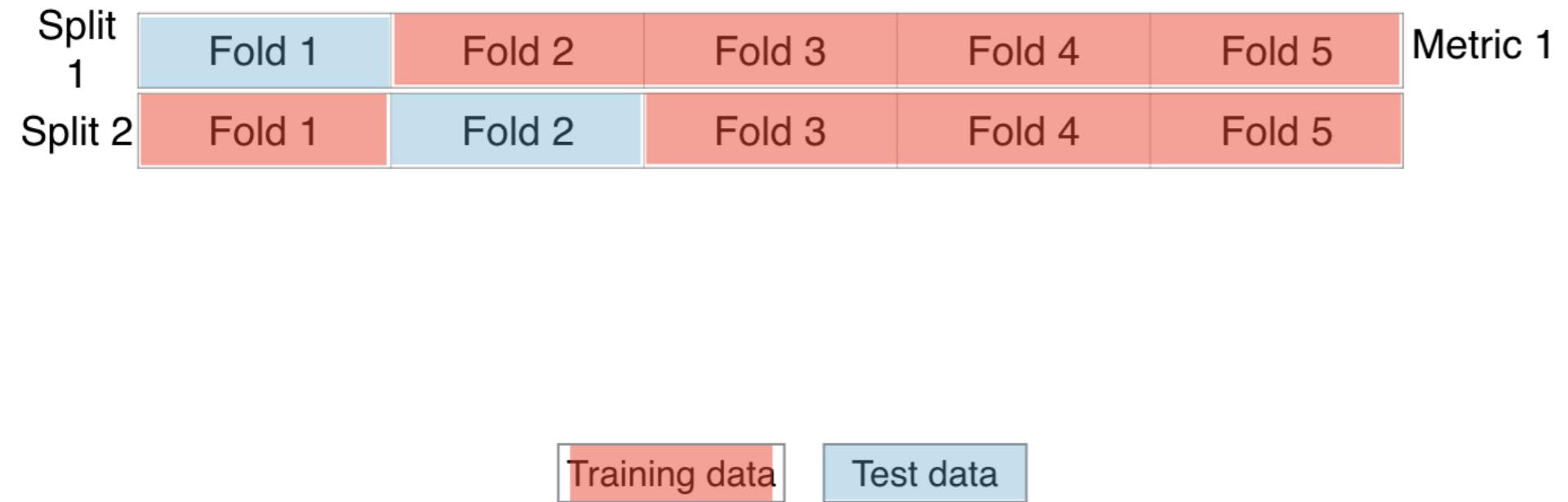
Cross-validation basics



Cross-validation basics



Cross-validation basics



Cross-validation basics

Split	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
1						
2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2

Training data Test data

Cross-validation basics

Split	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
1						
2						Metric 2
3						Metric 3

Training data Test data

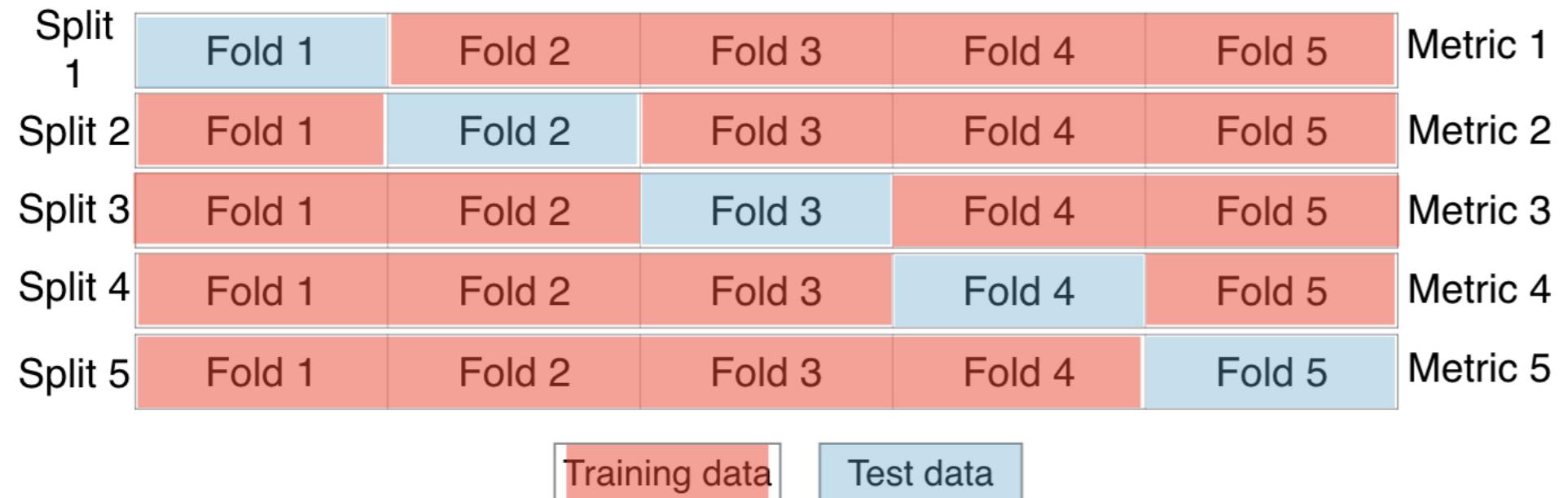
Cross-validation basics

Split	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric
1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 1
2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 2
3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 3
4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Metric 4

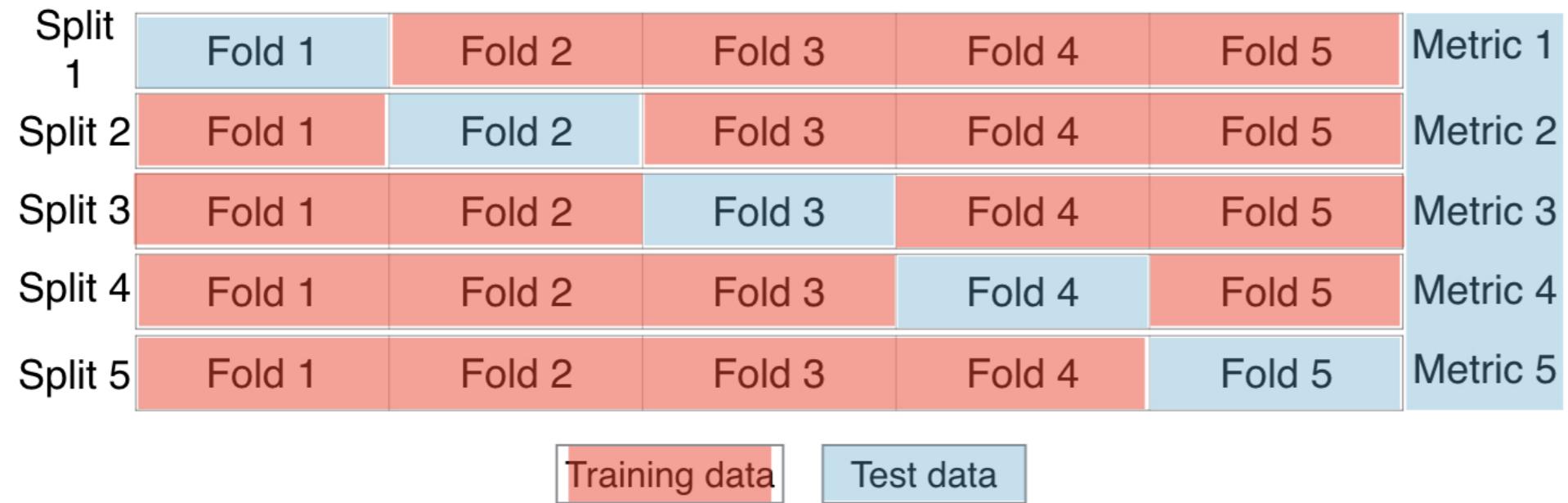
Training data

Test data

Cross-validation basics



Cross-validation basics



Cross-validation and model performance

- 5 folds = 5-fold CV
- 10 folds = 10-fold CV
- k folds = k -fold CV
- More folds = More computationally expensive

Cross-validation in scikit-learn

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=5)
print(cv_results)
```

```
[ 0.63919994  0.71386698  0.58702344  0.07923081 -0.25294154]
```

```
np.mean(cv_results)
```

```
0.35327592439587058
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Regularized regression

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

Why regularize?

- Recall: Linear regression minimizes a loss function
- It chooses a coefficient for each feature variable
- Large coefficients can lead to overfitting
- Penalizing large coefficients: Regularization

Ridge regression

- Loss function = OLS loss function +

$$\alpha * \sum_{i=1}^n a_i^2$$

- Alpha: Parameter we need to choose
- Picking alpha here is similar to picking k in k-NN
- Hyperparameter tuning (More in Chapter 3)
- Alpha controls model complexity
 - Alpha = 0: We get back OLS (Can lead to overfitting)
 - Very high alpha: Can lead to underfitting

Ridge regression in scikit-learn

```
from sklearn.linear_model import Ridge  
X_train, X_test, y_train, y_test = train_test_split(X, y,  
    test_size = 0.3, random_state=42)  
ridge = Ridge(alpha=0.1, normalize=True)  
ridge.fit(X_train, y_train)  
ridge_pred = ridge.predict(X_test)  
ridge.score(X_test, y_test)
```

0.69969382751273179

Lasso regression

- Loss function = OLS loss function +

$$\alpha * \sum_{i=1}^n |a_i|$$

Lasso regression in scikit-learn

```
from sklearn.linear_model import Lasso  
X_train, X_test, y_train, y_test = train_test_split(X, y,  
    test_size = 0.3, random_state=42)  
lasso = Lasso(alpha=0.1, normalize=True)  
lasso.fit(X_train, y_train)  
lasso_pred = lasso.predict(X_test)  
lasso.score(X_test, y_test)
```

0.59502295353285506

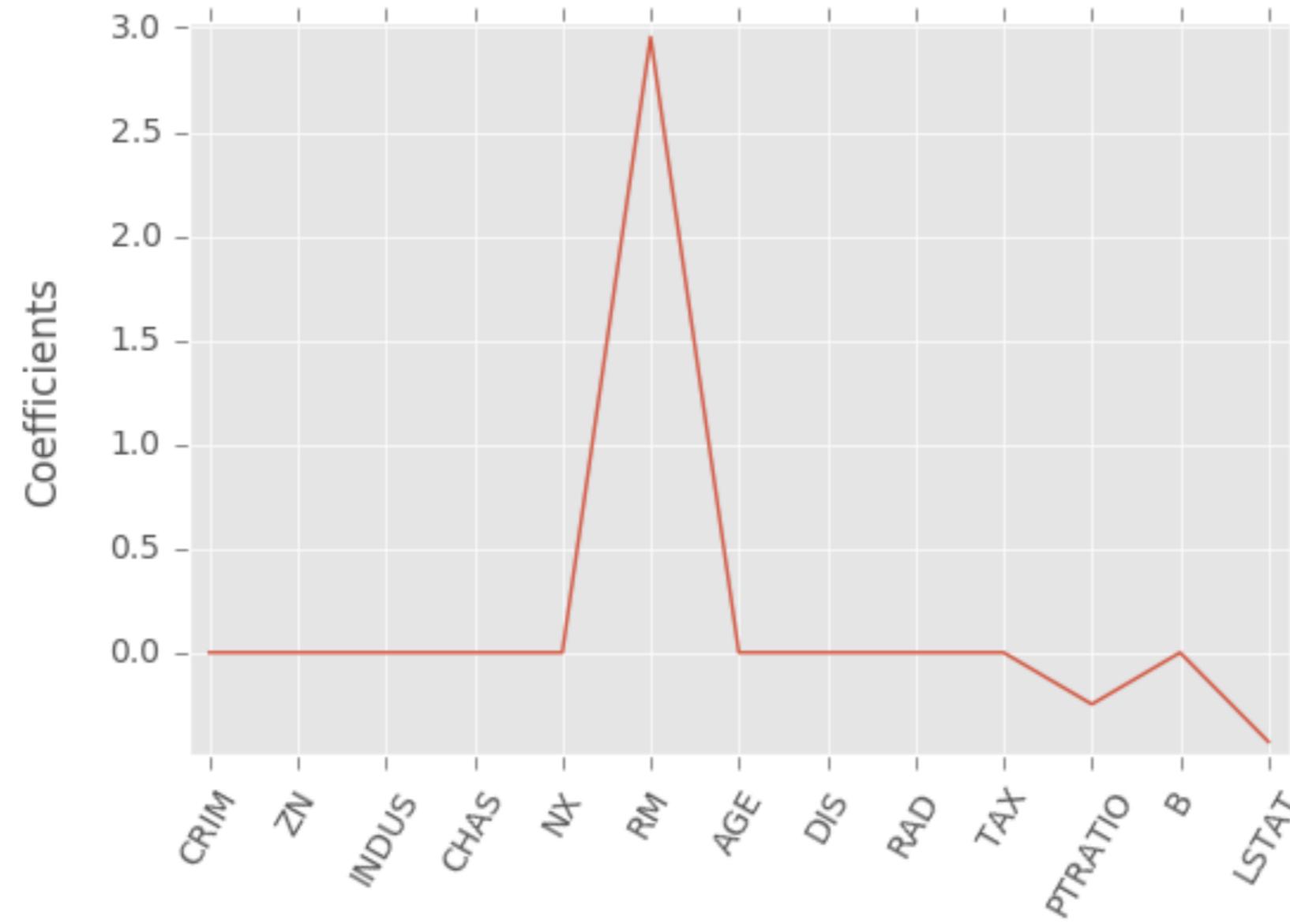
Lasso regression for feature selection

- Can be used to select important features of a dataset
- Shrinks the coefficients of less important features to exactly 0

Lasso for feature selection in scikit-learn

```
from sklearn.linear_model import Lasso  
names = boston.drop('MEDV', axis=1).columns  
lasso = Lasso(alpha=0.1)  
lasso_coef = lasso.fit(X, y).coef_  
_ = plt.plot(range(len(names)), lasso_coef)  
_ = plt.xticks(range(len(names)), names, rotation=60)  
_ = plt.ylabel('Coefficients')  
plt.show()
```

Lasso for feature selection in scikit-learn



Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

How good is your model?

SUPERVISED LEARNING WITH SCIKIT-LEARN



Andreas Müller

Core developer, scikit-learn

Classification metrics

- Measuring model performance with accuracy:
 - Fraction of correctly classified samples
 - Not always a useful metric

Class imbalance example: Emails

- Spam classification
 - 99% of emails are real; 1% of emails are spam
- Could build a classifier that predicts ALL emails as real
 - 99% accurate!
 - But horrible at actually classifying spam
 - Fails at its original purpose
- Need more nuanced metrics

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

Actual: Spam Email	
Actual: Real Email	

Predicted: Spam Email	Predicted: Real Email
True Positive	False Negative
False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

Actual: Spam Email	Predicted: Spam Email	Predicted: Real Email
Actual: Real Email	True Positive	False Negative

Predicted: Spam Email	Predicted: Real Email
True Positive	False Negative
False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative

- Accuracy:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Metrics from the confusion matrix

- Precision $\frac{tp}{tp+fp}$
- Recall $\frac{tp}{tp+tn}$
- F1score: $2 - \frac{precision*recall}{precision+recall}$
- High precision: Not many real emails predicted as spam
- High recall: Predicted most spam emails correctly

Confusion matrix in scikit-learn

```
from sklearn.metrics import classification_report  
from sklearn.metrics import confusion_matrix  
  
knn = KNeighborsClassifier(n_neighbors=8)  
X_train, X_test, y_train, y_test = train_test_split(X, y,  
    test_size=0.4, random_state=42)  
knn.fit(X_train, y_train)  
y_pred = knn.predict(X_test)
```

Confusion matrix in scikit-learn

```
print(confusion_matrix(y_test, y_pred))
```

```
[[52  7]
 [ 3 112]]
```

```
print(classification_report(y_test, y_pred))
```

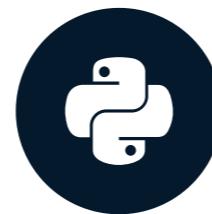
	precision	recall	f1-score	support
0	0.95	0.88	0.91	59
1	0.94	0.97	0.96	115
avg / total	0.94	0.94	0.94	174

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Logistic regression and the ROC curve

SUPERVISED LEARNING WITH SCIKIT-LEARN



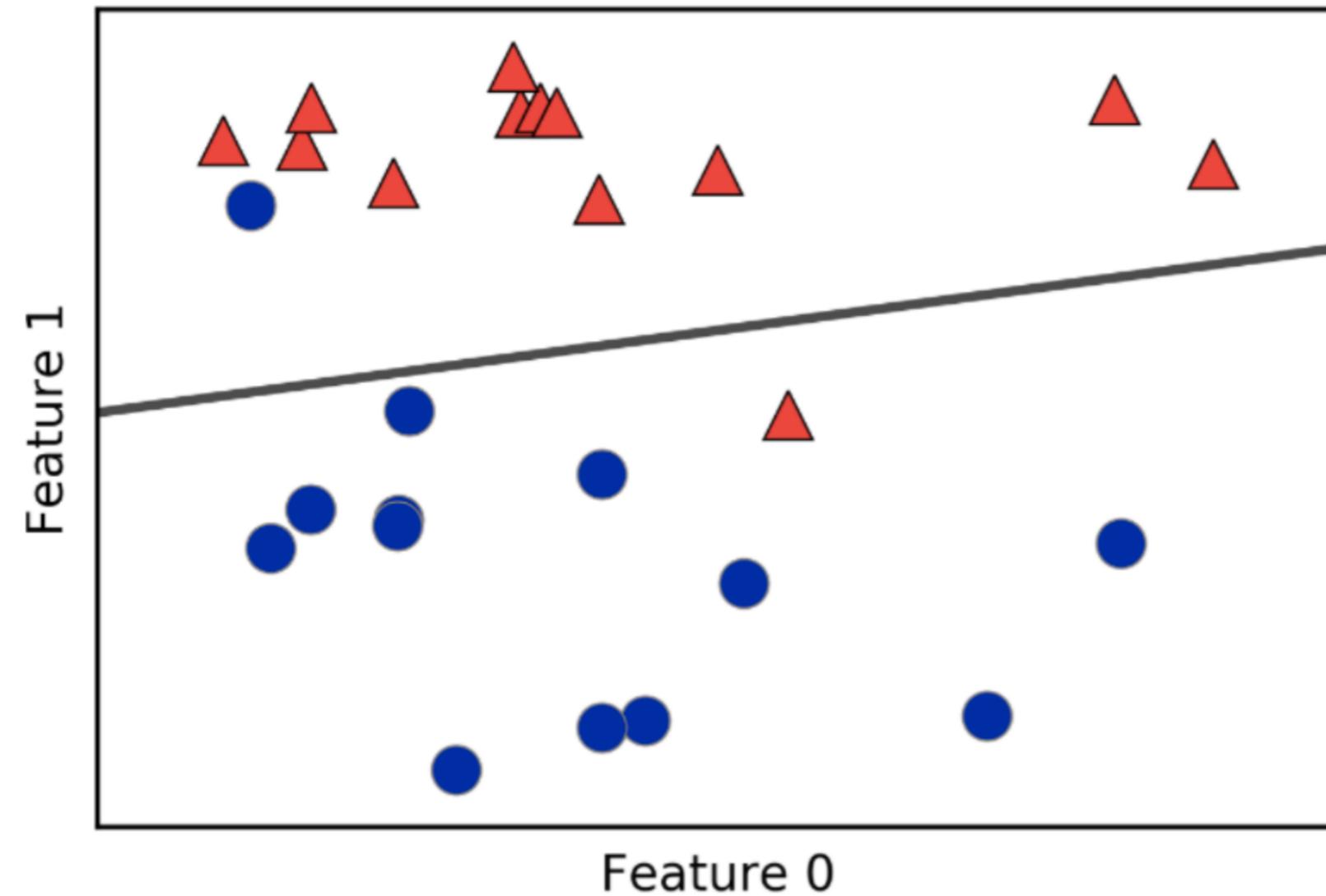
Hugo Bowne-Anderson
Data Scientist, DataCamp

Logistic regression for binary classification

- Logistic regression outputs probabilities
- If the probability ‘p’ is greater than 0.5:
 - The data is labeled ‘1’
- If the probability ‘p’ is less than 0.5:
 - The data is labeled ‘0’

Linear decision boundary

LogisticRegression



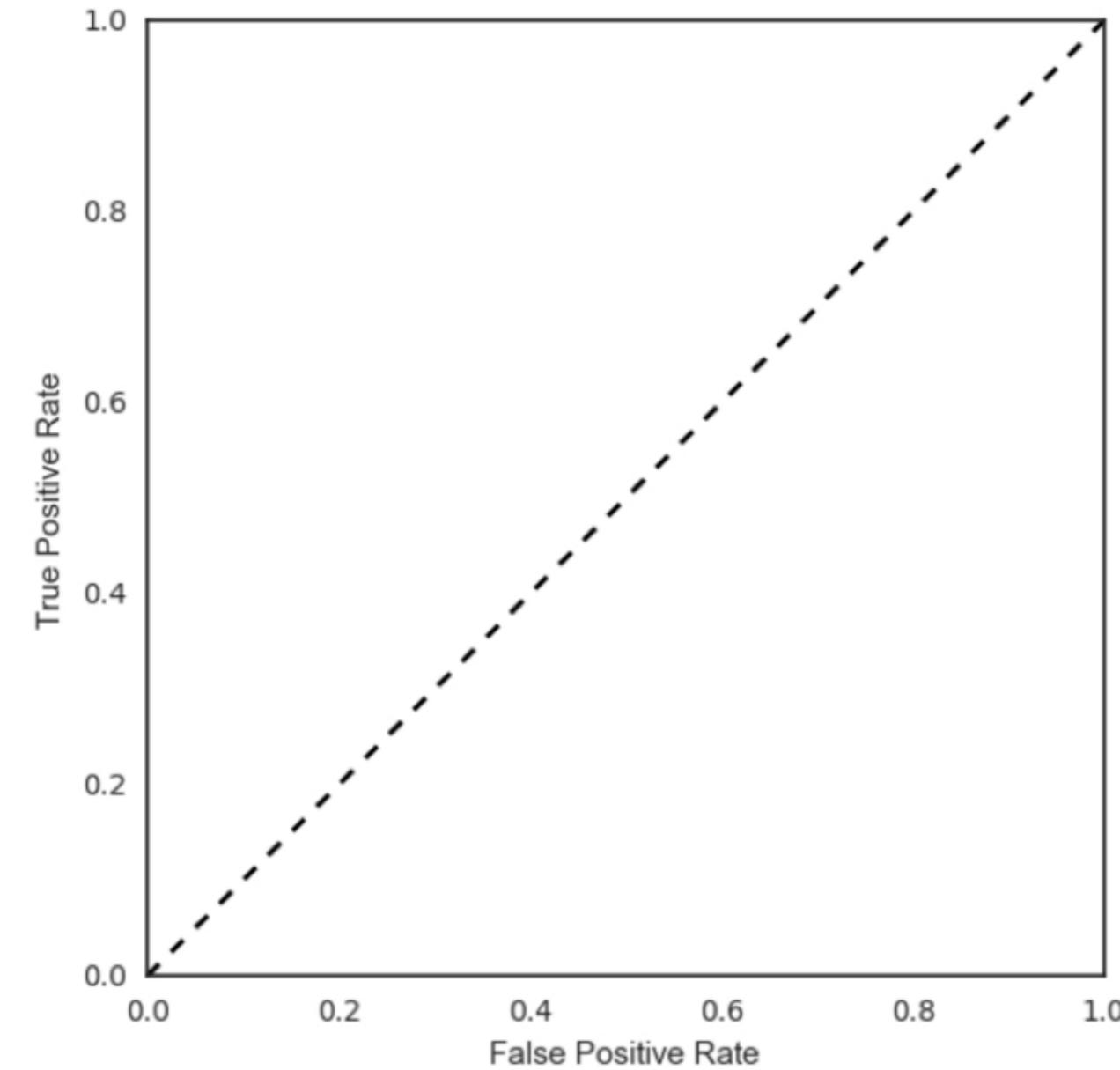
Logistic regression in scikit-learn

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

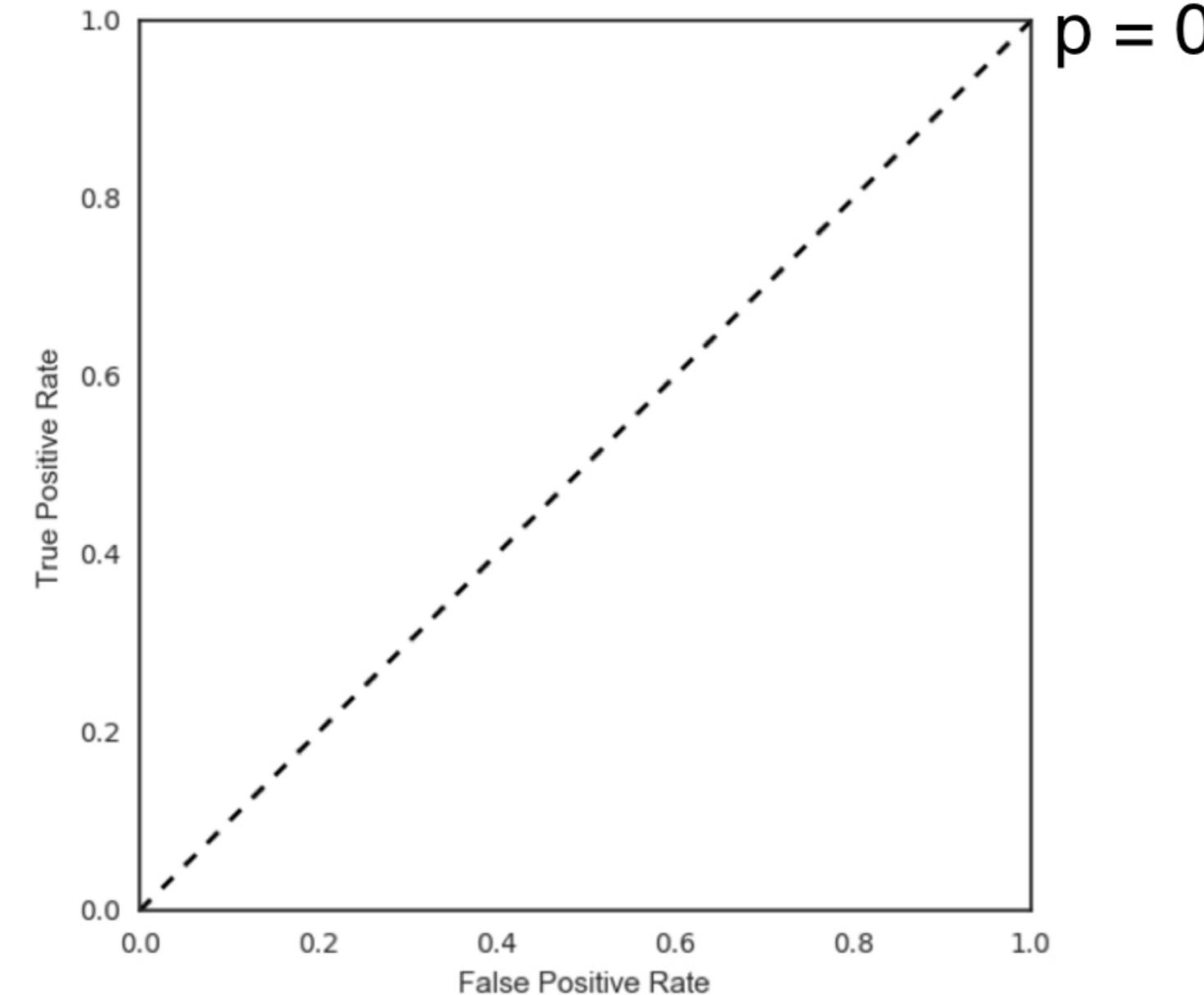
Probability thresholds

- By default, logistic regression threshold = 0.5
- Not specific to logistic regression
 - k-NN classifiers also have thresholds
- What happens if we vary the threshold?

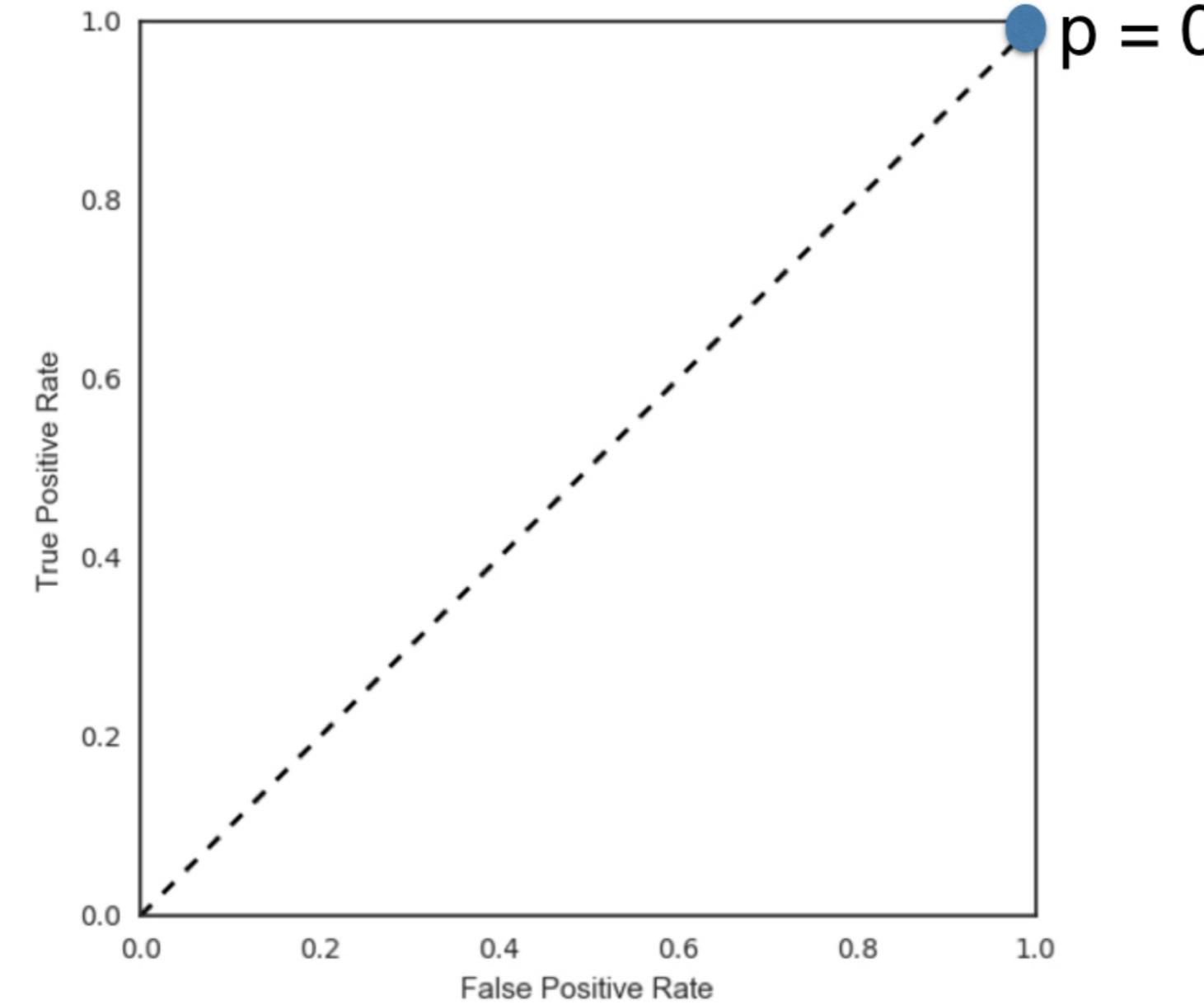
The ROC curve



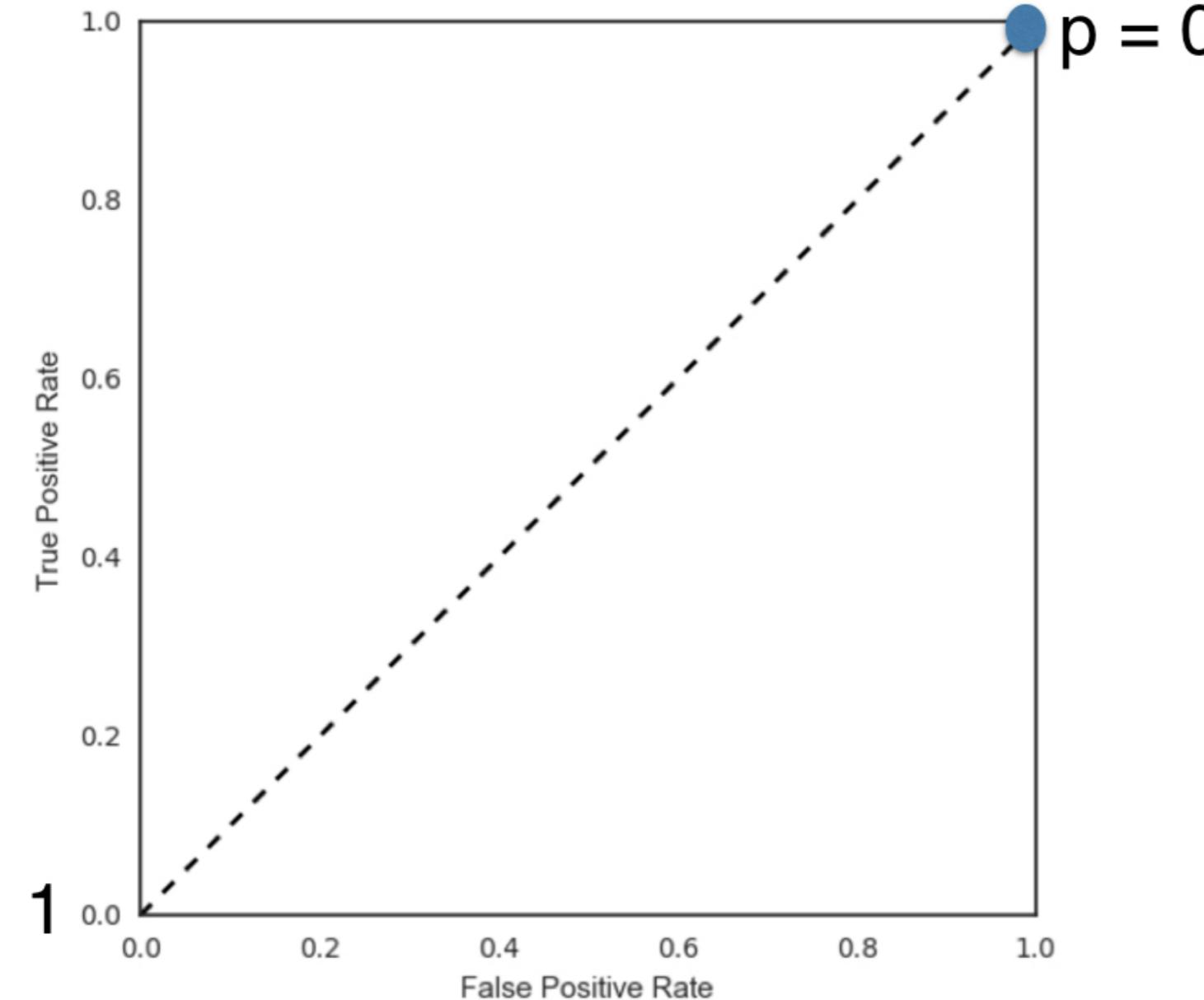
The ROC curve



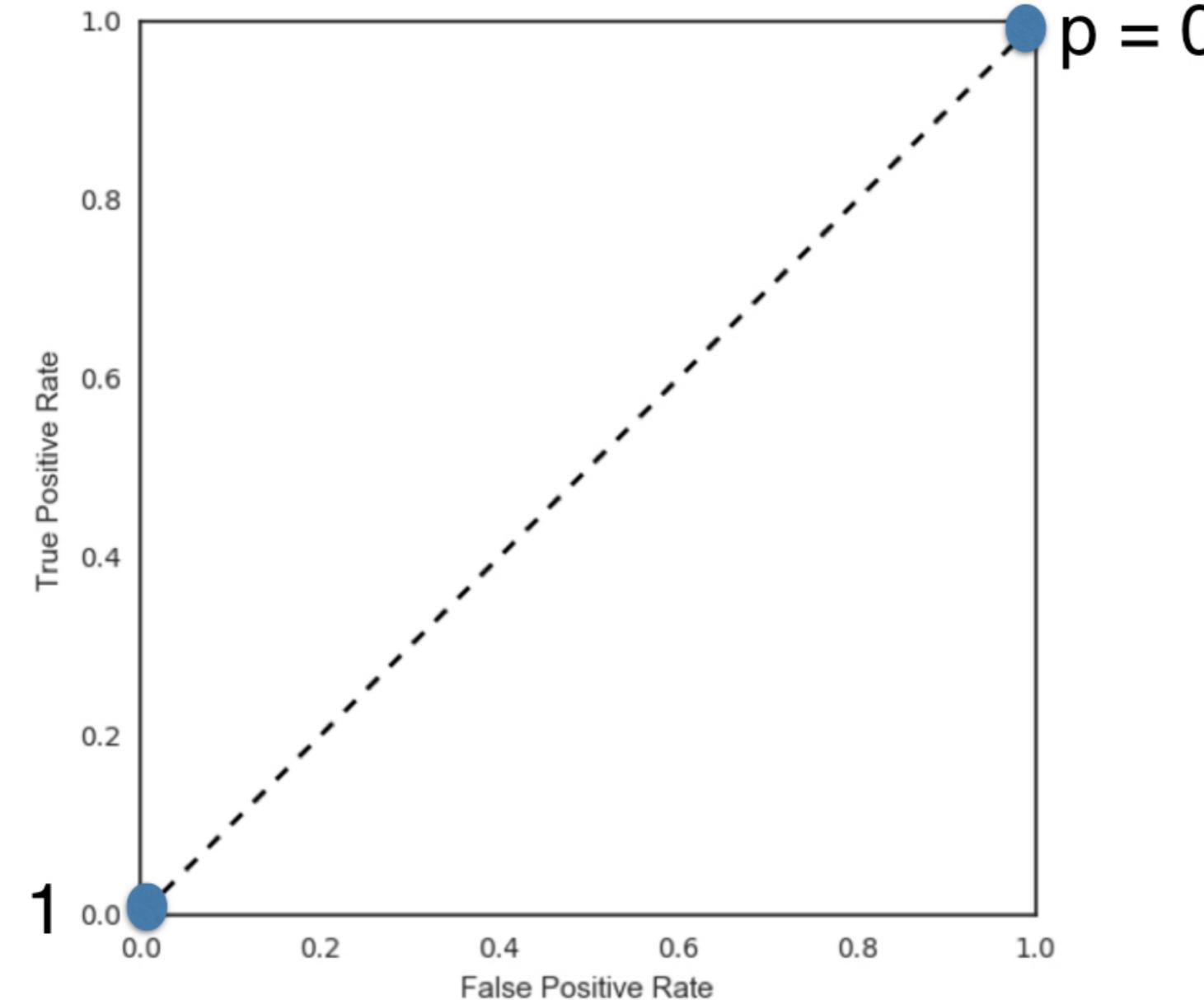
The ROC curve



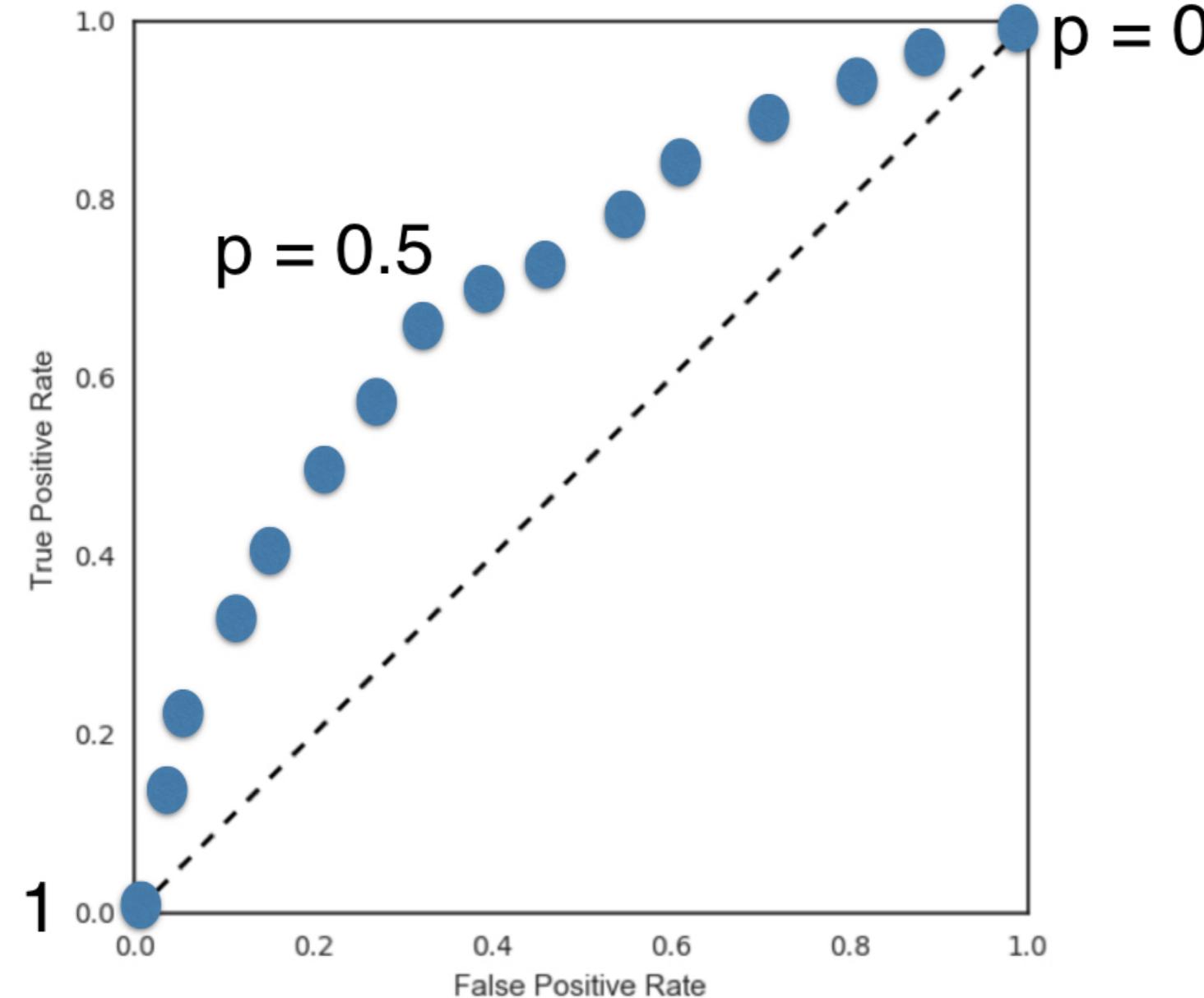
The ROC curve



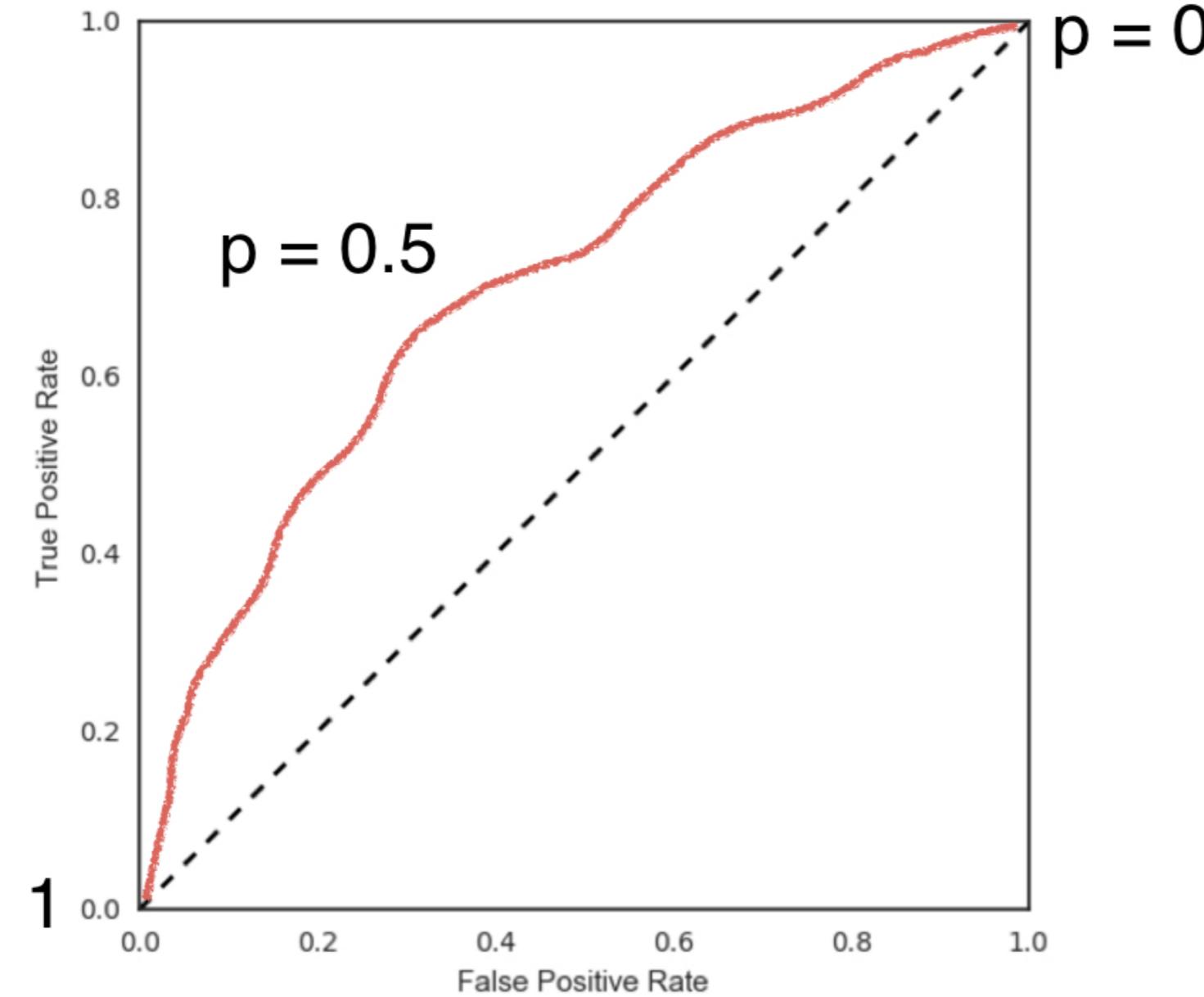
The ROC curve



The ROC curve



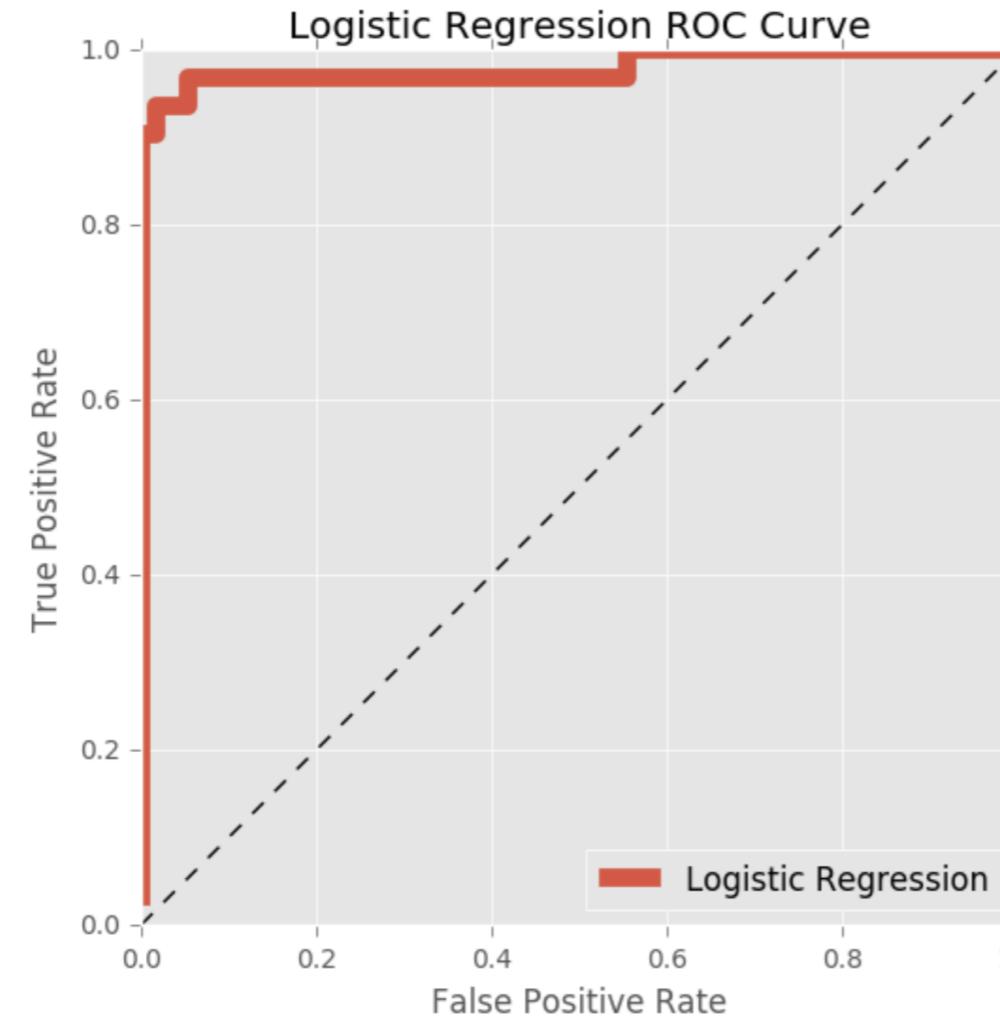
The ROC curve



Plotting the ROC curve

```
from sklearn.metrics import roc_curve  
  
y_pred_prob = logreg.predict_proba(X_test)[:,1]  
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)  
plt.plot([0, 1], [0, 1], 'k--')  
plt.plot(fpr, tpr, label='Logistic Regression')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Logistic Regression ROC Curve')  
plt.show();
```

Plotting the ROC curve



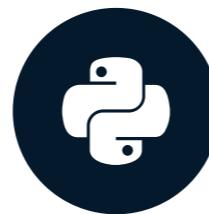
```
logreg.predict_proba(X_test)[:,1]
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Area under the ROC curve

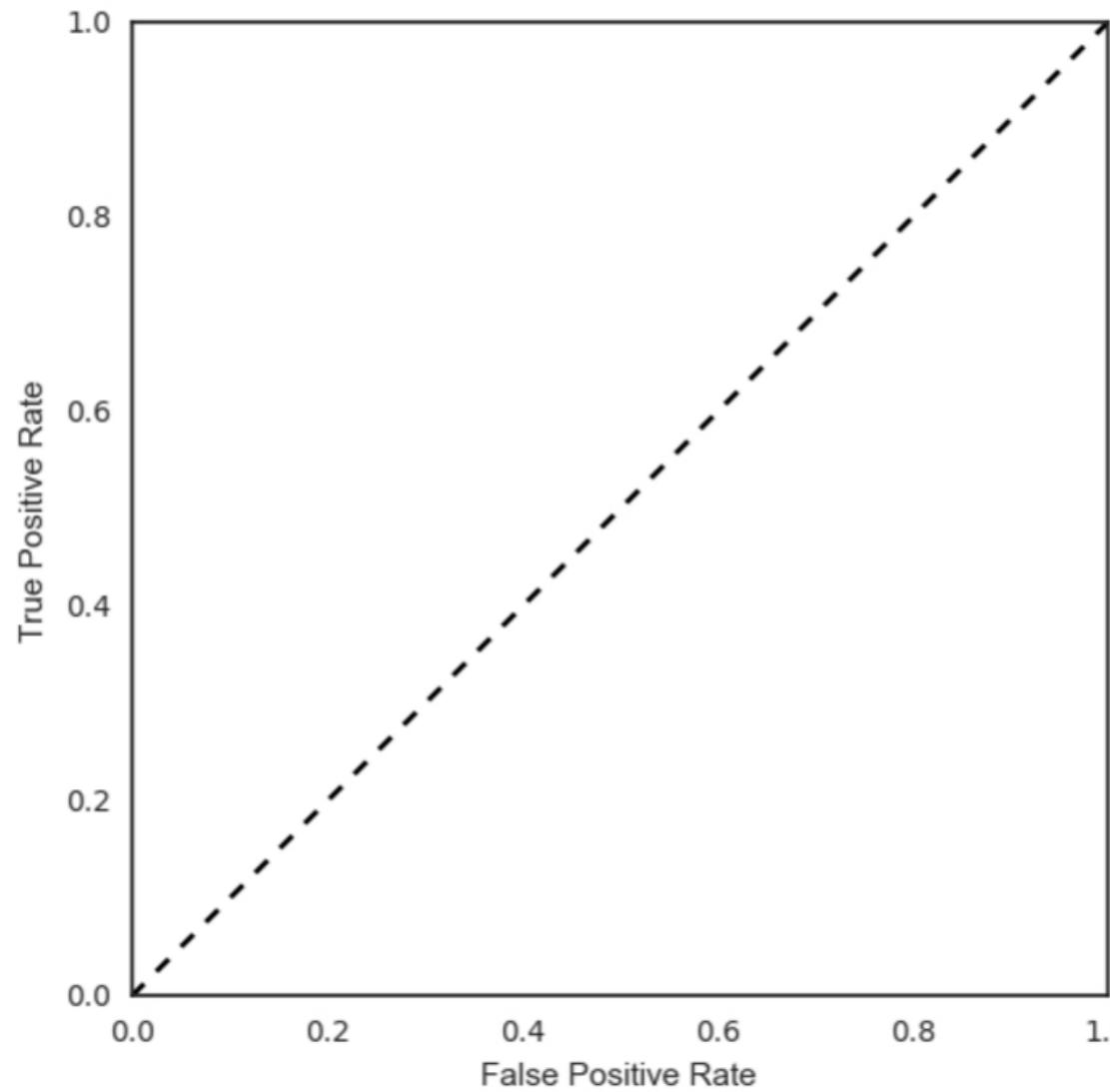
SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

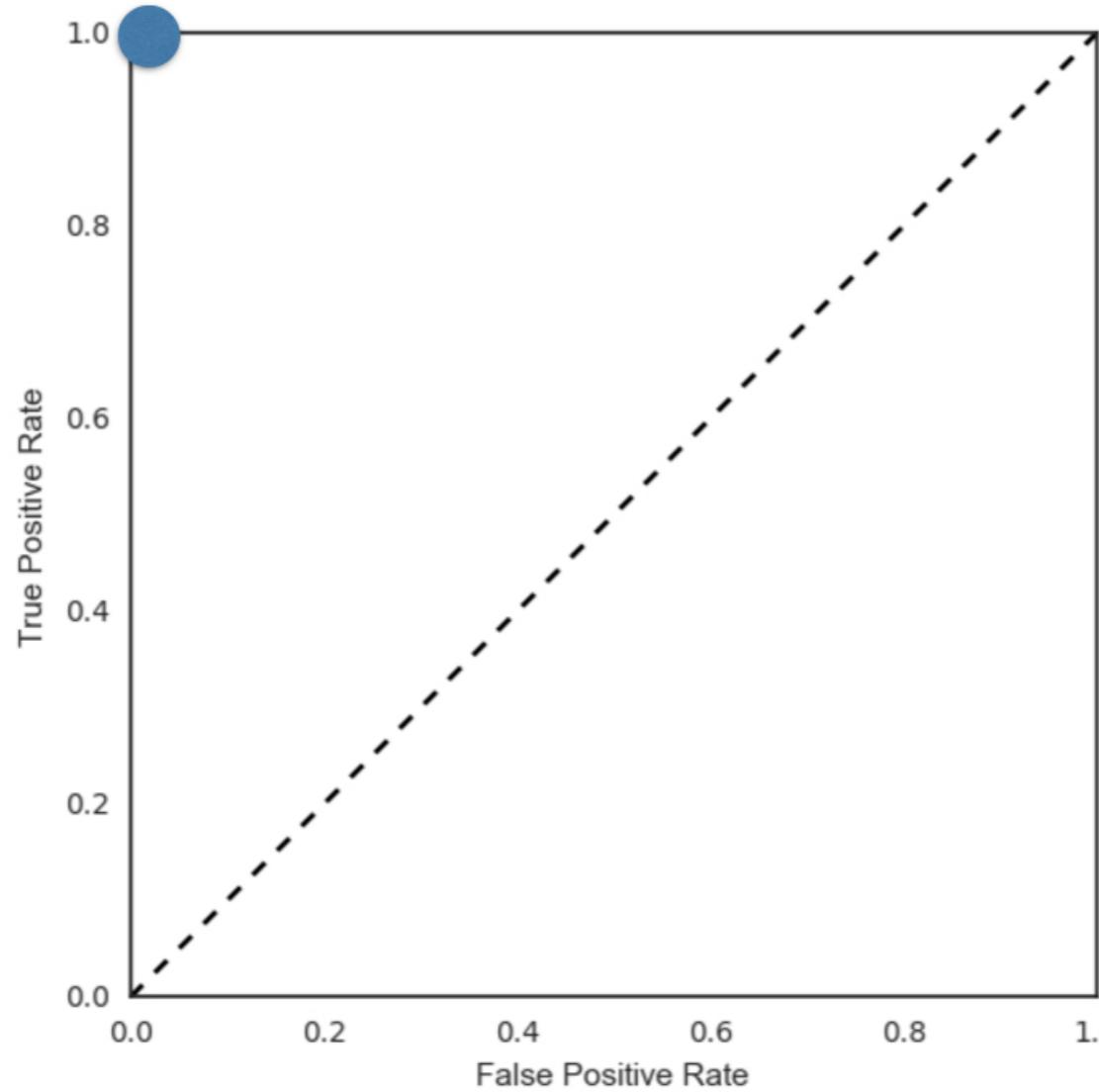
Area under the ROC curve (AUC)

- Larger area under the ROC curve = better model



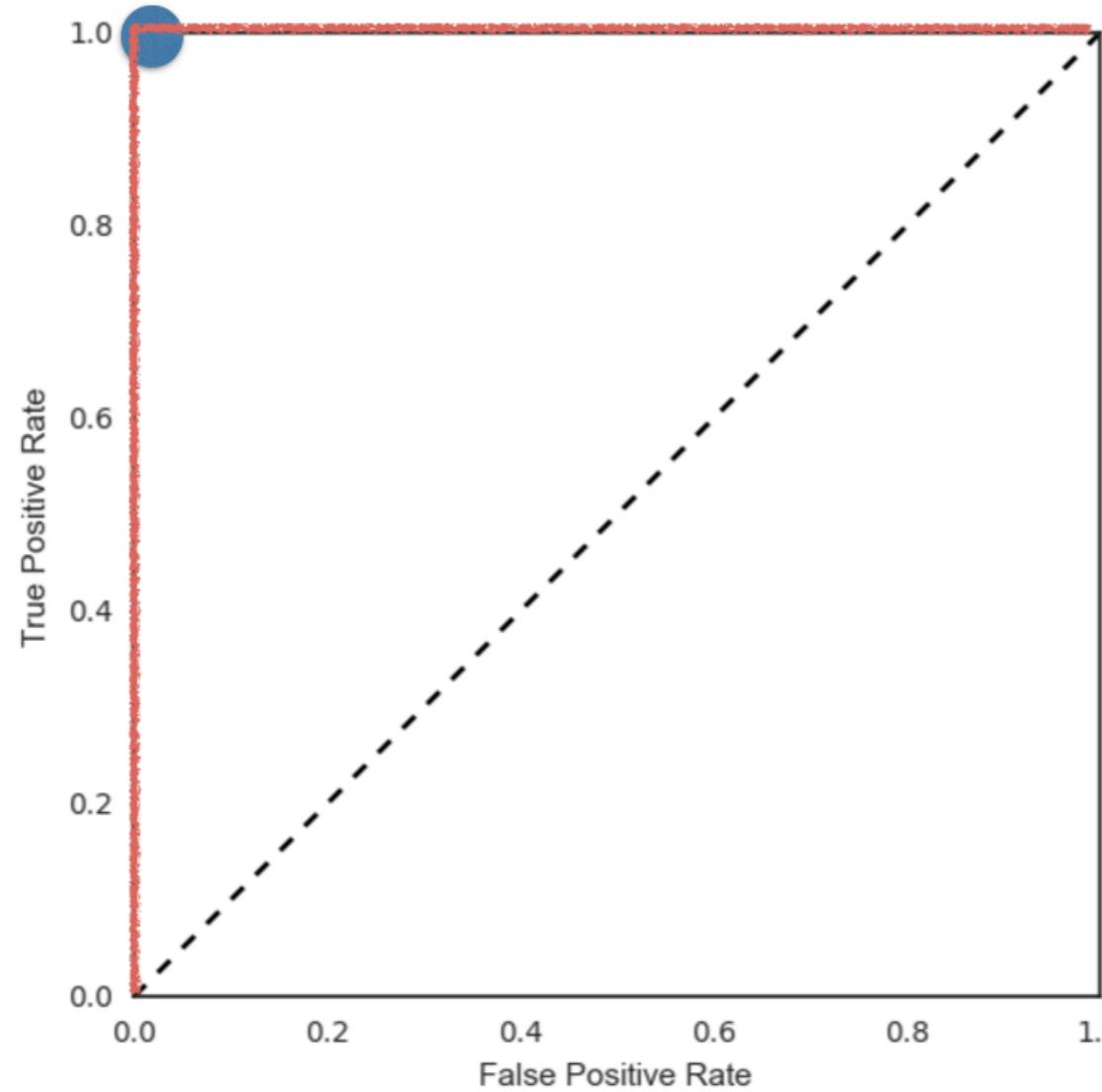
Area under the ROC curve (AUC)

- Larger area under the ROC curve = better model



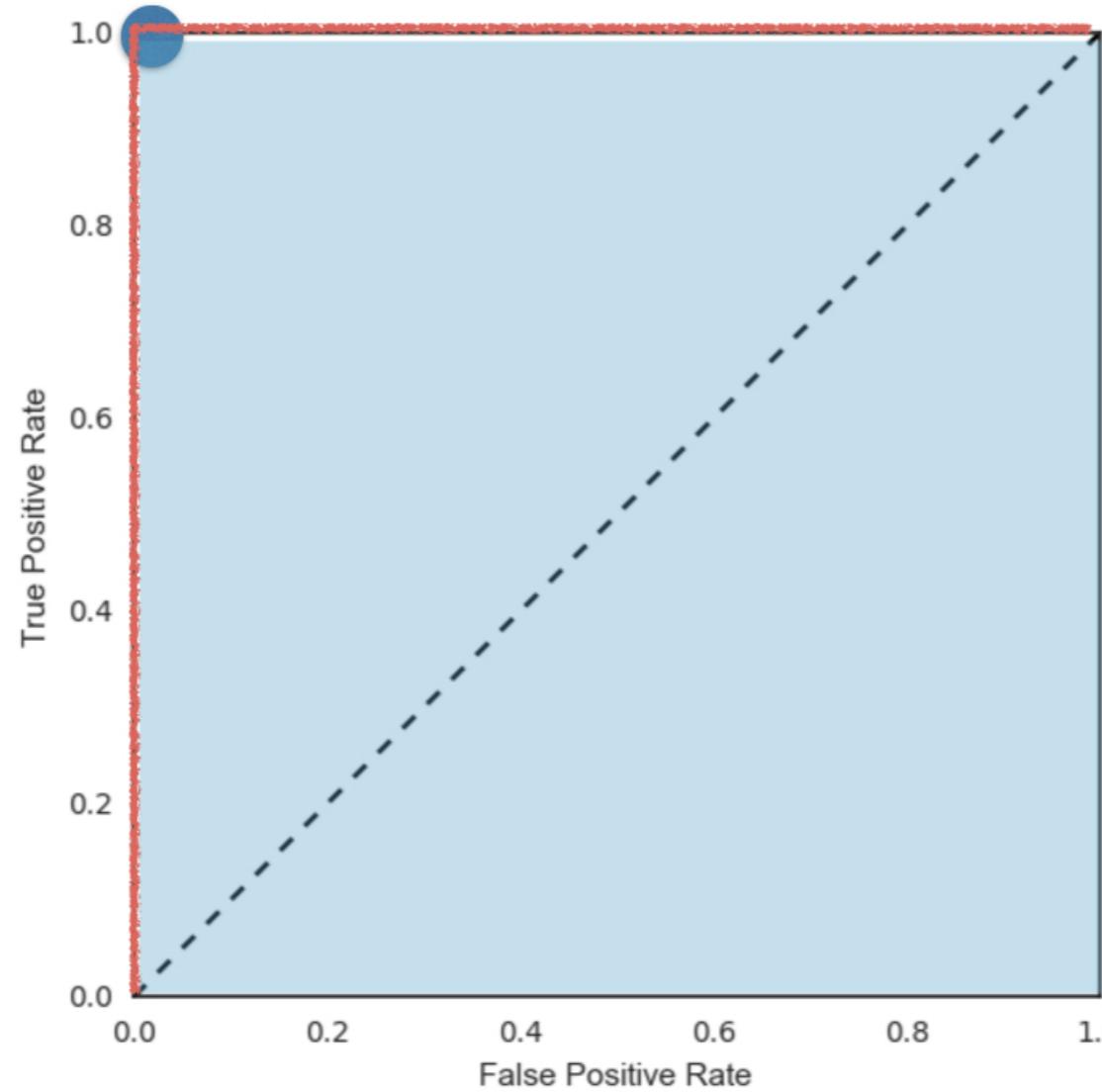
Area under the ROC curve (AUC)

- Larger area under the ROC curve = better model



Area under the ROC curve (AUC)

- Larger area under the ROC curve = better model



AUC in scikit-learn

```
from sklearn.metrics import roc_auc_score
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
logreg.fit(X_train, y_train)
y_pred_prob = logreg.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_pred_prob)
```

0.997466216216

AUC using cross-validation

```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(logreg, X, y, cv=5,
                             scoring='roc_auc')
print(cv_scores)
```

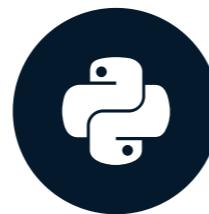
```
[ 0.99673203  0.99183007  0.99583796  1.           0.96140652]
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hyperparameter tuning

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

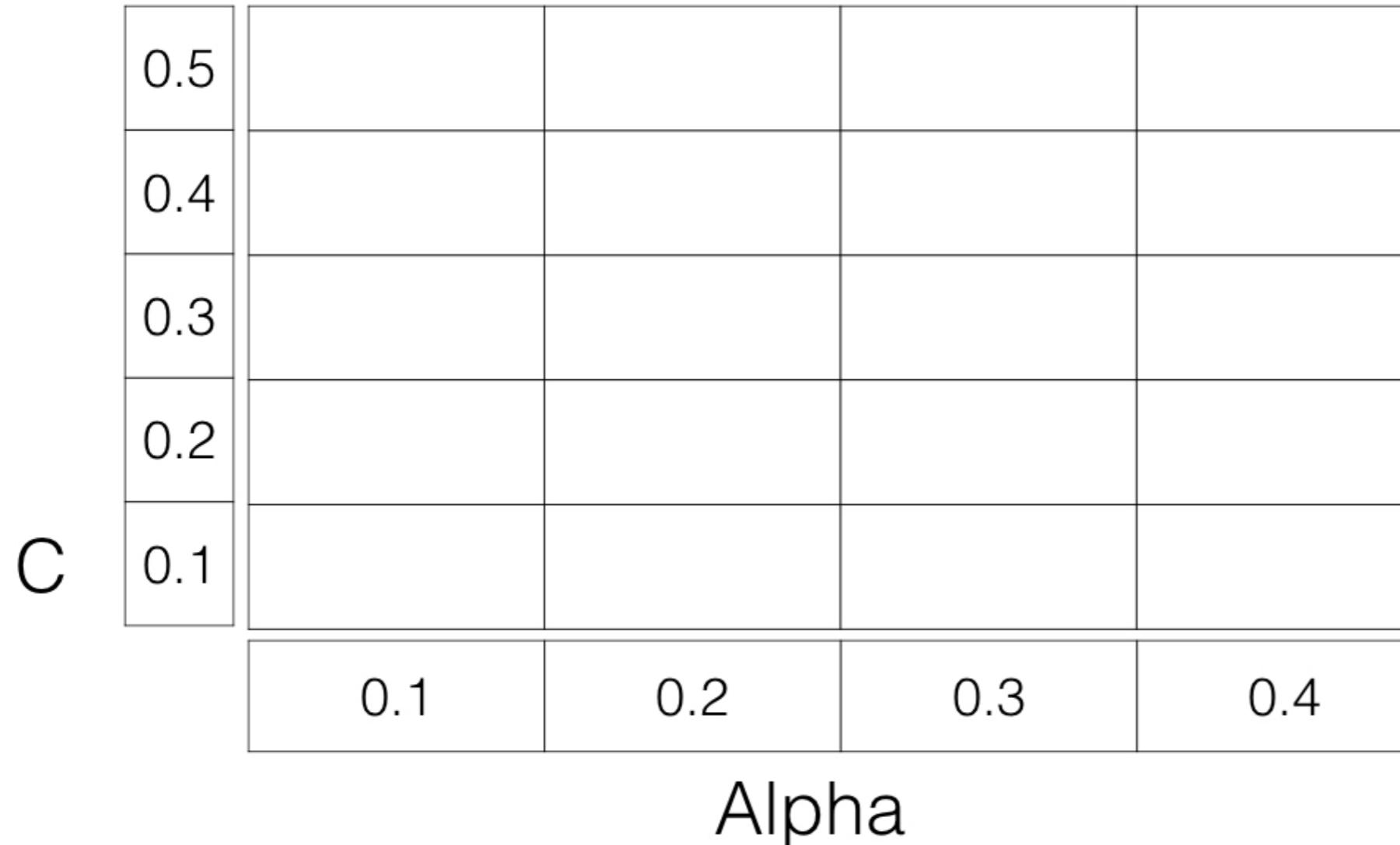
Hyperparameter tuning

- Linear regression: Choosing parameters
- Ridge/lasso regression: Choosing alpha
- k-Nearest Neighbors: Choosing n_neighbors
- Parameters like alpha and k: Hyperparameters
- Hyperparameters cannot be learned by fitting the model

Choosing the correct hyperparameter

- Try a bunch of different hyperparameter values
- Fit all of them separately
- See how well each performs
- Choose the best performing one
- It is essential to use cross-validation

Grid search cross-validation



Grid search cross-validation

0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
C	0.1	0.698	0.692	0.688
	0.1	0.2	0.3	0.4
		Alpha		

Grid search cross-validation

0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
C	0.1	0.698	0.692	0.688
	0.1	0.2	0.3	0.4
		Alpha		

GridSearchCV in scikit-learn

```
from sklearn.model_selection import GridSearchCV  
param_grid = {'n_neighbors': np.arange(1, 50)}  
knn = KNeighborsClassifier()  
knn_cv = GridSearchCV(knn, param_grid, cv=5)  
knn_cv.fit(X, y)  
knn_cv.best_params_
```

```
{'n_neighbors': 12}
```

```
knn_cv.best_score_
```

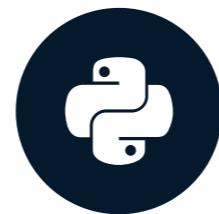
```
0.933216168717
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hold-out set for final evaluation

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

Hold-out set reasoning

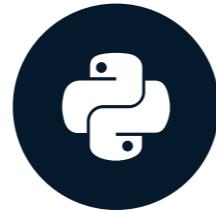
- How well can the model perform on never before seen data?
- Using ALL data for cross-validation is not ideal
- Split data into training and hold-out set at the beginning
- Perform grid search cross-validation on training set
- Choose best hyperparameters and evaluate on hold-out set

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Preprocessing data

SUPERVISED LEARNING WITH SCIKIT-LEARN



Andreas Müller

Core developer, scikit-learn

Dealing with categorical features

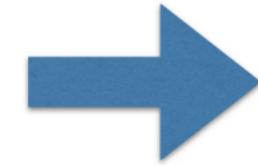
- Scikit-learn will not accept categorical features by default
- Need to encode categorical features numerically
- Convert to ‘dummy variables’
 - 0: Observation was NOT that category
 - 1: Observation was that category

Dummy variables

Origin
US
Europe
Asia

Dummy variables

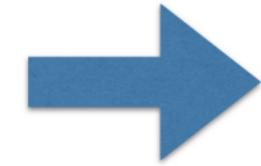
Origin
US
Europe
Asia



	origin_Asia	origin_Europe	origin_US
0	0	1	
0	1	0	
1	0	0	

Dummy variables

Origin
US
Europe
Asia



	origin_Asia	origin_US
0	1	0
0	0	1
1	0	0

Dealing with categorical features in Python

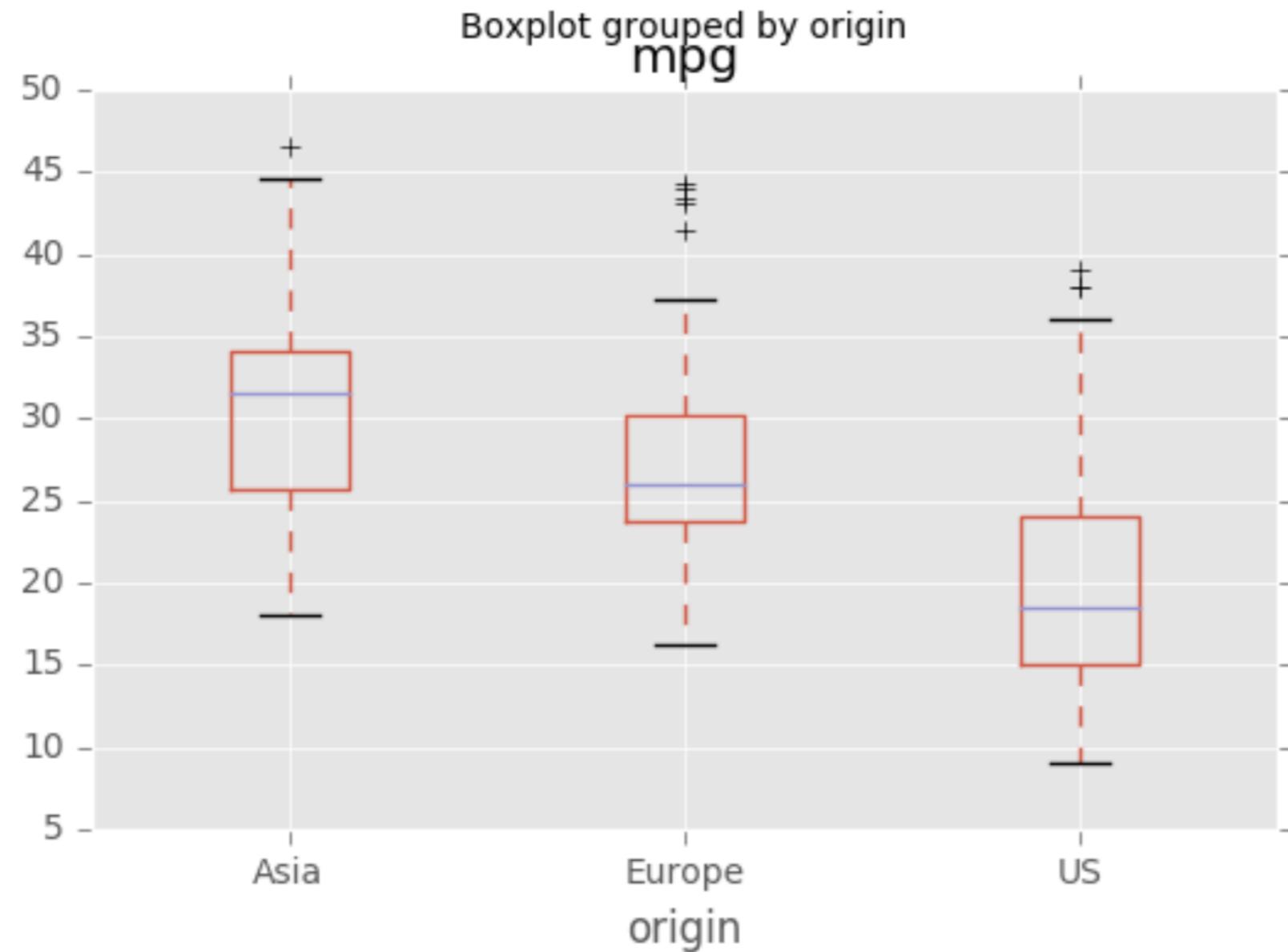
- scikit-learn: OneHotEncoder()
- pandas: get_dummies()

Automobile dataset

- mpg: Target Variable
- Origin: Categorical Feature

	mpg	displ	hp	weight	accel	origin	size
0	18.0	250.0	88	3139	14.5	US	15.0
1	9.0	304.0	193	4732	18.5	US	20.0
2	36.1	91.0	60	1800	16.4	Asia	10.0
3	18.5	250.0	98	3525	19.0	US	15.0
4	34.3	97.0	78	2188	15.8	Europe	10.0

EDA w/ categorical feature



Encoding dummy variables

```
import pandas as pd  
df = pd.read_csv('auto.csv')  
df_origin = pd.get_dummies(df)  
print(df_origin.head())
```

```
mpg    displ     hp   weight   accel   size  origin_Asia  origin_Europe  \\  
0  18.0    250.0    88    3139    14.5    15.0          0            0  
1   9.0    304.0   193    4732    18.5    20.0          0            0  
2  36.1    91.0    60    1800    16.4    10.0          1            0  
3  18.5    250.0    98    3525    19.0    15.0          0            0  
4  34.3    97.0    78    2188    15.8    10.0          0            1  
  
origin_US  
0      1  
1      1  
2      0  
3      1  
4      0
```

Encoding dummy variables

```
df_origin = df_origin.drop('origin_Asia', axis=1)  
print(df_origin.head())
```

	mpg	displ	hp	weight	accel	size	origin_Europe	origin_US
0	18.0	250.0	88	3139	14.5	15.0	0	1
1	9.0	304.0	193	4732	18.5	20.0	0	1
2	36.1	91.0	60	1800	16.4	10.0	0	0
3	18.5	250.0	98	3525	19.0	15.0	0	1
4	34.3	97.0	78	2188	15.8	10.0	1	0

Linear regression with dummy variables

```
from sklearn.model_selection import train_test_split  
from sklearn.linear_model import Ridge
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,  
    test_size=0.3, random_state=42)
```

```
ridge = Ridge(alpha=0.5, normalize=True).fit(X_train,  
    y_train)
```

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

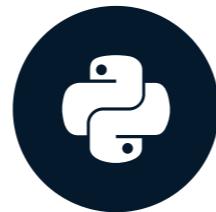
```
0.719064519022
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Handling missing data

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

PIMA Indians dataset

```
df = pd.read_csv('diabetes.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnancies    768 non-null int64
glucose        768 non-null int64
diastolic       768 non-null int64
triceps         768 non-null int64
insulin         768 non-null int64
bmi             768 non-null float64
dpf             768 non-null float64
age             768 non-null int64
diabetes        768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

PIMA Indians dataset

```
print(df.head())
```

```
pregnancies    glucose    diastolic    triceps    insulin    bmi    dpf    age    \\
0            6        148         72        35        0    33.6    0.627    50
1            1        85          66        29        0    26.6    0.351    31
2            8        183         64          0        0    23.3    0.672    32
3            1        89          66        23        94    28.1    0.167    21
4            0        137         40        35       168    43.1    2.288    33
diabetes
0            1
1            0
2            1
3            0
4            1
```

Dropping missing data

```
df.insulin.replace(0, np.nan, inplace=True)  
df.triceps.replace(0, np.nan, inplace=True)  
df.bmi.replace(0, np.nan, inplace=True)  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 768 entries, 0 to 767  
Data columns (total 9 columns):  
pregnancies    768 non-null int64  
glucose        768 non-null int64  
diastolic      768 non-null int64  
triceps        541 non-null float64  
insulin         394 non-null float64  
bmi             757 non-null float64  
dpf             768 non-null float64  
age              768 non-null int64  
diabetes        768 non-null int64  
dtypes: float64(4), int64(5)  
memory usage: 54.1 KB
```

Dropping missing data

```
df = df.dropna()  
df.shape
```

```
(393, 9)
```

Imputing missing data

- Making an educated guess about the missing values
- Example: Using the mean of the non-missing entries

```
from sklearn.preprocessing import Imputer  
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)  
imp.fit(X)  
X = imp.transform(X)
```

Imputing within a pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
logreg = LogisticRegression()
steps = [('imputation', imp),
          ('logistic_regression', logreg)]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3, random_state=42)
```

Imputing within a pipeline

```
pipeline.fit(X_train, y_train)  
y_pred = pipeline.predict(X_test)  
pipeline.score(X_test, y_test)
```

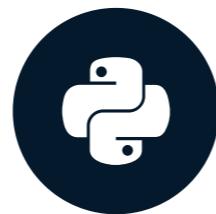
0.75324675324675328

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Centering and scaling

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo Bowne-Anderson
Data Scientist, DataCamp

Why scale your data?

```
print(df.describe())
```

```
fixed acidity    free sulfur dioxide    total sulfur dioxide    density    \\
count    1599.000000                1599.000000                1599.000000    1599.000000
mean     8.319637                  15.874922                  46.467792      0.996747
std      1.741096                  10.460157                  32.895324      0.001887
min      4.600000                  1.000000                  6.000000      0.990070
25%     7.100000                  7.000000                  22.000000      0.995600
50%     7.900000                  14.000000                 38.000000      0.996750
75%     9.200000                  21.000000                 62.000000      0.997835
max     15.900000                 72.000000                 289.000000     1.003690
pH        pH                      sulphates          alcohol         quality
count    1599.000000                1599.000000                1599.000000    1599.000000
mean     3.311113                  0.658149                  10.422983      0.465291
std      0.154386                  0.169507                  1.065668      0.498950
min      2.740000                  0.330000                  8.400000      0.000000
25%     3.210000                  0.550000                  9.500000      0.000000
50%     3.310000                  0.620000                  10.200000     0.000000
75%     3.400000                  0.730000                  11.100000     1.000000
max     4.010000                  2.000000                 14.900000     1.000000
```

Why scale your data?

- Many models use some form of distance to inform them
- Features on larger scales can unduly influence the model
- Example: k-NN uses distance explicitly when making predictions
- We want features to be on a similar scale
- Normalizing (or scaling and centering)

Ways to normalize your data

- Standardization: Subtract the mean and divide by variance
- All features are centered around zero and have variance one
- Can also subtract the minimum and divide by the range
- Minimum zero and maximum one
- Can also normalize so the data ranges from -1 to +1
- See scikit-learn docs for further details

Scaling in scikit-learn

```
from sklearn.preprocessing import scale  
X_scaled = scale(X)
```

```
np.mean(X), np.std(X)
```

```
(8.13421922452, 16.7265339794)
```

```
np.mean(X_scaled), np.std(X_scaled)
```

```
(2.54662653149e-15, 1.0)
```

Scaling in a pipeline

```
from sklearn.preprocessing import StandardScaler  
steps = [('scaler', StandardScaler()),  
         ('knn', KNeighborsClassifier())]  
pipeline = Pipeline(steps)  
X_train, X_test, y_train, y_test = train_test_split(X, y,  
                                                    test_size=0.2, random_state=21)  
knn_scaled = pipeline.fit(X_train, y_train)  
y_pred = pipeline.predict(X_test)  
accuracy_score(y_test, y_pred)
```

0.956

```
knn_unscaled = KNeighborsClassifier().fit(X_train, y_train)  
knn_unscaled.score(X_test, y_test)
```

0.928

CV and scaling in a pipeline

```
steps = [('scaler', StandardScaler()),  
         ('knn', KNeighborsClassifier())]  
pipeline = Pipeline(steps)  
parameters = {knn__n_neighbors: np.arange(1, 50)}  
X_train, X_test, y_train, y_test = train_test_split(X, y,  
                                                 test_size=0.2, random_state=21)  
cv = GridSearchCV(pipeline, param_grid=parameters)  
cv.fit(X_train, y_train)  
y_pred = cv.predict(X_test)
```

Scaling and CV in a pipeline

```
print(cv.best_params_)
```

```
{'knn__n_neighbors': 41}
```

```
print(cv.score(X_test, y_test))
```

```
0.956
```

```
print(classification_report(y_test, y_pred))
```

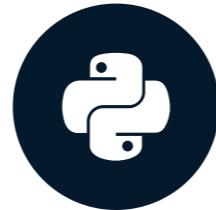
	precision	recall	f1-score	support
0	0.97	0.90	0.93	39
1	0.95	0.99	0.97	75
avg / total	0.96	0.96	0.96	114

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Final thoughts

SUPERVISED LEARNING WITH SCIKIT-LEARN



Hugo and Andy

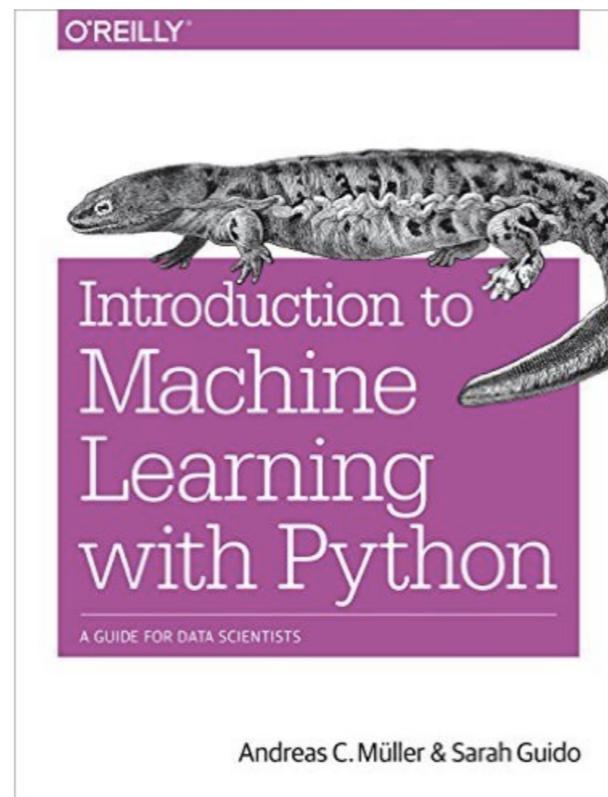
Data Scientists

What you've learned

- Using machine learning techniques to build predictive models
- For both regression and classification problems
- With real-world data
- Underfitting and overfitting
- Test-train split
- Cross-validation
- Grid search

What you've learned

- Regularization, lasso and ridge regression
- Data preprocessing
- For more: Check out the scikit-learn documentation



Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN