SUPERVISED LEARNING WITH SCIKIT-LEARN

Supervised learning

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

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

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Exploratory data analysis

The Iris dataset

- Features:
  - Petal length
  - Petal width
  - Sepal length
  - Sepal width
- Target variable: Species
  - Versicolor
  - Virginica
  - Setosa

The Iris dataset in scikit-learn

```
In [1]: from sklearn import datasets

In [2]: import pandas as pd

In [3]: import numpy as np

In [4]: import matplotlib.pyplot as plt

In [5]: plt.style.use('ggplot')

In [6]: iris = datasets.load_iris()

In [7]: type(iris)
Out[7]: sklearn.datasets.base.Bunch

In [8]: print(iris.keys())
dict_keys(['data', 'target_names', 'DESCR', 'feature_names', 'target'])
```

The Iris dataset in scikit-learn

```
In [9]: type(iris.data), type(iris.target)
Out[9]: (numpy.ndarray, numpy.ndarray)

In [10]: iris.data.shape
Out[10]: (150, 4)

In [11]: iris.target_names
Out[11]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

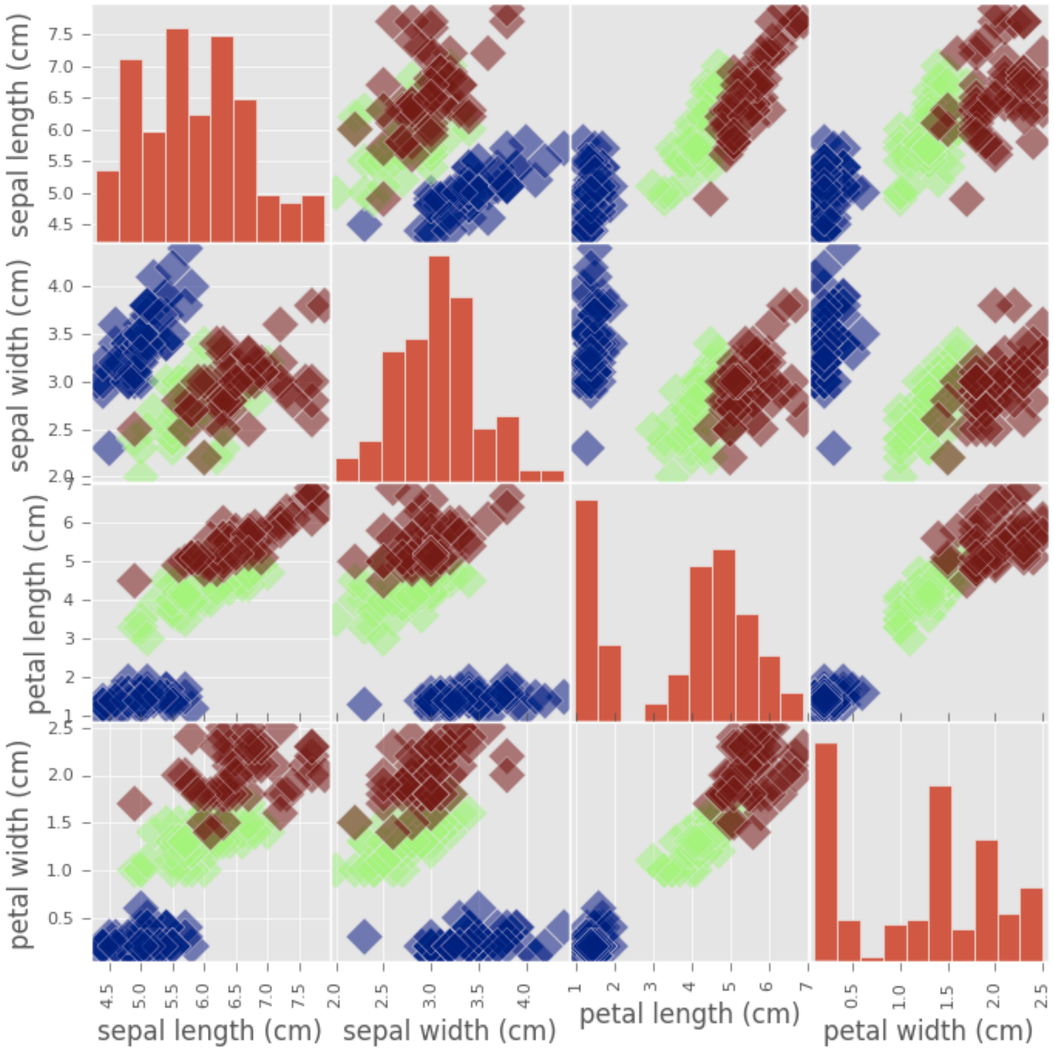
Exploratory data analysis (EDA)

```
In [12]: X = iris.data

In [13]: y = iris.target

In [14]: df = pd.DataFrame(X, columns=iris.feature_names)

In [15]: 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

```
In [16]: _ = pd.scatter_matrix(df, c = y, figsize = [8, 8],
    ...:                       s=150, marker = 'D')
```

Visual EDA


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

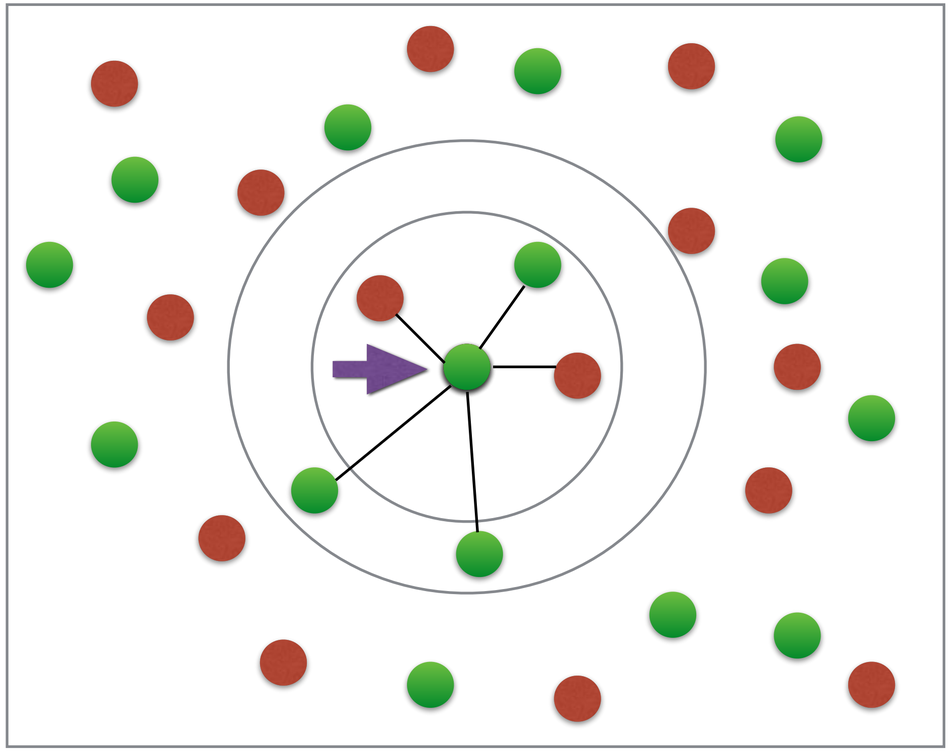
SUPERVISED LEARNING WITH SCIKIT-LEARN

The classification challenge

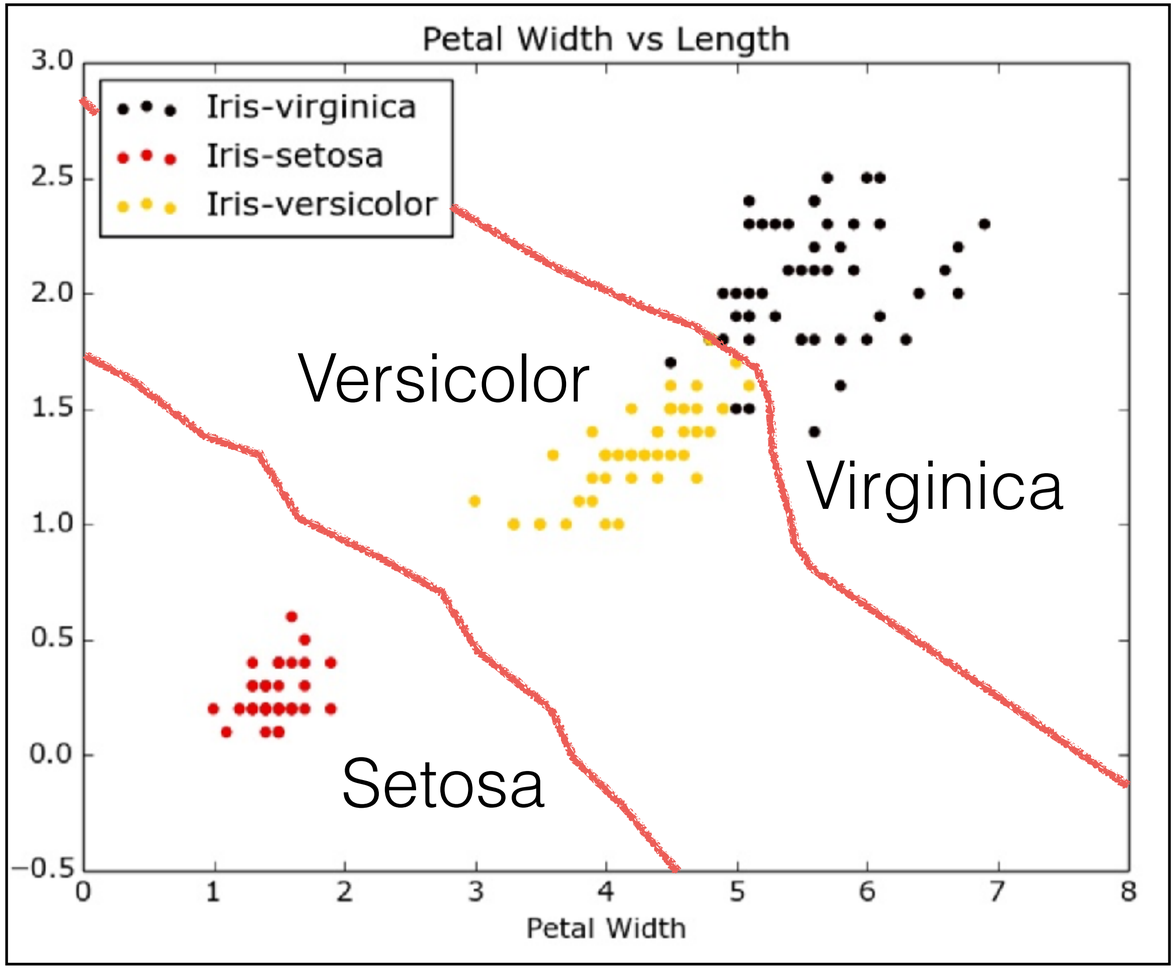
k-Nearest Neighbors

- 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-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

```
In [1]: from sklearn.neighbors import KNeighborsClassifier

In [2]: knn = KNeighborsClassifier(n_neighbors=6)

In [3]: knn.fit(iris['data'], iris['target'])
Out[3]: KNeighborsClassifier(algorithm='auto', leaf_size=30,
   ...: metric='minkowski',metric_params=None, n_jobs=1,
   ...: n_neighbors=6, p=2,weights='uniform')

In [4]: iris['data'].shape
Out[4]: (150, 4)

In [5]: iris['target'].shape
Out[5]: (150,)
```

Predicting on unlabeled data

```
In [6]: prediction = knn.predict(X_new)

In [7]: X_new.shape
Out[7]: (3, 4)

In [8]: print('Prediction {}'.format(prediction))
Prediction: [1 1 0]
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Measuring model performance

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

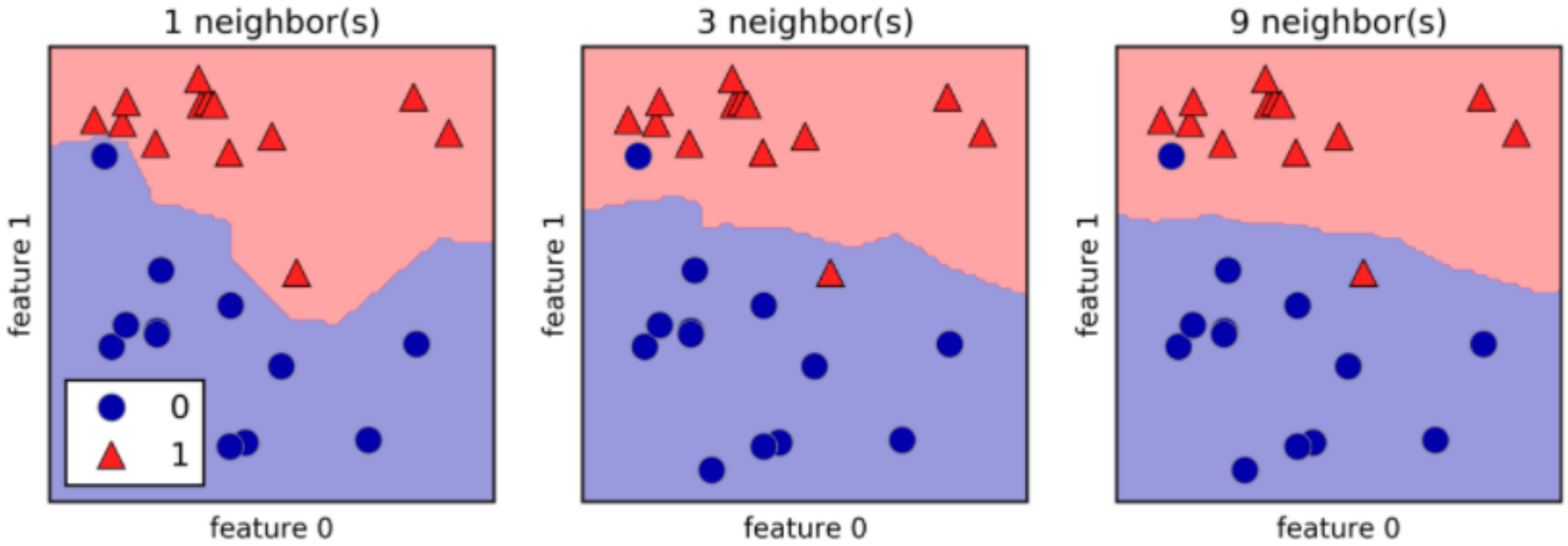
- Could compute accuracy on data used to fit classifier
  - NOT indicative of ability to generalize
- Split data into training and test set
  - Fit/train the classifier on the training set
  - Make predictions on test set
  - Compare predictions with the known labels

Train/test split

```
In [1]: from sklearn.model_selection import train_test_split

In [2]: X_train, X_test, y_train, y_test =
   ...: train_test_split(X, y, test_size=0.3,
   ...:                  random_state=21, stratify=y)

In [3]: knn = KNeighborsClassifier(n_neighbors=8)

In [4]: knn.fit(X_train, y_train)

In [5]: y_pred = knn.predict(X_test)

In [6]: 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]

In [7]: knn.score(X_test, y_test)
Out[7]: 0.9555555555555556
```

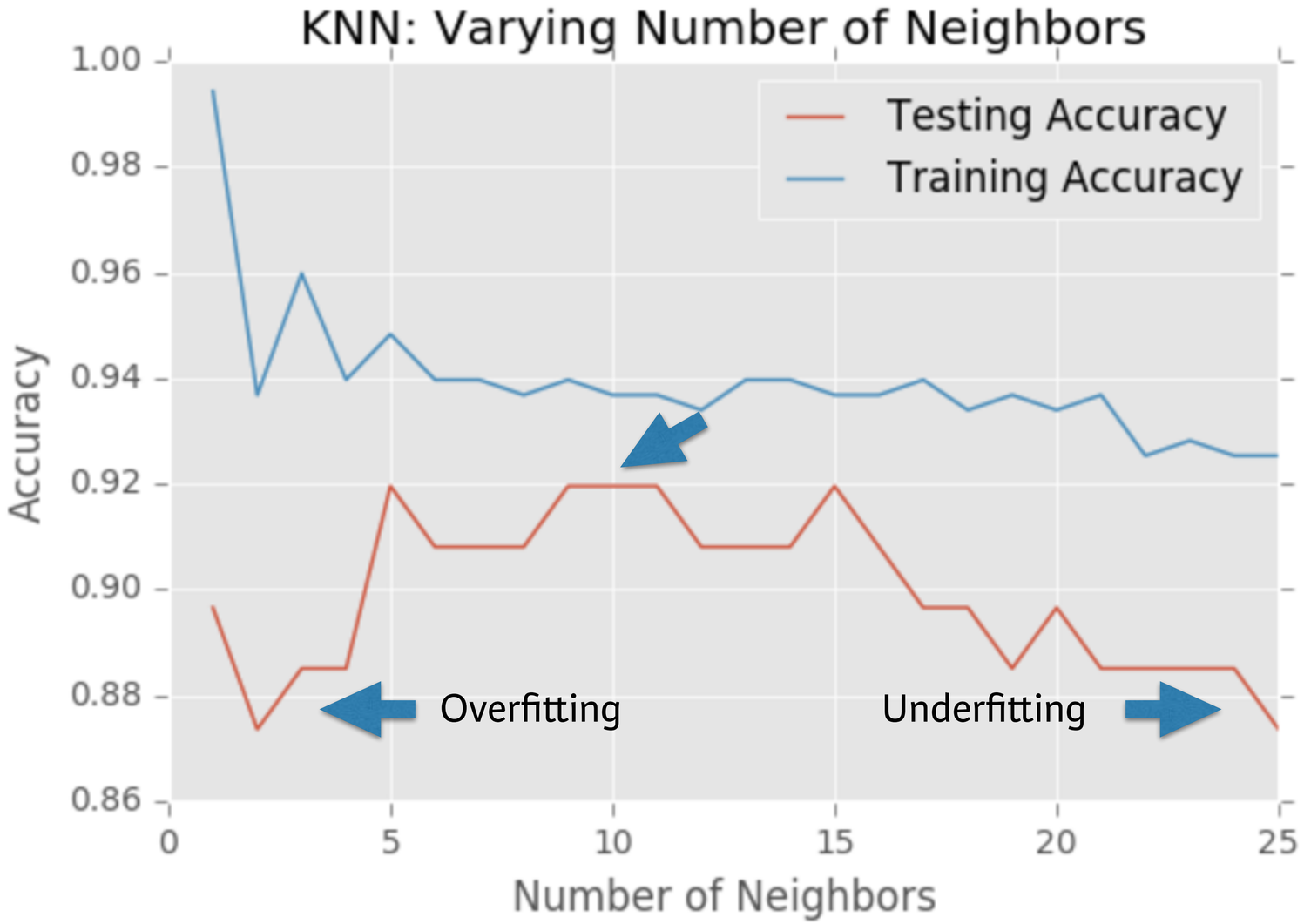
Model complexity

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



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

Model complexity and over/underfitting



SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Introduction to regression

Boston housing data

```
In [1]: boston = pd.read_csv('boston.csv')

In [2]: 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

```
In [3]: X = boston.drop('MEDV', axis=1).values

In [4]: y = boston['MEDV'].values
```

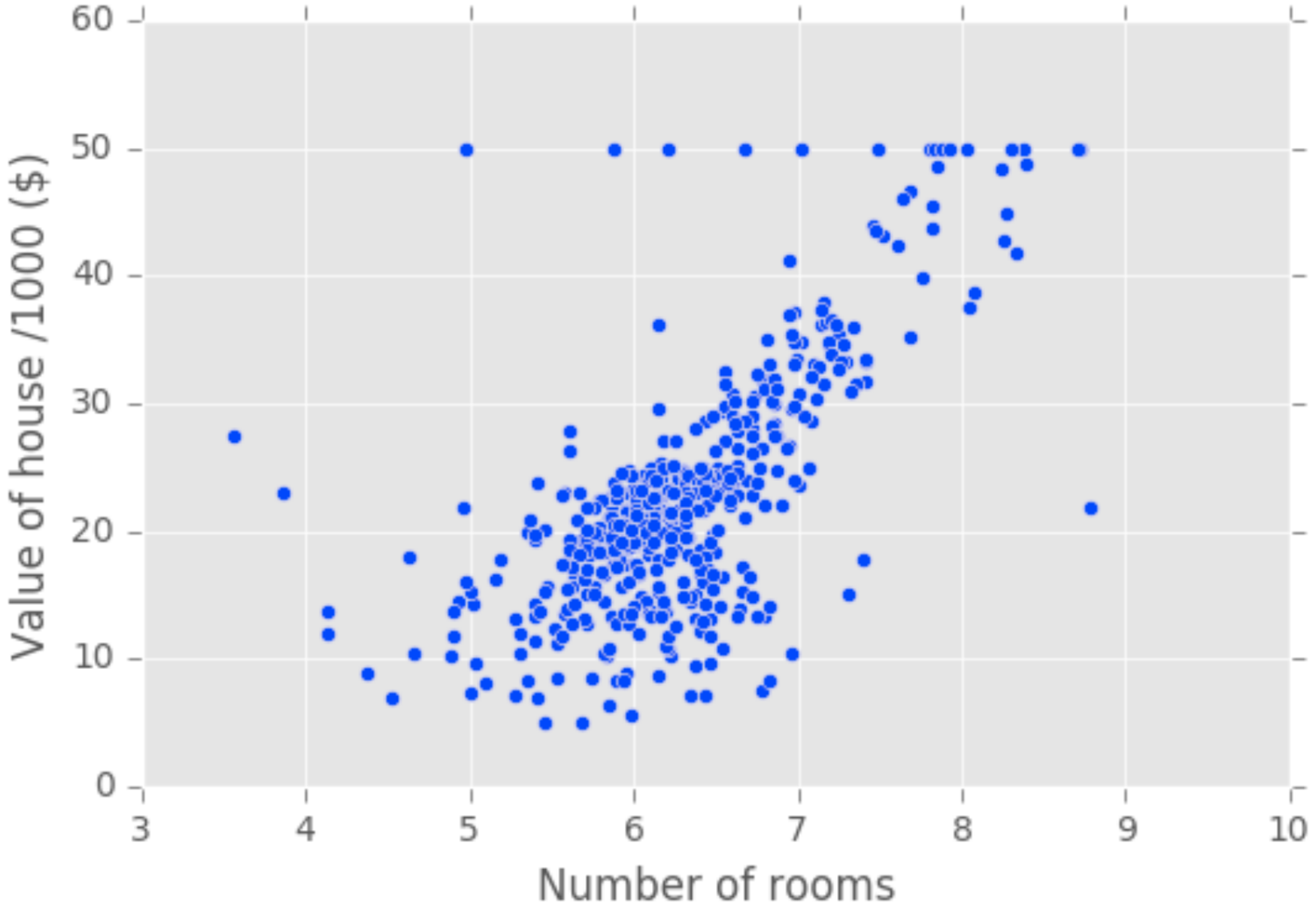
Predicting house value from a single feature

```
In [5]: X_rooms = X[:,5]

In [6]: type(X_rooms), type(y)
Out[6]: (numpy.ndarray, numpy.ndarray)

In [7]: y = y.reshape(-1, 1)

In [8]: X_rooms = X_rooms.reshape(-1, 1)
```

Plotting house value vs. number of rooms

```
In [9]: plt.scatter(X_rooms, y)

In [10]: plt.ylabel('Value of house /1000 ($)')

In [11]: plt.xlabel('Number of rooms')

In [12]: plt.show();
```

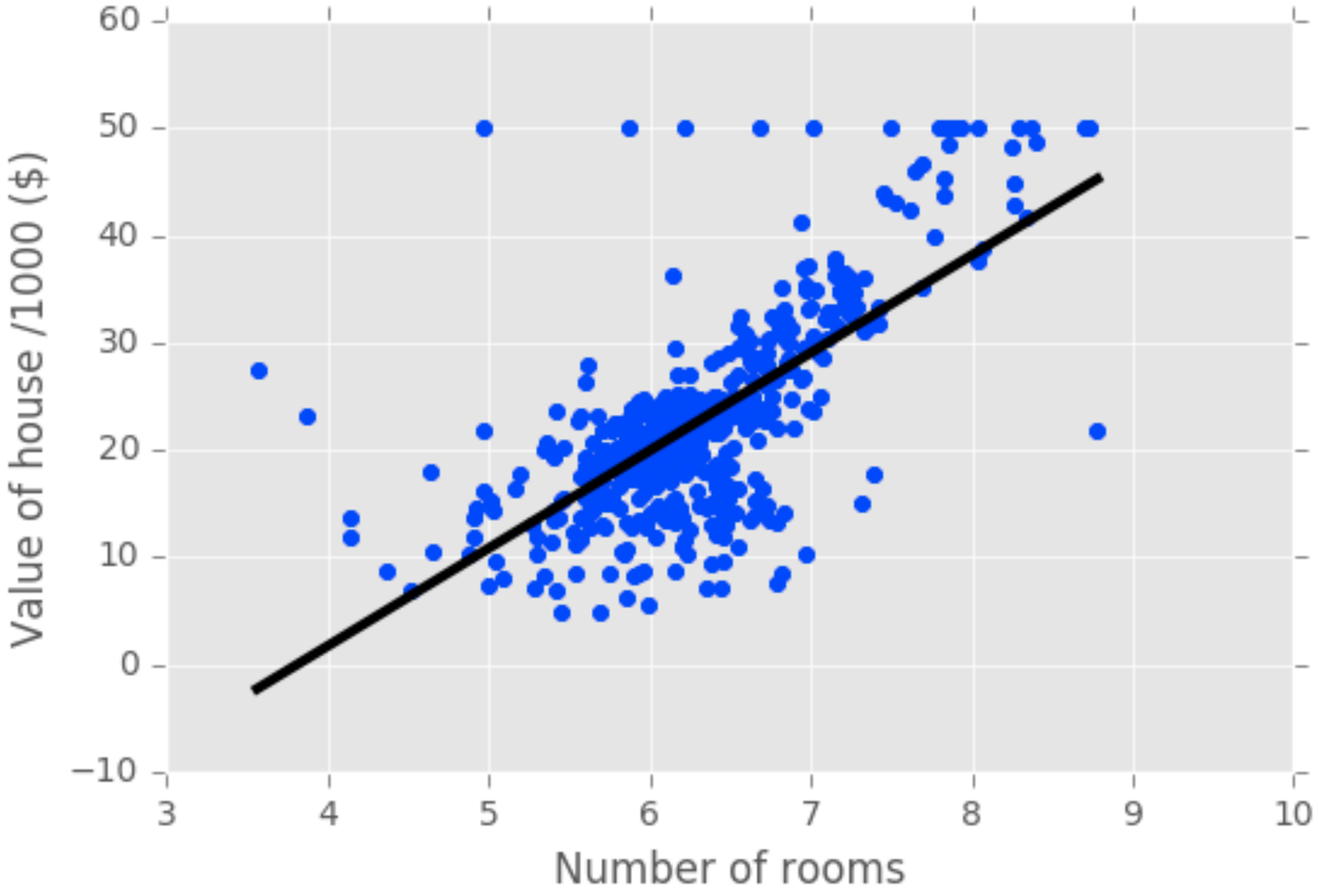
Plotting house value vs. number of rooms



Fitting a regression model

```
In [13]: import numpy as np

In [14]: from sklearn import linear_model

In [15]: reg = linear_model.LinearRegression()

In [16]: reg.fit(X_rooms, y)

In [17]: prediction_space = np.linspace(min(X_rooms),
    ...:                                max(X_rooms)).reshape(-1, 1)

In [18]: plt.scatter(X_rooms, y, color='blue')

In [19]: plt.plot(prediction_space, reg.predict(prediction_space),
    ...:          color='black', linewidth=3)

In [20]: plt.show()
```

Fitting a regression model



SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

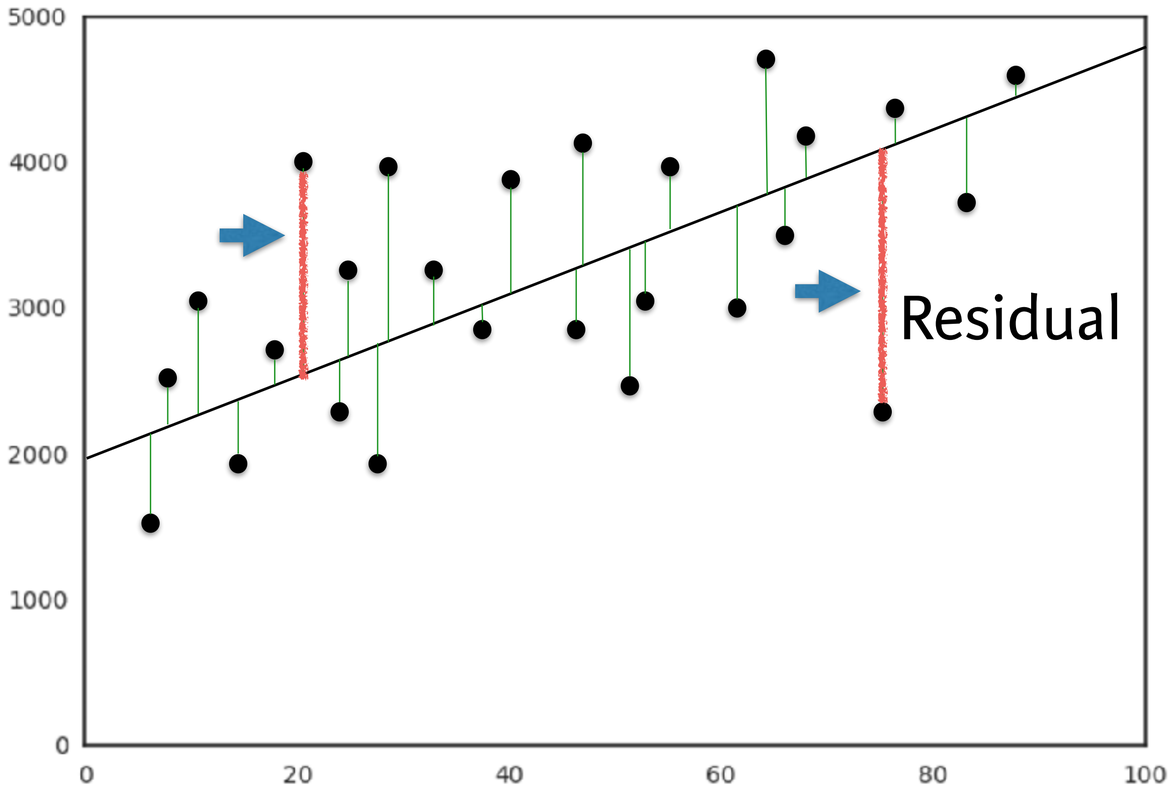
The basics of linear regression

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 function for any given line
  - Choose the line that minimizes the error 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:

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

  - Must specify coefficient for each feature and the variable b
- Scikit-learn API works exactly the same way:
  - Pass two arrays: Features, and target

Linear regression on all features

```
In [1]: from sklearn.model_selection import train_test_split

In [2]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size = 0.3, random_state=42)

In [3]: reg_all = linear_model.LinearRegression()

In [4]: reg_all.fit(X_train, y_train)

In [5]: y_pred = reg_all.predict(X_test)

In [6]: reg_all.score(X_test, y_test)
Out[6]: 0.7112260057484926
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

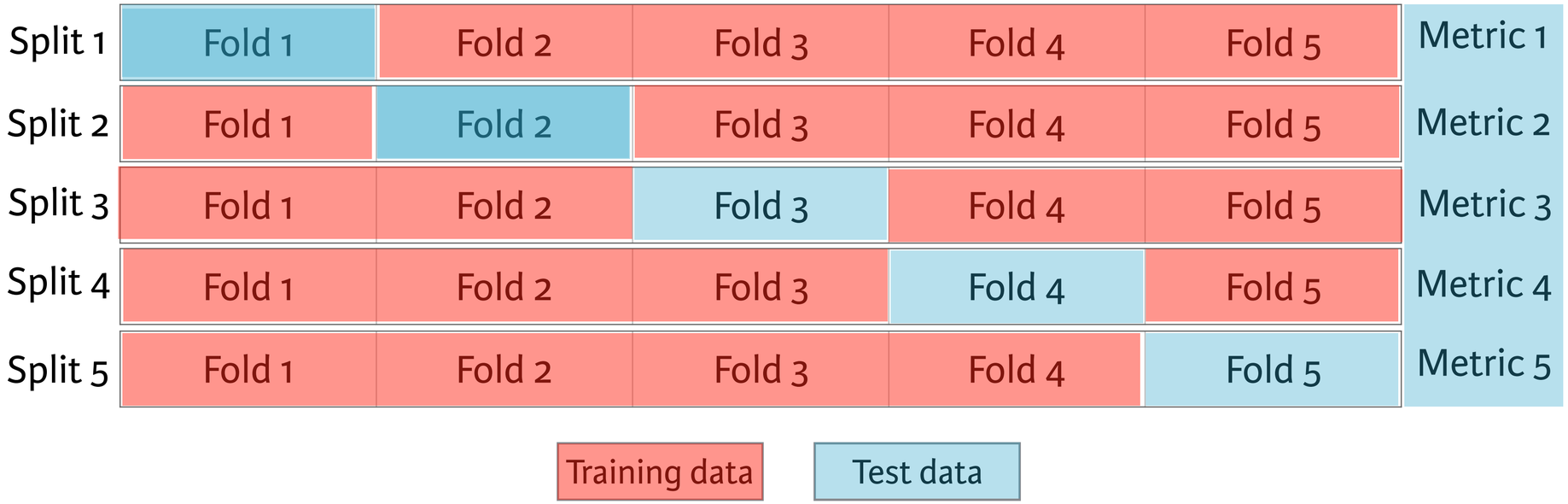
SUPERVISED LEARNING WITH SCIKIT-LEARN

Cross-validation

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



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

```
In [1]: from sklearn.model_selection import cross_val_score

In [2]: reg = linear_model.LinearRegression()

In [3]: cv_results = cross_val_score(reg, X, y, cv=5)

In [4]: print(cv_results)
[ 0.63919994  0.71386698  0.58702344  0.07923081 -0.25294154]

In [5]: np.mean(cv_results)
Out[5]: 0.3532759243958705 8
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Regularized regression

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

```
In [1]: from sklearn.linear_model import Ridge

In [2]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size = 0.3, random_state=42)

In [3]: ridge = Ridge(alpha=0.1, normalize=True)

In [4]: ridge.fit(X_train, y_train)

In [5]: ridge_pred = ridge.predict(X_test)

In [6]: ridge.score(X_test, y_test)
Out[6]: 0.6996938275127317 9
```

Lasso regression

- Loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} |a_i|$

Lasso regression in scikit-learn

```
In [1]: from sklearn.linear_model import Lasso

In [2]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size = 0.3, random_state=42)

In [3]: lasso = Lasso(alpha=0.1, normalize=True)

In [4]: lasso.fit(X_train, y_train)

In [5]: lasso_pred = lasso.predict(X_test)

In [6]: lasso.score(X_test, y_test)
Out[6]: 0.5950229535328506
```

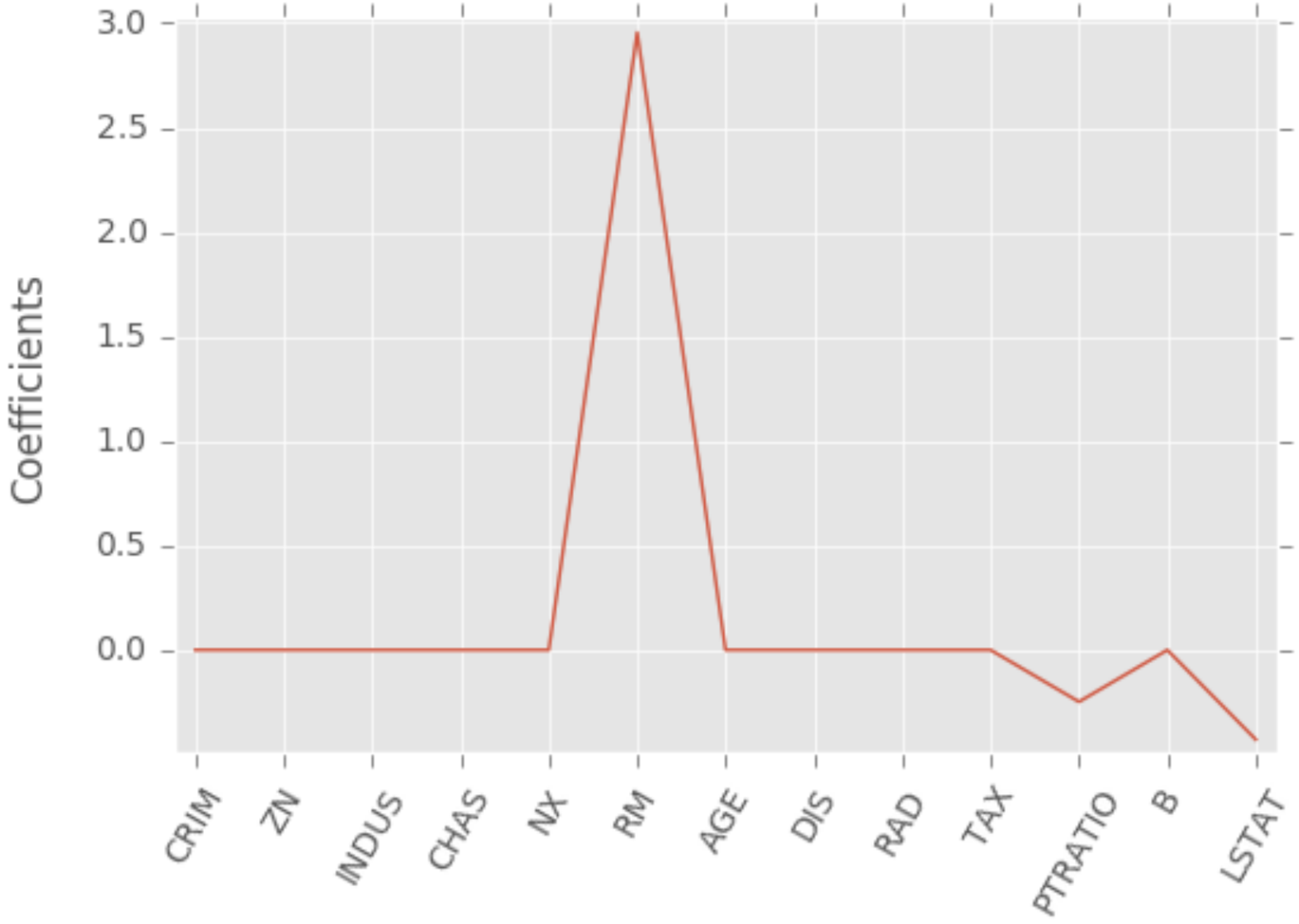
Lasso regression for feature selection

- 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

```
In [1]: from sklearn.linear_model import Lasso

In [2]: names = boston.drop('MEDV', axis=1).columns

In [3]: lasso = Lasso(alpha=0.1)

In [4]: lasso_coef = lasso.fit(X, y).coef_

In [5]: _ = plt.plot(range(len(names)), lasso_coef)

In [6]: _ = plt.xticks(range(len(names)), names, rotation=60)

In [7]: _ = plt.ylabel('Coefficients')

In [8]: plt.show()
```

Lasso for feature selection in scikit-learn



SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

How good is your model?

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

- Accuracy: $\frac{tp + tn}{tp + tn + fp + fn}$

Metrics from the confusion matrix

- Precision : $\frac{tp}{tp + fp}$
- Recall : $\frac{tp}{tp + fn}$
- F1 score : $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

- High precision: Not many real emails predicted as spam
- High recall: Predicted most spam emails correctly

Confusion matrix in scikit-learn

```
In [1]: from sklearn.metrics import classification_report

In [2]: from sklearn.metrics import confusion_matrix

In [3]: knn = KNeighborsClassifier(n_neighbors=8)

In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size=0.4, random_state=42)

In [5]: knn.fit(X_train, y_train)

In [6]: y_pred = knn.predict(X_test)
```

Confusion matrix in scikit-learn

```
In [7]: print(confusion_matrix(y_test, y_pred))
[[52  7]
 [ 3 112]]

In [8]: 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
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Logistic regression and the ROC curve

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

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

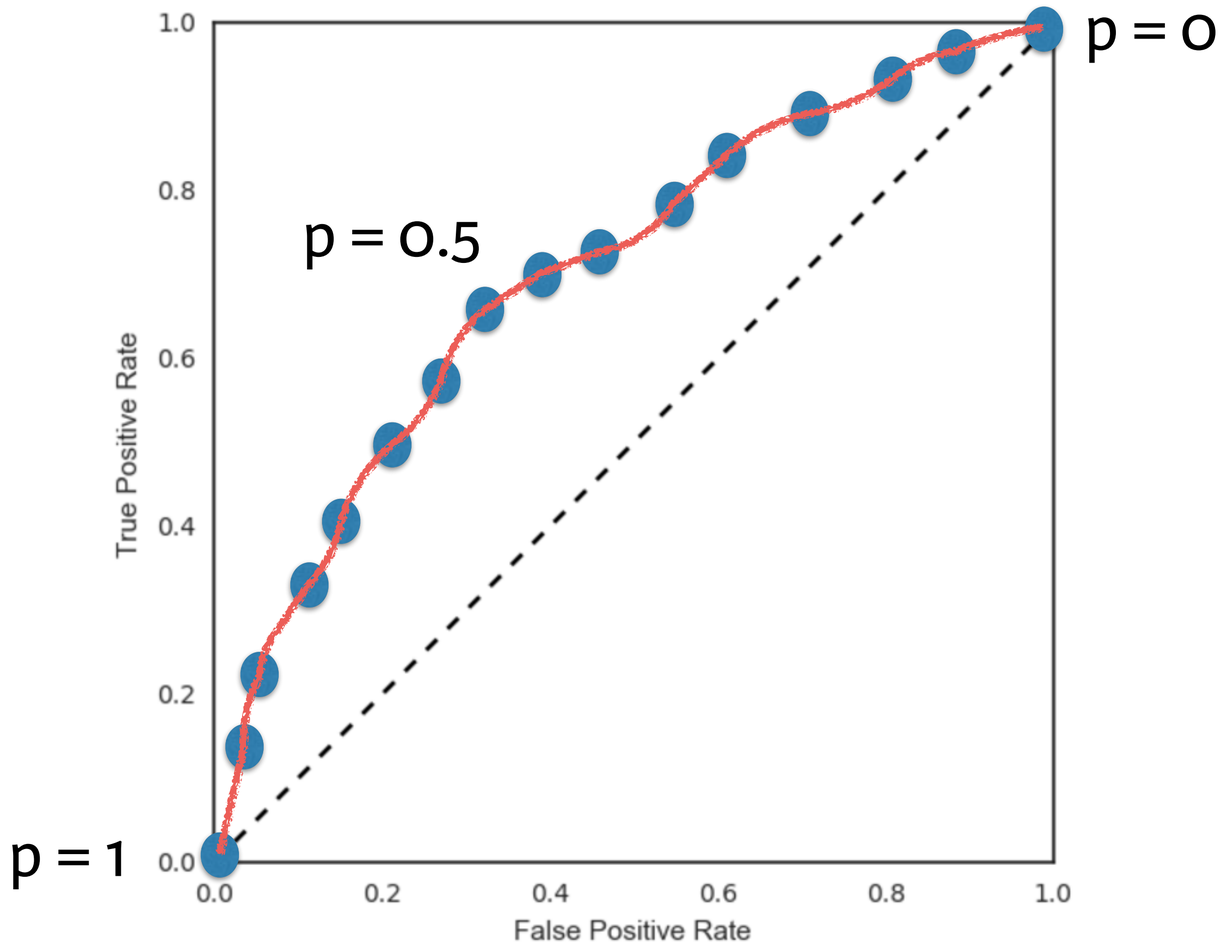
Logistic regression in scikit-learn

```
In [1]: from sklearn.linear_model import LogisticRegression

In [2]: from sklearn.model_selection import train_test_split

In [3]: logreg = LogisticRegression()

In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size=0.4, random_state=42)

In [5]: logreg.fit(X_train, y_train)

In [6]: 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?

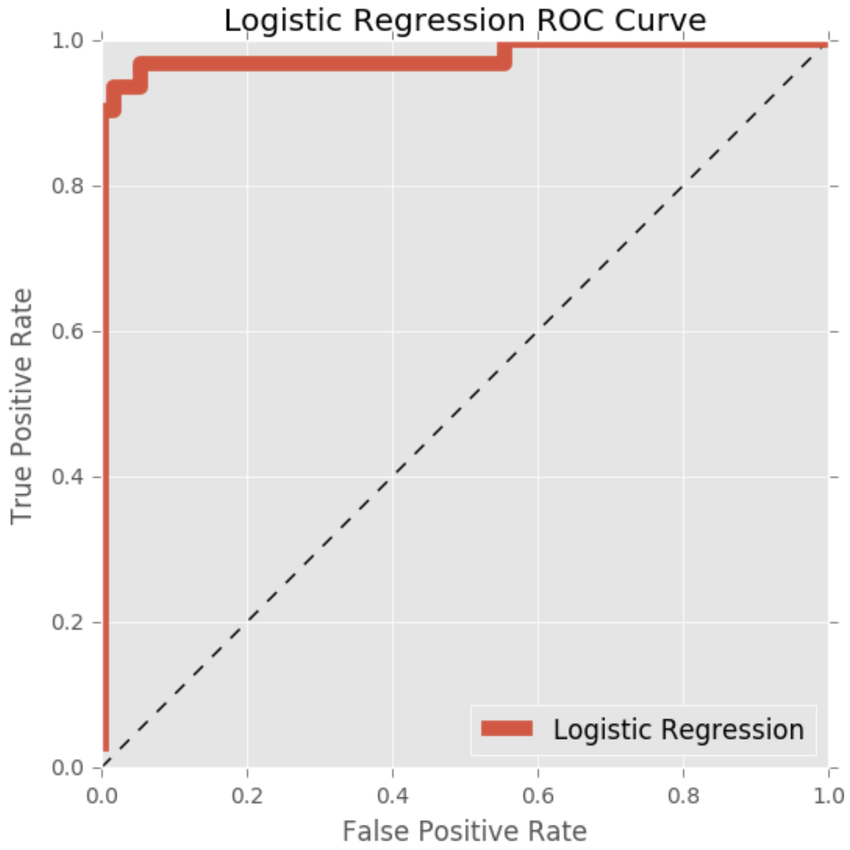
The ROC curve



Plotting the ROC curve

```
In [1]: from sklearn.metrics import roc_curve

In [2]: y_pred_prob = logreg.predict_proba(X_test)[:,1]

In [3]: fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

In [4]: plt.plot([0, 1], [0, 1], 'k--')

In [5]: plt.plot(fpr, tpr, label='Logistic Regression')

In [6]: plt.xlabel('False Positive Rate’)

In [7]: plt.ylabel('True Positive Rate')

In [8]: plt.title('Logistic Regression ROC Curve')

In [9]: plt.show();
```

Plotting the ROC curve




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

SUPERVISED LEARNING WITH SCIKIT-LEARN

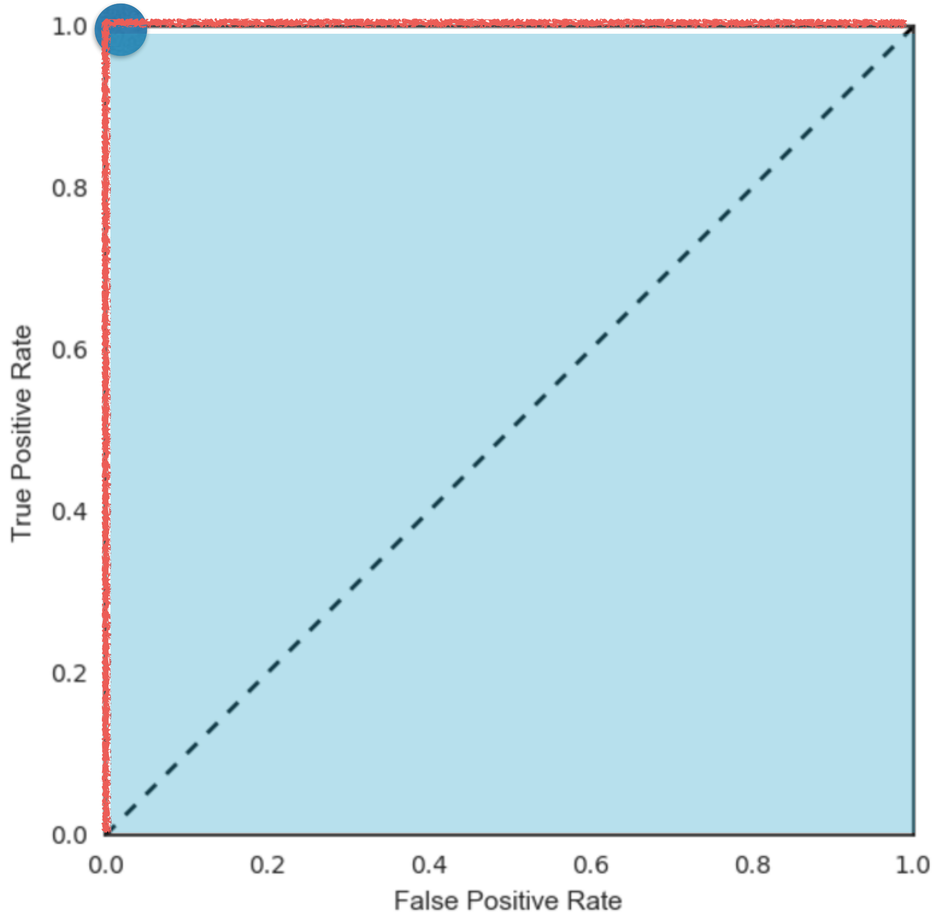
Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Area under the ROC curve

Area under the ROC curve (AUC)

- Larger area under the ROC curve = better model



AUC in scikit-learn

```
In [1]: from sklearn.metrics import roc_auc_score

In [2]: logreg = LogisticRegression()

In [3]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size=0.4, random_state=42)

In [4]: logreg.fit(X_train, y_train)

In [5]: y_pred_prob = logreg.predict_proba(X_test)[:,1]

In [6]: roc_auc_score(y_test, y_pred_prob)
Out[6]: 0.997466216216
```

AUC using cross-validation

```
In [7]: from sklearn.model_selection import cross_val_score

In [8]: cv_scores = cross_val_score(logreg, X, y, cv=5,
   ...:                             scoring='roc_auc')

In [9]: print(cv_scores)
[ 0.99673203  0.99183007  0.99583796  1.          0.96140652]
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hyperparameter tuning

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

C \ Alpha	0.1	0.2	0.3	0.4
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
0.1	0.698	0.692	0.688	0.675

GridSearchCV in scikit-learn

```
In [1]: from sklearn.model_selection import GridSearchCV

In [2]: param_grid = {'n_neighbors': np.arange(1, 50)}

In [3]: knn = KNeighborsClassifier()

In [4]: knn_cv = GridSearchCV(knn, param_grid, cv=5)

In [5]: knn_cv.fit(X, y)

In [6]: knn_cv.best_params_
Out[6]: {'n_neighbors': 12}

In [7]: knn_cv.best_score_
Out[7]: 0.933216168717
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hold-out set for final evaluation

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

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

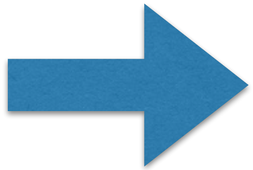
Preprocessing data

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



origin_Asia	origin_US	ɔe	origin_US
0	1		1
0	0		0
1	0		0

Dealing with categorical features in Python

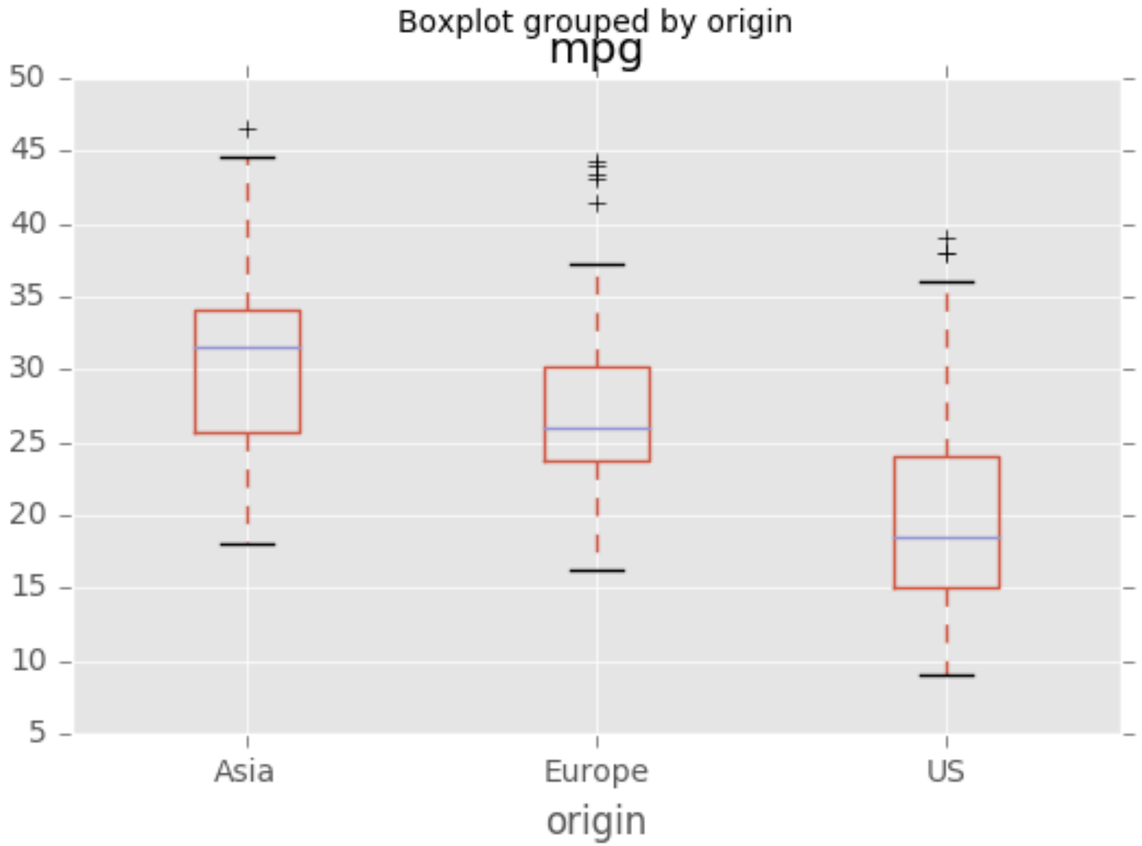
- scikit-learn: OneHotEncoder()
- pandas: get_dummies()

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

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('auto.csv')

In [3]: df_origin = pd.get_dummies(df)

In [4]: 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

```
In [5]: df_origin = df_origin.drop('origin_Asia', axis=1)

In [6]: 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

```
In [7]: from sklearn.model_selection import train_test_split

In [8]: from sklearn.linear_model import Ridge

In [9]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size=0.3, random_state=42)

In [10]: ridge = Ridge(alpha=0.5, normalize=True).fit(X_train,
   ...:                                               y_train)

In [11]: ridge.score(X_test, y_test)
Out[11]: 0.719064519022
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Handling missing data

PIMA Indians dataset

```
In [1]: df = pd.read_csv('diabetes.csv')

In [2]: 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

```
In [3]: 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

```
In [8]: df.insulin.replace(0, np.nan, inplace=True)

In [9]: df.triceps.replace(0, np.nan, inplace=True)

In [10]: df.bmi.replace(0, np.nan, inplace=True)

In [11]: 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

```
In [12]: df = df.dropna()

In [13]: df.shape
Out[13]: (393, 9)
```

Imputing missing data

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

```
In [1]: from sklearn.preprocessing import Imputer

In [2]: imp = Imputer(missing_values='NaN', strategy='mean', axis=0)

In [3]: imp.fit(X)

In [4]: X = imp.transform(X)
```

Imputing within a pipeline

```
In [1]: from sklearn.pipeline import Pipeline

In [2]: from sklearn.preprocessing import Imputer

In [3]: imp = Imputer(missing_values='NaN', strategy='mean', axis=0)

In [4]: logreg = LogisticRegression()

In [5]: steps = [('imputation', imp),
   ...:          ('logistic_regression', logreg)]

In [6]: pipeline = Pipeline(steps)

In [7]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size=0.3, random_state=42)
```

Imputing within a pipeline

```
In [8]: pipeline.fit(X_train, y_train)

In [9]: y_pred = pipeline.predict(X_test)

In [10]: pipeline.score(X_test, y_test)
Out[10]: 0.7532467532467532
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Centering and scaling

Why scale your data?

```
In [1]: 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    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

```
In [2]: from sklearn.preprocessing import scale

In [3]: X_scaled = scale(X)

In [4]: np.mean(X), np.std(X)
Out[4]: (8.13421922452, 16.7265339794)

In [5]: np.mean(X_scaled), np.std(X_scaled)
Out[5]: (2.54662653149e-15, 1.0)
```

Scaling in a pipeline

```
In [6]: from sklearn.preprocessing import StandardScaler

In [7]: steps = [('scaler', StandardScaler()),
   ...:          ('knn', KNeighborsClassifier())]

In [8]: pipeline = Pipeline(steps)

In [9]: X_train, X_test, y_train, y_test = train_test_split(X, y,
   ...: test_size=0.2, random_state=21)

In [10]: knn_scaled = pipeline.fit(X_train, y_train)

In [11]: y_pred = pipeline.predict(X_test)

In [12]: accuracy_score(y_test, y_pred)
Out[12]: 0.956

In [13]: knn_unscaled = KNeighborsClassifier().fit(X_train, y_train)

In [14]: knn_unscaled.score(X_test, y_test)
Out[14]: 0.928
```

CV and scaling in a pipeline

```
In [14]: steps = [('scaler', StandardScaler()),
    ...:          ('knn', KNeighborsClassifier())]

In [15]: pipeline = Pipeline(steps)

In [16]: parameters = {knn__n_neighbors=np.arange(1, 50)}

In [17]: X_train, X_test, y_train, y_test = train_test_split(X, y,
    ...: test_size=0.2, random_state=21)

In [18]: cv = GridSearchCV(pipeline, param_grid=parameters)

In [19]: cv.fit(X_train, y_train)

In [20]: y_pred = cv.predict(X_test)
```

Scaling and CV in a pipeline

```
In [21]: print(cv.best_params_)
{'knn__n_neighbors': 41}

In [22]: print(cv.score(X_test, y_test))
0.956

In [23]: 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
```

SUPERVISED LEARNING WITH SCIKIT-LEARN

Let’s practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

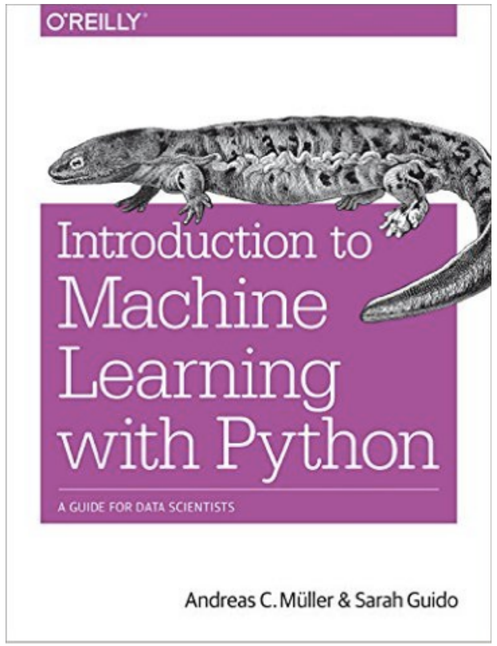
Final thoughts

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



SUPERVISED LEARNING WITH SCIKIT-LEARN

Congratulations!