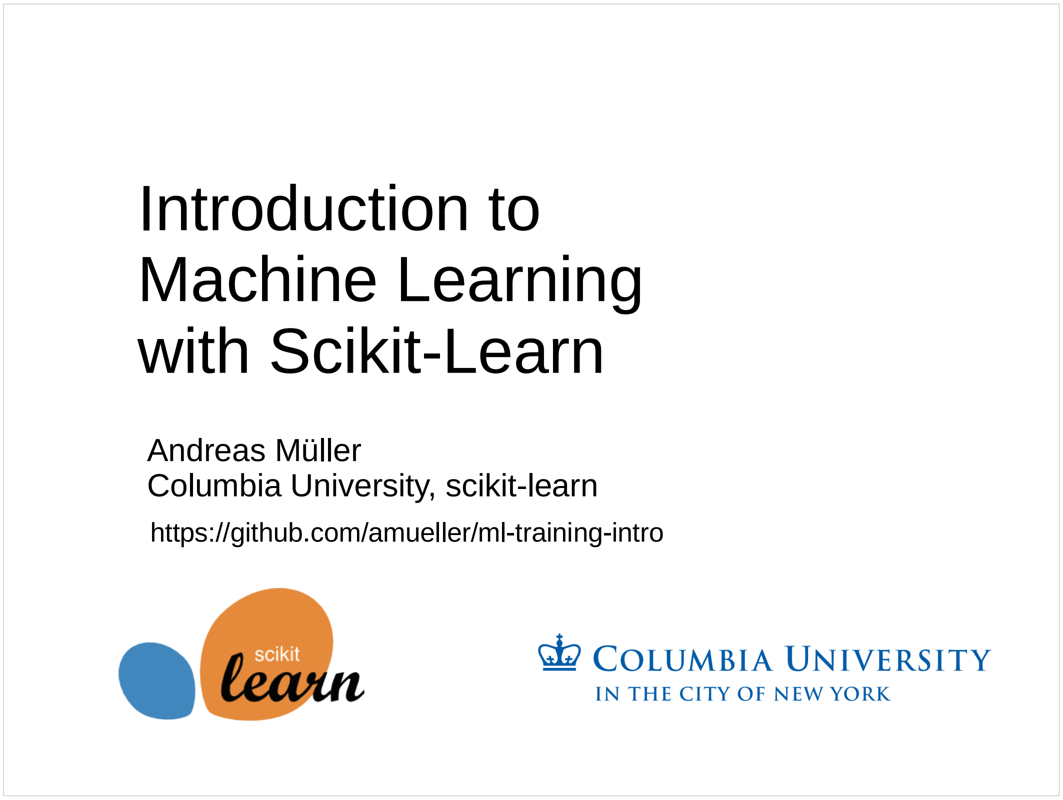


Introduction to Machine Learning with Scikit-Learn

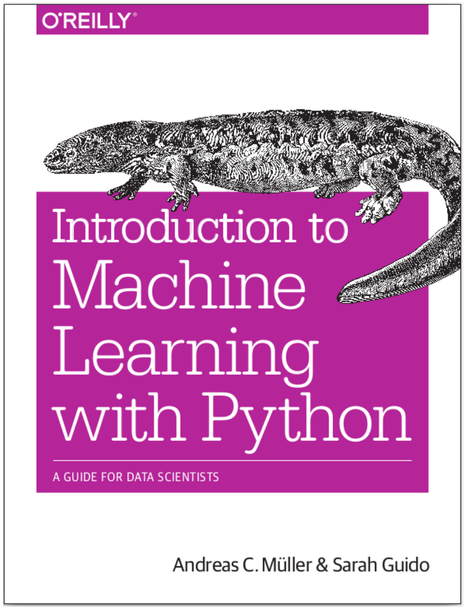
Andreas Müller
Columbia University, scikit-learn

https://github.com/amueller/ml-training-intro

Hey and welcome to my talk on machine learning and how to do it with python.

My goal for the next 45 minutes is to convince you that machine learning can help you in your research, make your life easier, and allow you to process much more data than you could before – and that it's all pretty easy to do.

I'm andy, I work at Columbia and I've been contributing to the scikit-learn project for a couple of years now. Scikit-learn is the main machine learning tool in python and I'll tell you a bit more about it later.



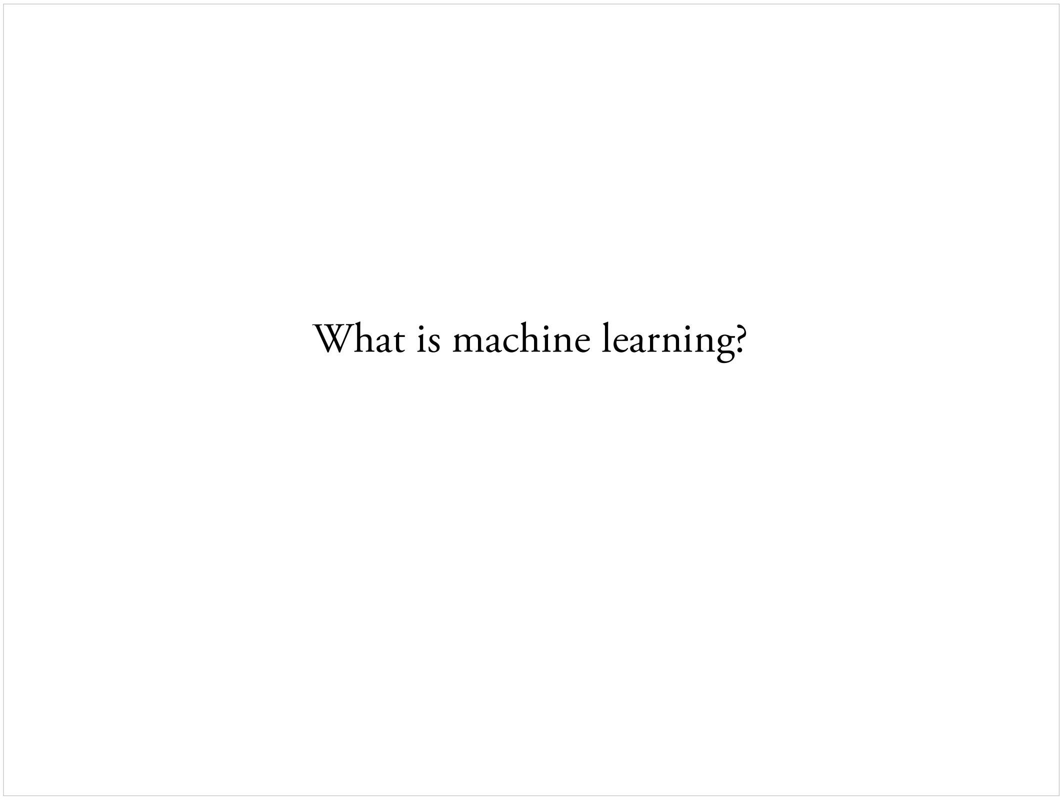
O'REILLY

Introduction to Machine Learning with Python

A GUIDE FOR DATA SCIENTISTS

Andreas C. Müller & Sarah Guido

Lecture series:
http://www.cs.columbia.edu/~amueller/comsw4995s18/schedule/



What is machine learning?

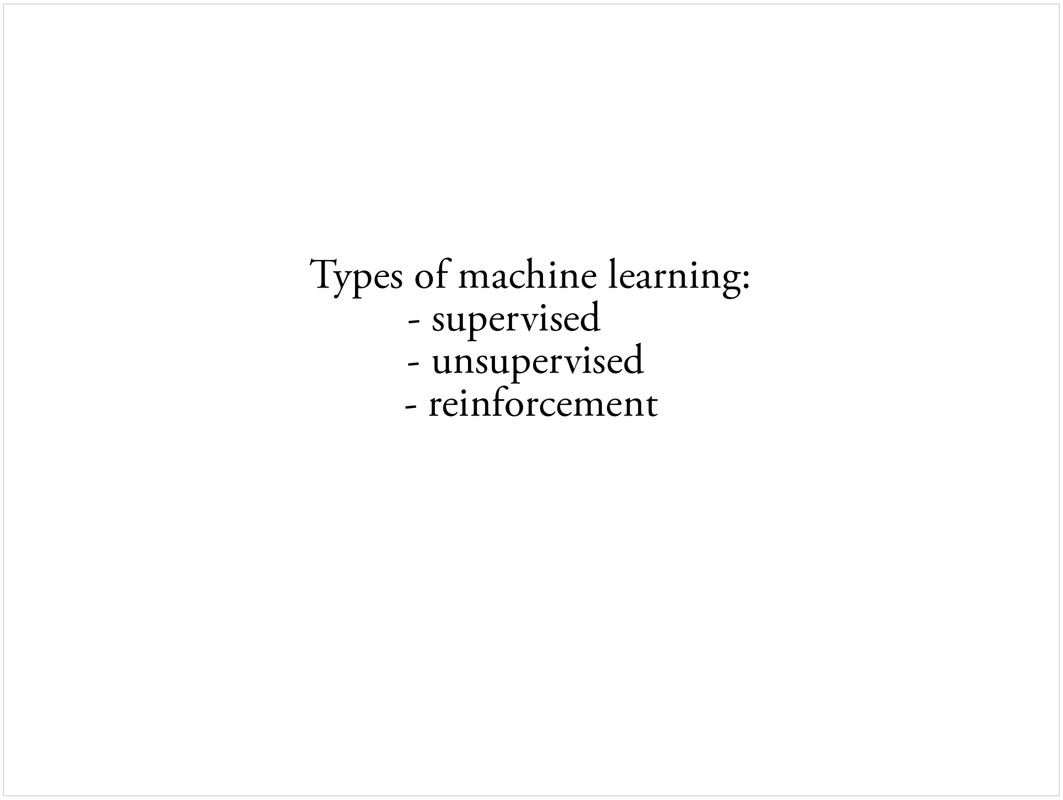
Machine learning is about extracting knowledge from data. It is closely related to statistics and optimization.

What distinguishes machine learning is that it is very focused on prediction.

We want to learn from a large dataset how to make decisions based on future observations.

You could say that the input to a machine learning program is the dataset, and the output is a program that can make decisions on future observations.

Machine learning is really widely used now, and I want to give you some examples that you probably all saw already today.



Types of machine learning:

- supervised
- unsupervised
- reinforcement

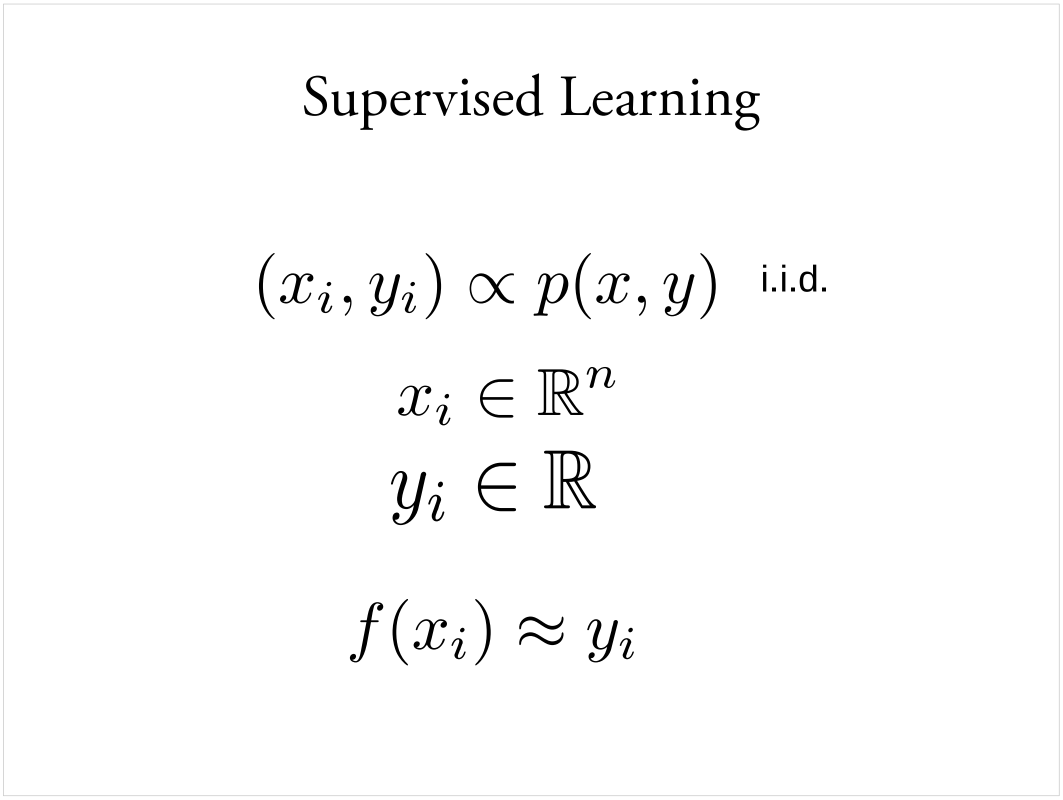
There are three main branches of machine learning.

Who can name them?

They called supervised learning, unsupervised learning and reinforcement learning.

What are they?

This course will heavily focus on supervised learning, but you should be aware of the distinction. Supervised learning is also the most commonly used type in industry and research right now, though reinforcement learning becomes increasingly important.



Supervised Learning

$$(x_i, y_i) \propto p(x, y) \quad \text{i.i.d.}$$

$$x_i \in \mathbb{R}^n$$

$$y_i \in \mathbb{R}$$

$$f(x_i) \approx y_i$$

In supervised learning, the dataset we learn form is input-output pairs (x_i, y_i), where x_i is some n_dimensional input, or feature vector, and y_i is the desired output we want to learn.

Generally, we assume these samples are drawn from some unknown joint distribution p(x, y).

What does iid mean?

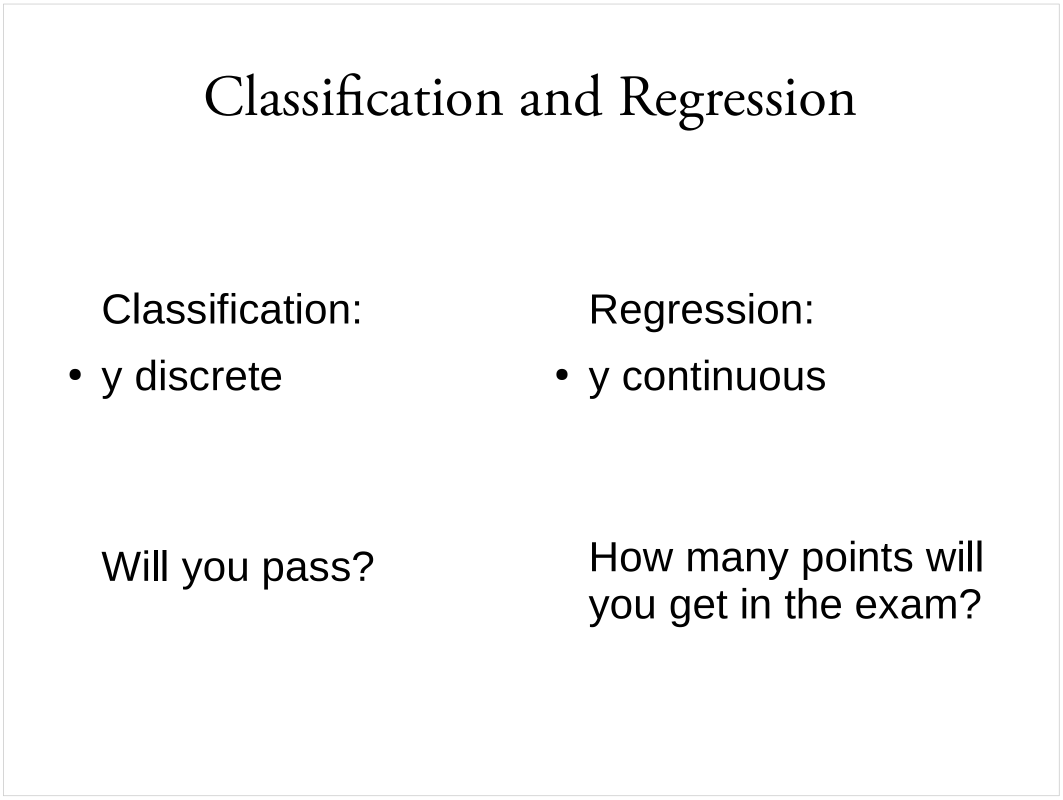
We say they are drawn iid, which stands for independent identically distributed. In other words, the x_i, y_i pairs are independent and all come from the same distribution p.

You can think of this as there being some process that goes from x_i to y_i, but that we don’t know. We write this as a probability distribution and not as a function since even if there is a real process creating y from x, this process might not be deterministic.

The goal is to learn a function f so that for new inputs x for which we don’t observe y, f(x) is close to y.

This approach is very similar to function approximation.

The name supervised comes from the fact that during learning, a supervisor gives you the correct answers y_i.



Classification and Regression

Classification:

- y discrete

Will you pass?

Regression:

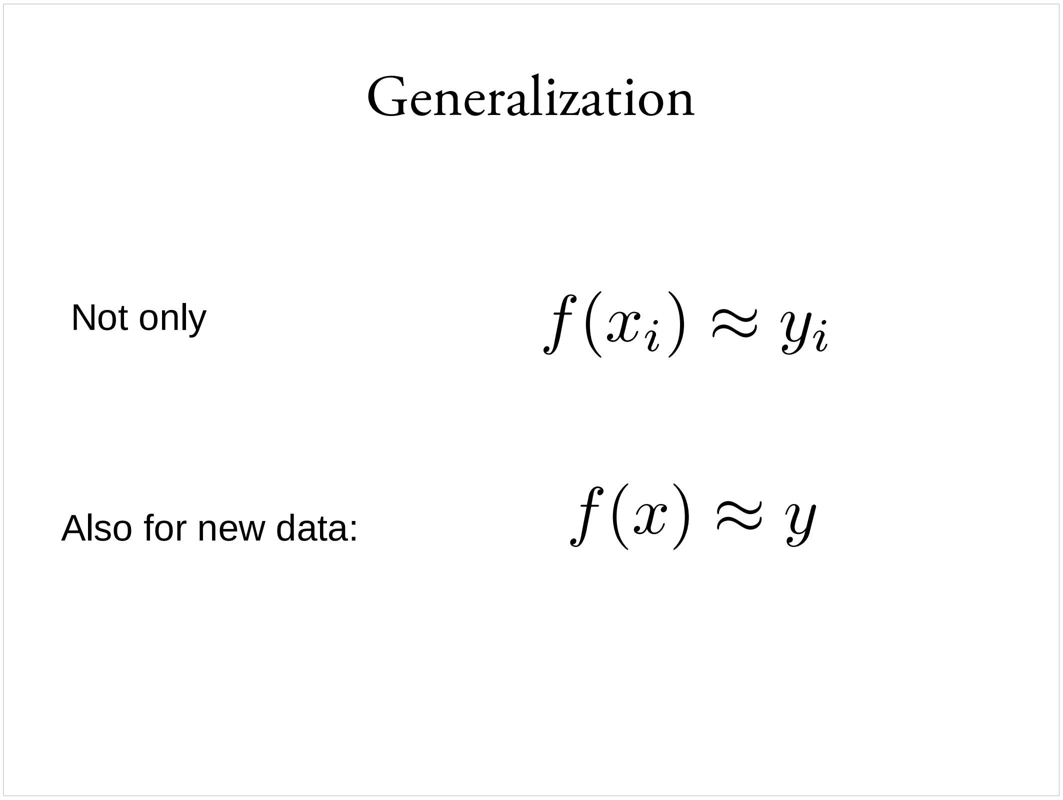
- y continuous

How many points will you get in the exam?

So getting back to supervised learning, there are two basic kinds, called classification and regression.
The difference is quite simple, if y is continuous, then it's regression, and if y is discrete, it's classification.

That's pretty simple, still let me give an example.
If I want to predict whether a student will pass the class, it's a classification problem. There are two possible answers, "yes" and "no".
If I want to predict how many points a student gets on an exam, it's a regression problem, there is a continuous, gradual output space.

There are generalizations of this where we try to predict more than one variable, but we won't go into that in this course. The main reason this distinction is important is because the way we measure how good a prediction is is very different for the two.



Generalization

Not only $f(x_i) \approx y_i$

Also for new data: $f(x) \approx y$

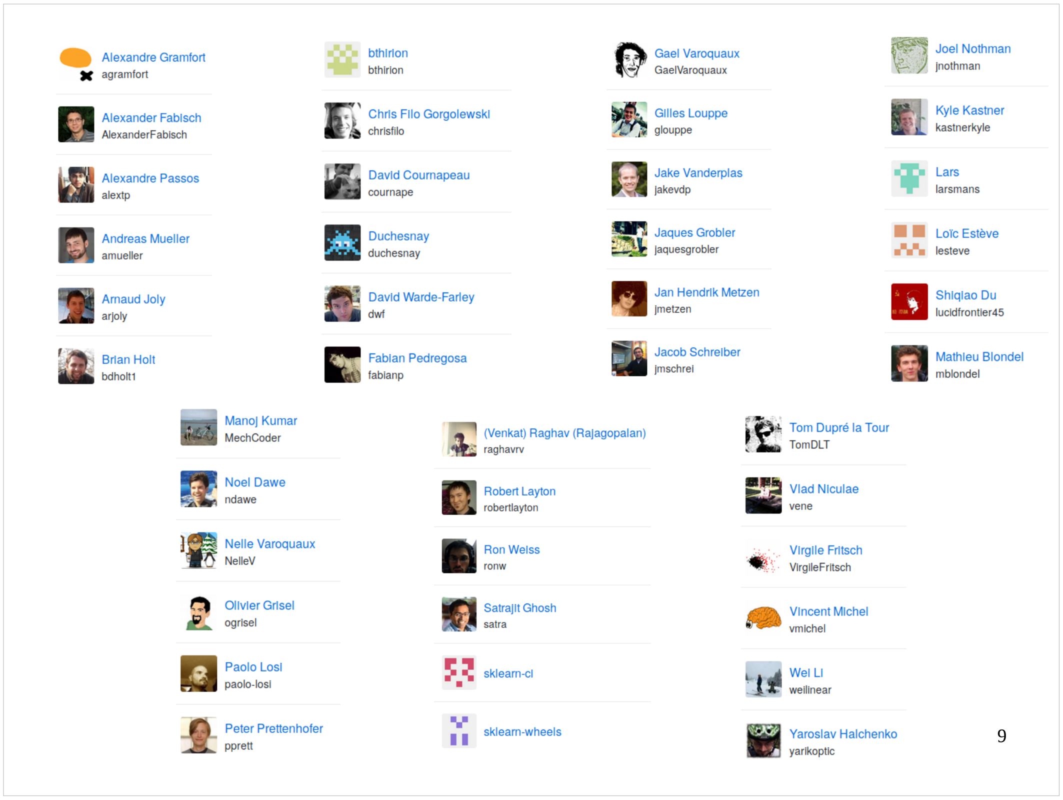
For both regression and classification, it’s important to keep in mind the concept of generalization.

Let’s say we have a regression task. We have features, that is data vectors x_i and targets y_i drawn from a joint distribution.

We now want to learn a function f, such that f(x) is approximately y, not on this training data, but on new data drawn from this distribution. This is what’s called generalization, and this is a core distinction to function approximation. In principle we don’t care about how well we do on x_i, we only care how well we do on new samples from the distribution

We’ll go into much more detail about generalization in two weeks, when we dive into supervised learning.

Classification
Regression
Clustering
Semi-Supervised Learning
Feature Selection
Feature Extraction
Manifold Learning
Dimensionality Reduction
Kernel Approximation
Hyperparameter Optimization
Evaluation Metrics
Out-of-core learning
……



Alexandre Gramfort
agramfort

Alexander Fabisch
AlexanderFabisch

Alexandre Passos
alextp

Andreas Mueller
amueller

Arnaud Joly
arjoly

Brian Holt
bdholt1

bthirion
bthirion

Chris Filo Gorgolewski
chrisfilo

David Cournapeau
cournape

Duchesnay
duchesnay

David Warde-Farley
dwf

Fabian Pedregosa
fabianp

Gael Varoquaux
GaelVaroquaux

Gilles Louppe
glouppe

Jake Vanderplas
jakevdp

Jaques Grobler
jaquesgrobler

Jan Hendrik Metzen
jmetzen

Jacob Schreiber
jmschrei

Joel Nothman
jnothman

Kyle Kastner
kastnerkyle

Lars
larsmans

Loïc Estève
lesteve

Shiqiao Du
lucidfrontier45

Mathieu Blondel
mblondel

Manoj Kumar
MechCoder

Noel Dawe
ndawe

Nelle Varoquaux
NelleV

Olivier Grisel
ogrisel

Paolo Losi
paolo-losi

Peter Prettenhofer
pprett

(Venkat) Raghav (Rajagopalan)
raghavrv

Robert Layton
robertlayton

Ron Weiss
ronw

Satrajit Ghosh
satra

sklearn-ci

sklearn-wheels

Tom Dupré la Tour
TomDLT

Vlad Niculae
vene

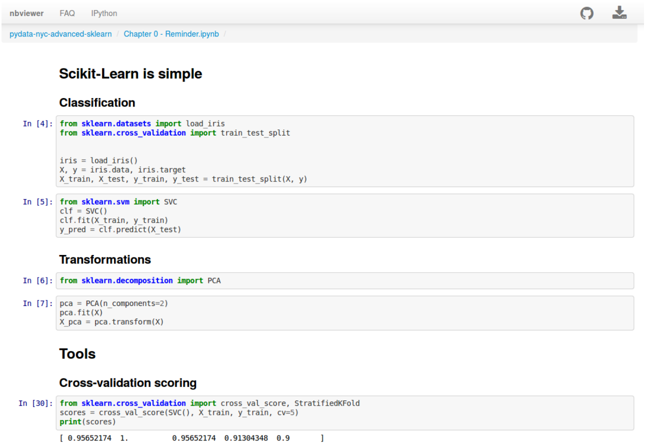
Virgile Fritsch
VirgileFritsch

Vincent Michel
vmichel

Wei Li
weilinear

Yaroslav Halchenko
yarikoptic

Get the notebooks!



Scikit-Learn is simple

Classification

```
In [4]: from sklearn.datasets import load_iris
        from sklearn.cross_validation import train_test_split


        iris = load_iris()
        X, y = iris.data, iris.target
        X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
In [5]: from sklearn.svm import SVC
        clf = SVC()
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
```

Transformations

```
In [6]: from sklearn.decomposition import PCA
```

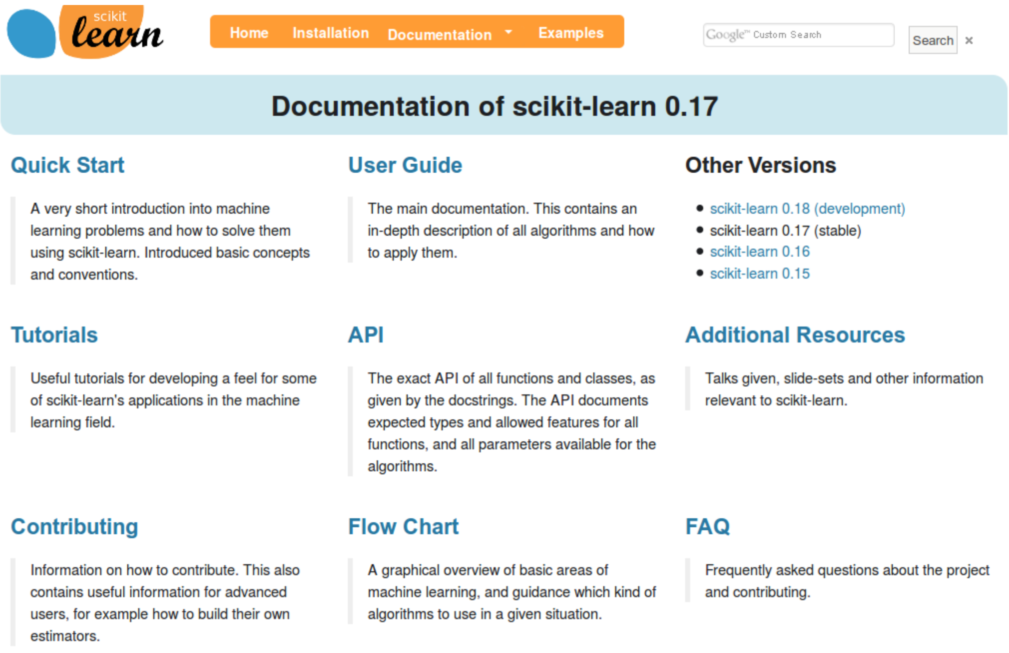
```
In [7]: pca = PCA(n_components=2)
        pca.fit(X)
        X_pca = pca.transform(X)
```

Tools

Cross-validation scoring

```
In [30]: from sklearn.cross_validation import cross_val_score, StratifiedKFold
         scores = cross_val_score(SVC(), X_train, y_train, cv=5)
         print(scores)

         [ 0.95652174  1.          0.95652174  0.91304348  0.9       ]
```

https://github.com/amueller/ml-training-intro



Documentation of scikit-learn 0.17

Quick Start

A very short introduction into machine learning problems and how to solve them using scikit-learn. Introduced basic concepts and conventions.

User Guide

The main documentation. This contains an in-depth description of all algorithms and how to apply them.

Other Versions

- scikit-learn 0.18 (development)
- scikit-learn 0.17 (stable)
- scikit-learn 0.16
- scikit-learn 0.15

Tutorials

Useful tutorials for developing a feel for some of scikit-learn's applications in the machine learning field.

API

The exact API of all functions and classes, as given by the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

Additional Resources

Talks given, slide-sets and other information relevant to scikit-learn.

Contributing

Information on how to contribute. This also contains useful information for advanced users, for example how to build their own estimators.

Flow Chart

A graphical overview of basic areas of machine learning, and guidance which kind of algorithms to use in a given situation.

FAQ

Frequently asked questions about the project and contributing.

`http://scikit-learn.org/`

Representing Data

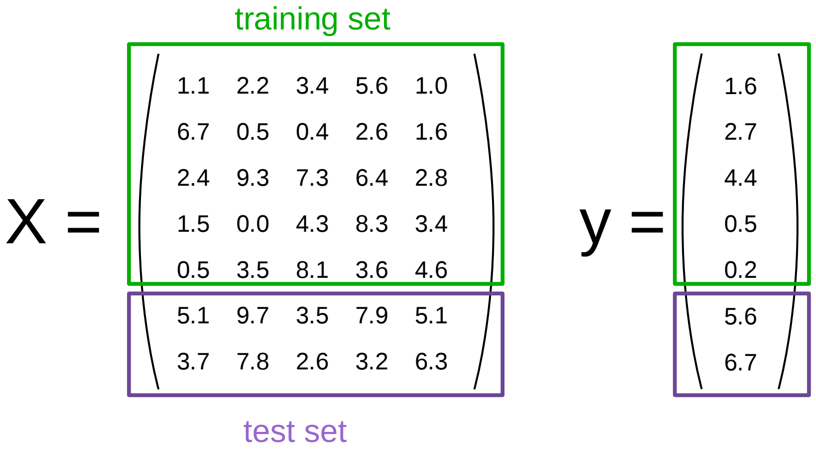
one sample

$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix} \qquad y = \begin{pmatrix} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 6.7 \end{pmatrix}$$

one feature

outputs / labels

Training and Testing Data

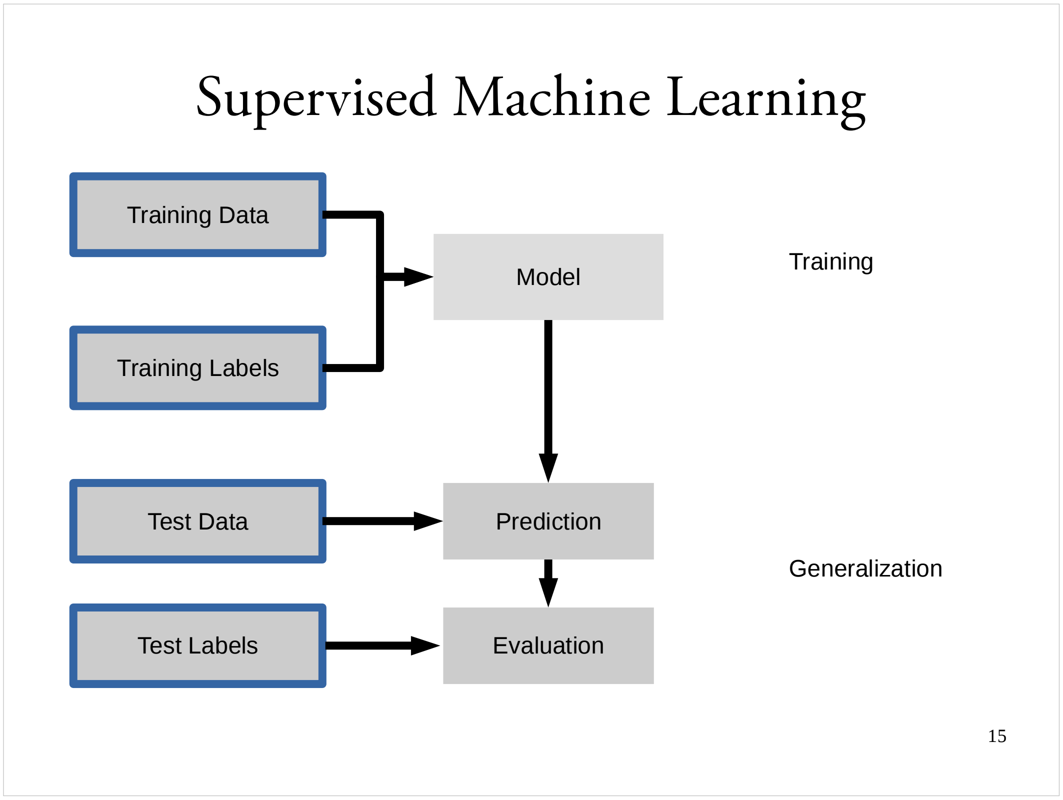


training set

$$X = \left(\begin{array}{ccccc} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ \hline 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{array}\right) \qquad y = \left(\begin{array}{c} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ \hline 5.6 \\ 6.7 \end{array}\right)$$

test set

IPython Notebook:
Part 0 – Data Loading



Supervised Machine Learning

Training Data

Training Labels

Model

Training

Test Data

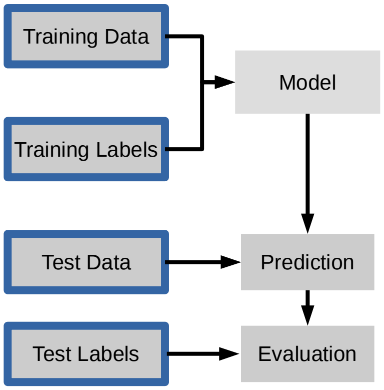
Prediction

Generalization

Test Labels

Evaluation

```
clf = RandomForestClassifier()

clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```



Training Data

Training Labels

Model

Test Data

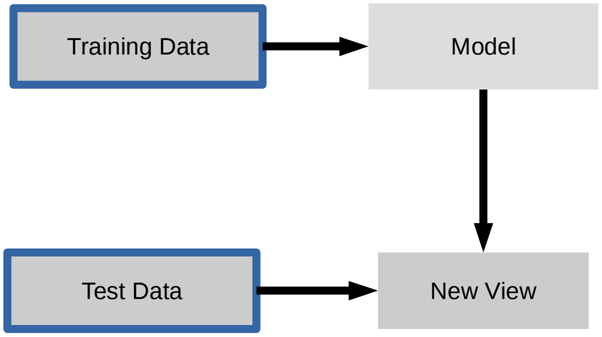
Prediction

Test Labels

Evaluation

IPython Notebook:
Part 1 - Introduction to Scikit-learn

Unsupervised Machine Learning



Training Data

Model

Test Data

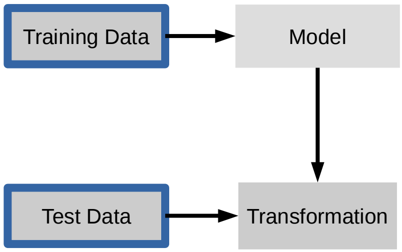
New View

Unsupervised Transformations

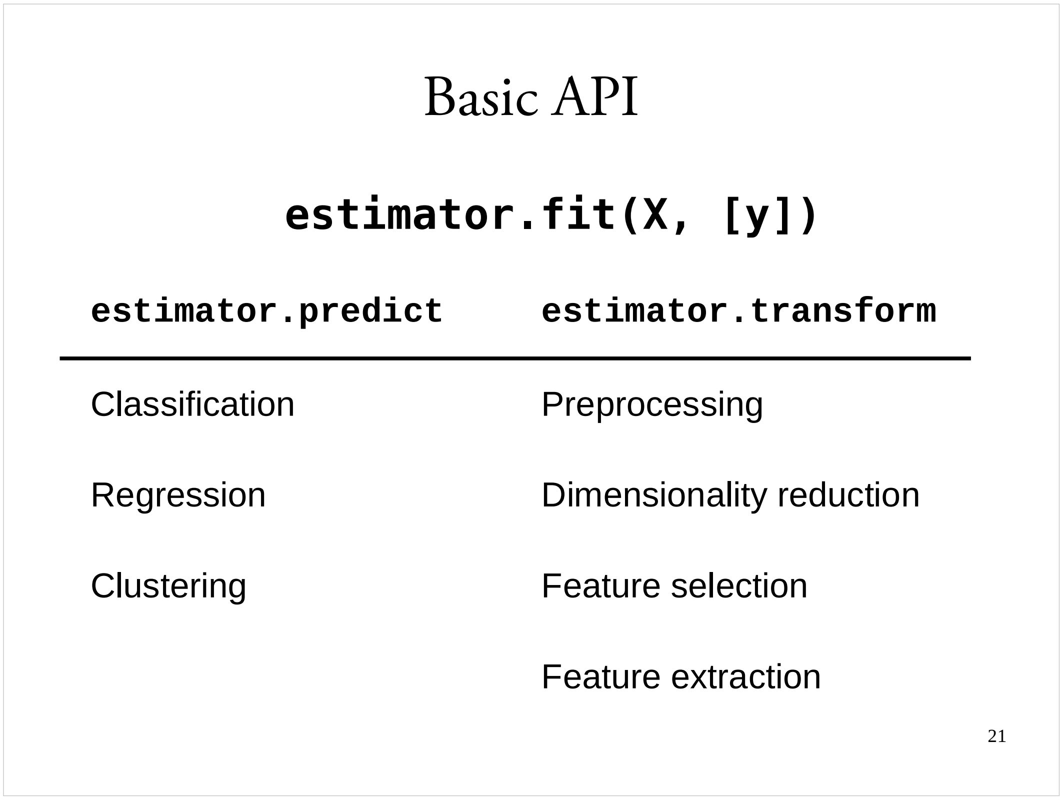
```
pca = PCA()
```

```
pca.fit(X_train)
```

```
X_new = pca.transform(X_test)
```



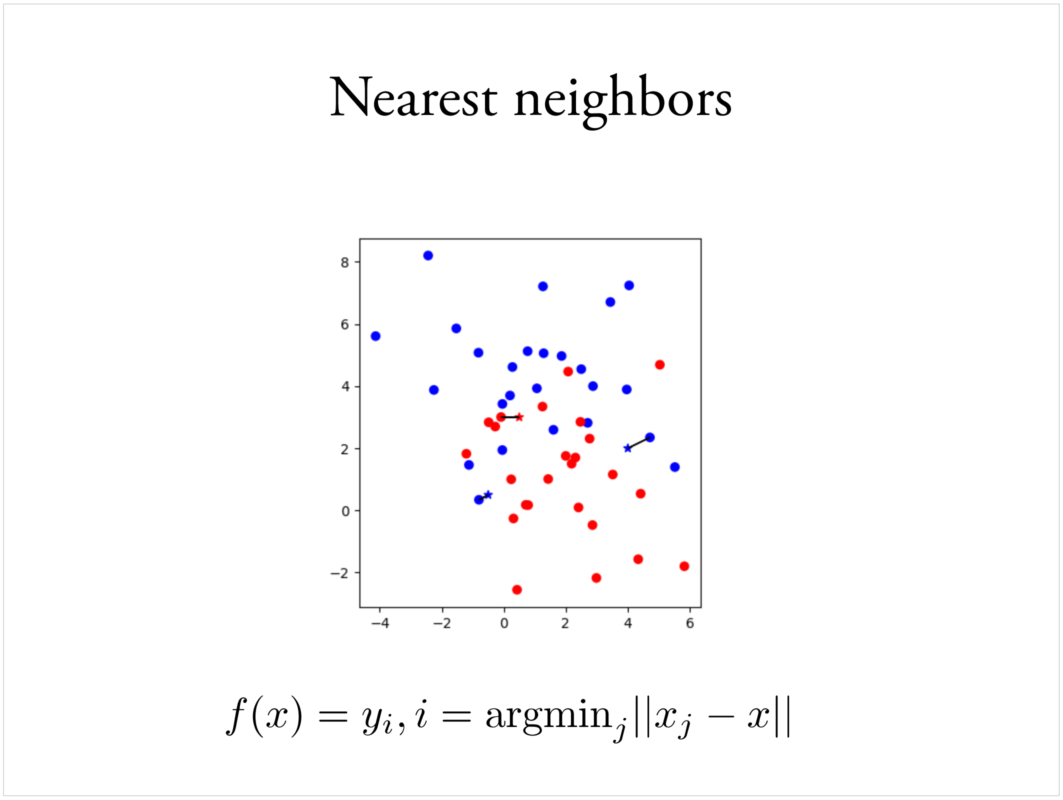
IPython Notebook:
Part 2 – Unsupervised Transformers



Basic API

`estimator.fit(X, [y])`

`estimator.predict`	**`estimator.transform`**
Classification	Preprocessing
Regression	Dimensionality reduction
Clustering	Feature selection
	Feature extraction



Nearest neighbors

$$f(x) = y_i, i = \text{argmin}_j ||x_j - x||$$

Let’s say we have this two-class classification dataset here, with two features, one on the x axis and one on the y axis.

And we have three new points as marked by the stars here.

If I make a prediction using a one nearest neighbor classifier, what will it predict?

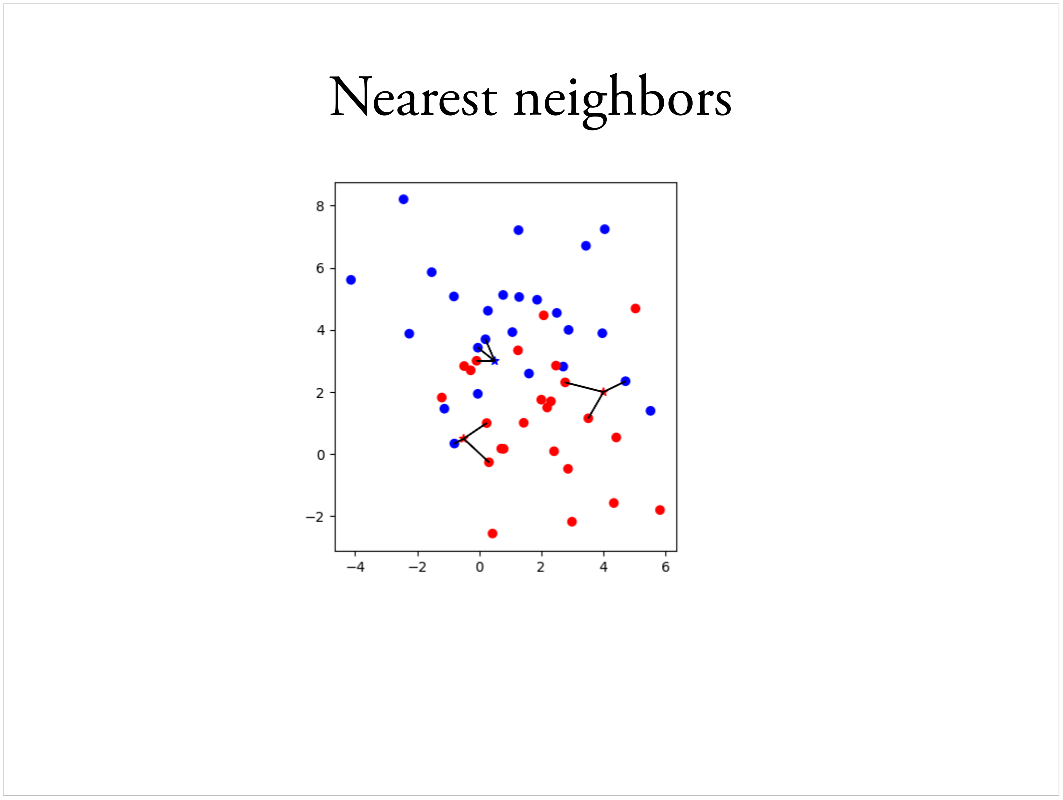
It will predict the label of the closest data point in the training set.

That is basically the simplest machine learning algorithm I can come up with.

Here’s the formula:

the prediction for a new x is the y_i so that x_i is the closest point in the training set.

Ok, so now how do we find out whether this model is any good?



Nearest neighbors

So this was the predictions as made by one-nearest neighbor.

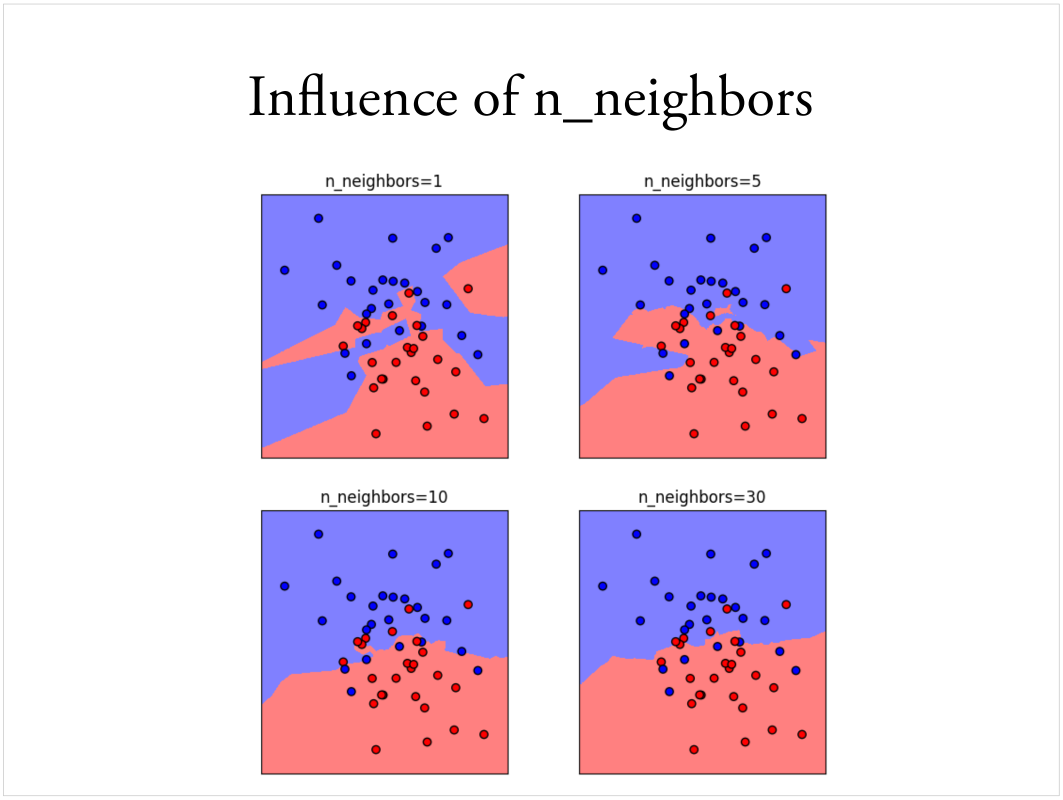
But we can also consider more neighbors, for example three. Here is the three nearest neighbors for each of the points and the corresponding labels.

We can then make a prediction by considering the majority among these three neighbors.

And as you can see, in this case all the points changed their labels! (I was actually quite surprised when I saw that, I just picked some points at random).

Clearly the number of neighbors that we consider matters a lot. But what is the right number?

The is a problem you’ll encounter a lot in machine learning, the problem of tuning parameters of the model, also called hyper-parameters, which can not be learned directly from the data.



Influence of n_neighbors

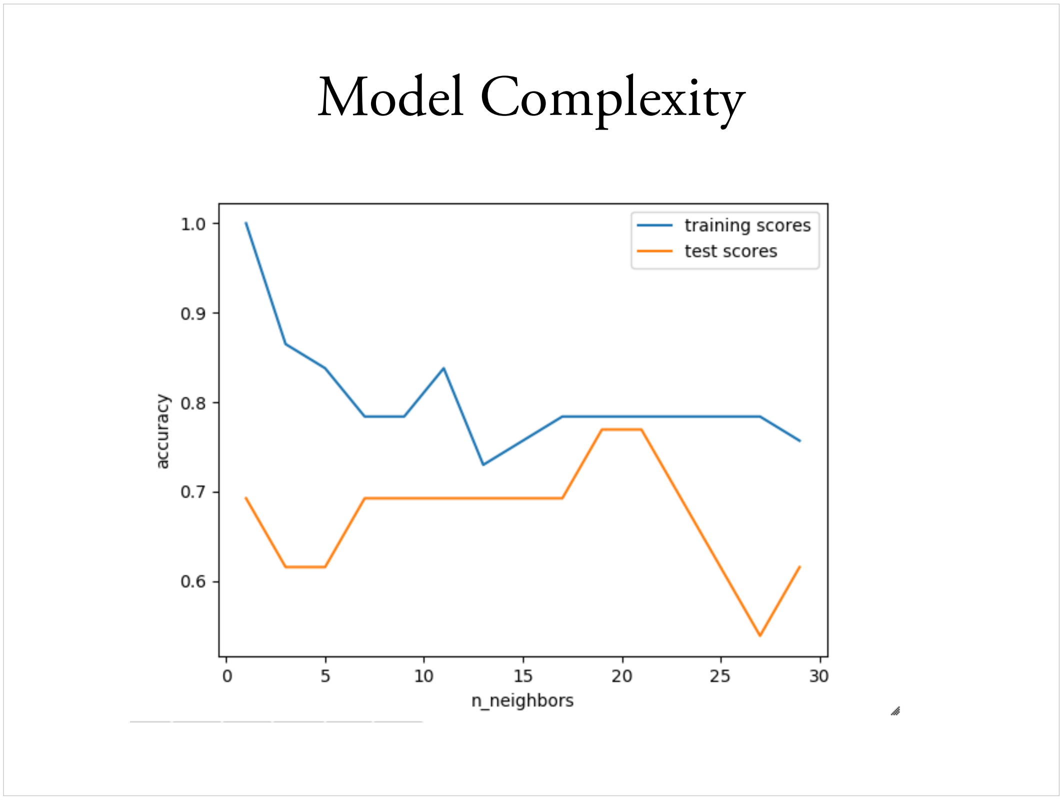
Here’s an overview of how the classification changes if we consider different numbers of neighbors.

You can see as red and blue circles the training data. And the background is colored according to which class a datapoint would be assigned to for each location.

For one neighbor, you can see that each point in the training set has a little area around it that would be classified according to it’s label. This means all the training points would be classified correctly, but it leads to a very complex shape of the decision boundary.

If we increase the number of neighbors, the boundary between red and blue simplifies, and with 40 neighbors we mostly end up with a line.

This also means that now many of the training data points would be labeled incorrectly.



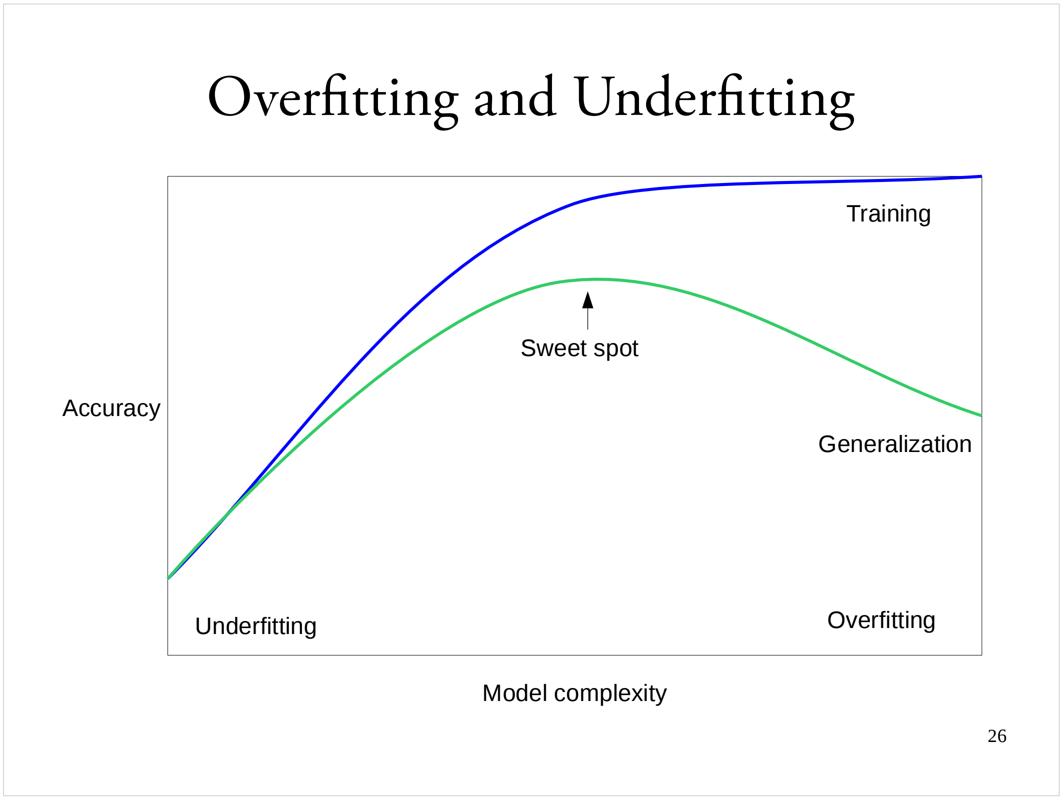
Model Complexity

We can look at this in more detail by comparing training and test set scores for the different numbers of neighbors.

Here, I did a random 75%/25% split again. This is a very noisy plot as the dataset is very small and I only did a random split, but you can see a trend here.

You can see that for a single neighbor, the training score is 1 so perfect accuracy, but the test score is only 70%. If we increase the number of neighbors we consider, the training score goes down, but the test score goes up, with an optimum at 19 and 21, but then both go down again.

This is a very typical behavior, that I sketched in a schematic for you.



Overfitting and Underfitting

This chart has accuracy on the y axis, and an abstract concept of model complexity on the x axis.

If we make our machine learning models more complex, we will get better training set accuracy, as the model will be able to capture more of the variations in the data.

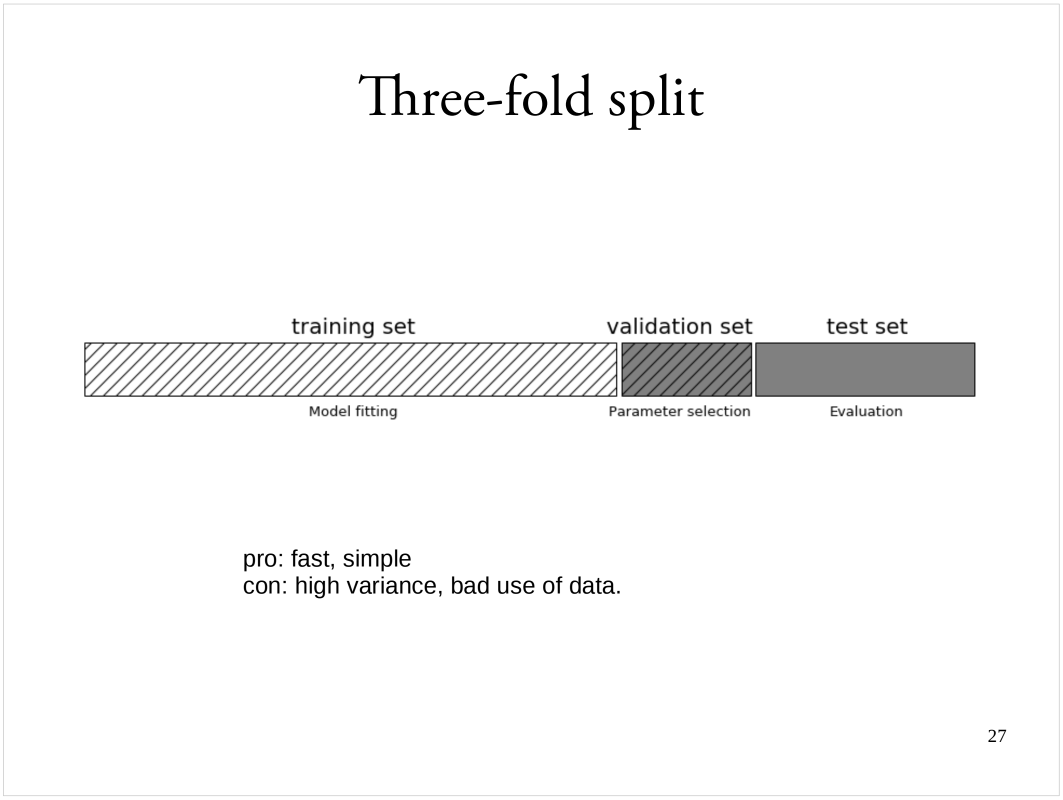
But if we look at the generalization performance, we get a different story. If the model complexity is too low, the model will not be able to capture the main trends, and a more complex model means better generalization.

However, if we make the model too complex, generalization performance drops again, because we basically learn to memorize the dataset.

If we use too simple a model, this is often called underfitting, while if we use to complex a model, this is called overfitting. And somewhere in the middle is a sweet spot.

Most models have some way to tune model complexity, and we’ll see many of them in the next couple of weeks.

So going back to nearest neighbors, what parameters correspond to high model complexity and what to low model complexity? high n_neighbors = low complexity!



Three-fold split

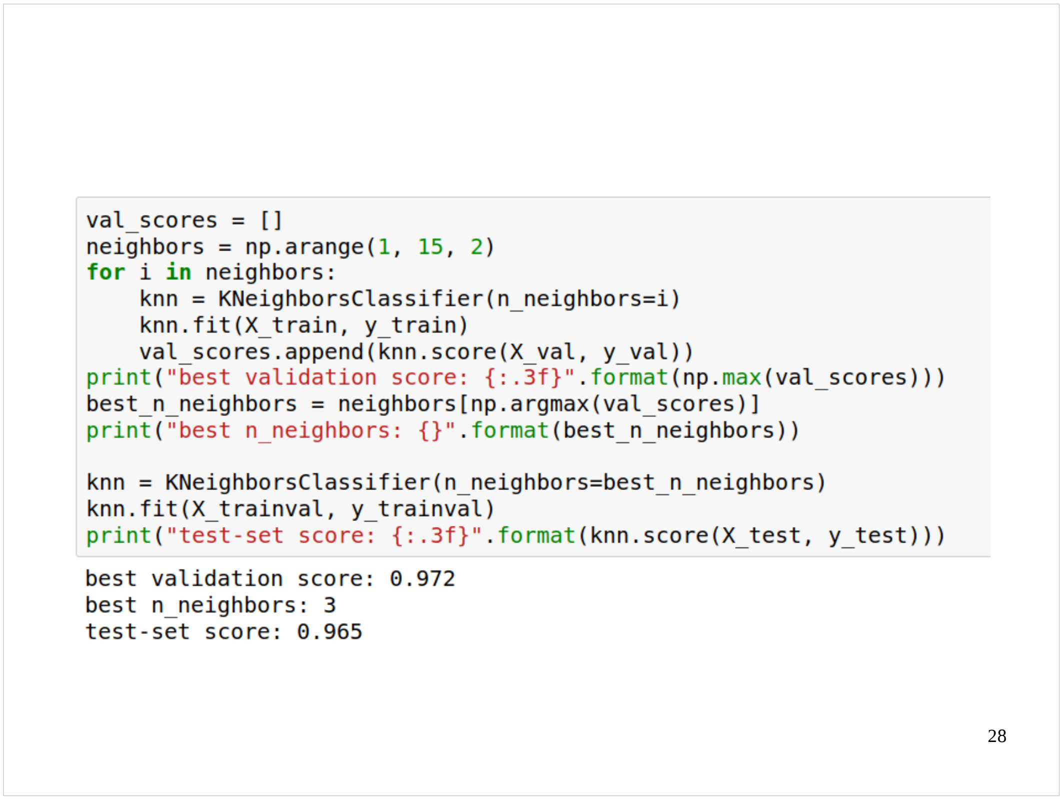
pro: fast, simple
con: high variance, bad use of data.

The simplest way to combat this overfitting to the test set is by using a three-fold split of the data, into a training, a validation and a test set.

We use the training set for model building, the validation set for parameter selection and the test set for a final evaluation of the model.

So how many models should you try out on the test set? Only one! Ideally use use the test-set exactly once, otherwise you make a multiple hypothesis testing error!

What are downsides of this? We lose a lot of data for evaluation, and the results depend on the particular sampling.



```
val_scores = []
neighbors = np.arange(1, 15, 2)
for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    val_scores.append(knn.score(X_val, y_val))
print("best validation score: {:.3f}".format(np.max(val_scores)))
best_n_neighbors = neighbors[np.argmax(val_scores)]
print("best n_neighbors: {}".format(best_n_neighbors))

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_trainval, y_trainval)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

```
best validation score: 0.972
best n_neighbors: 3
test-set score: 0.965
```

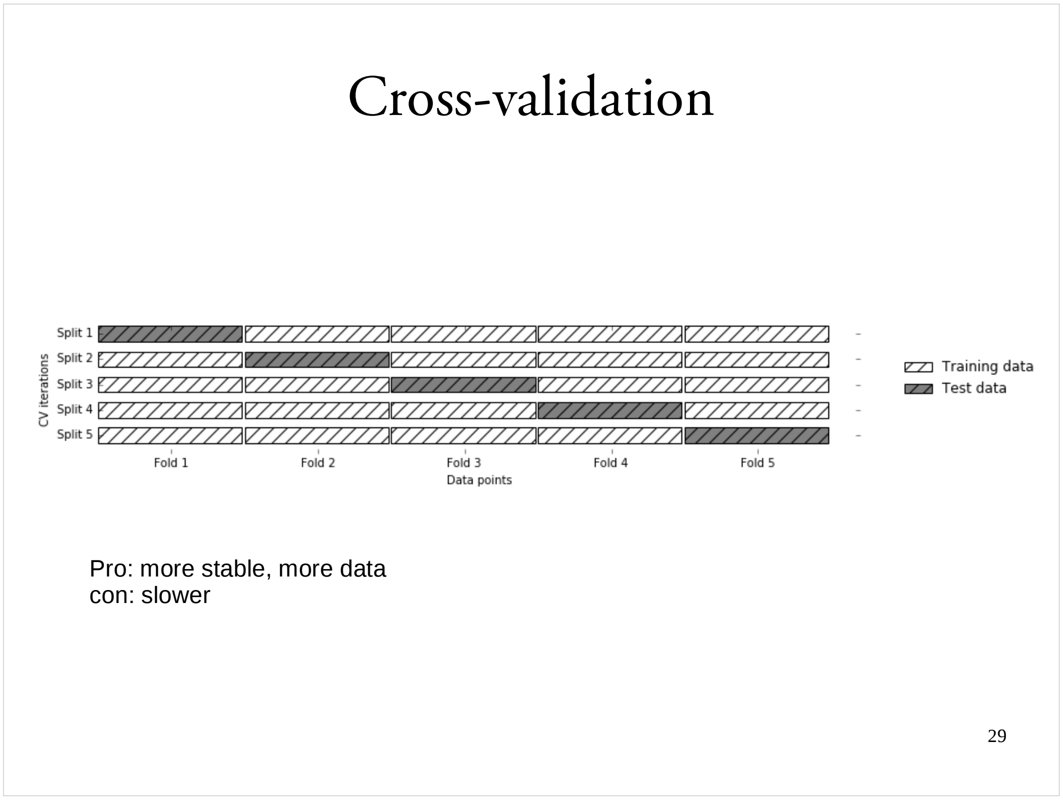
Here is an implementation of the three-fold split for selecting the number of neighbors.

For each number of neighbors that we want to try, we build a model on the training set, and evaluate it on the validation set.

We then pick the best validation set score, here that’s 97.2%, achieved when using three neighbors.

We then retrain the model with this parameter, and evaluate on the test set.

Do you have any idea how to make this more robust?



Cross-validation

Pro: more stable, more data
con: slower

The answer is of course cross-validation. In cross-validation, you split your data into multiple folds, usually 5 or 10, and built multiple models.

You start by using fold1 as the test data, and the remaining ones as the training data. You build your model on the training data, and evaluate it on the test fold.

For each of the splits of the data, you get a model evaluation and a score. In the end, you can aggregate the scores, for example by taking the mean.

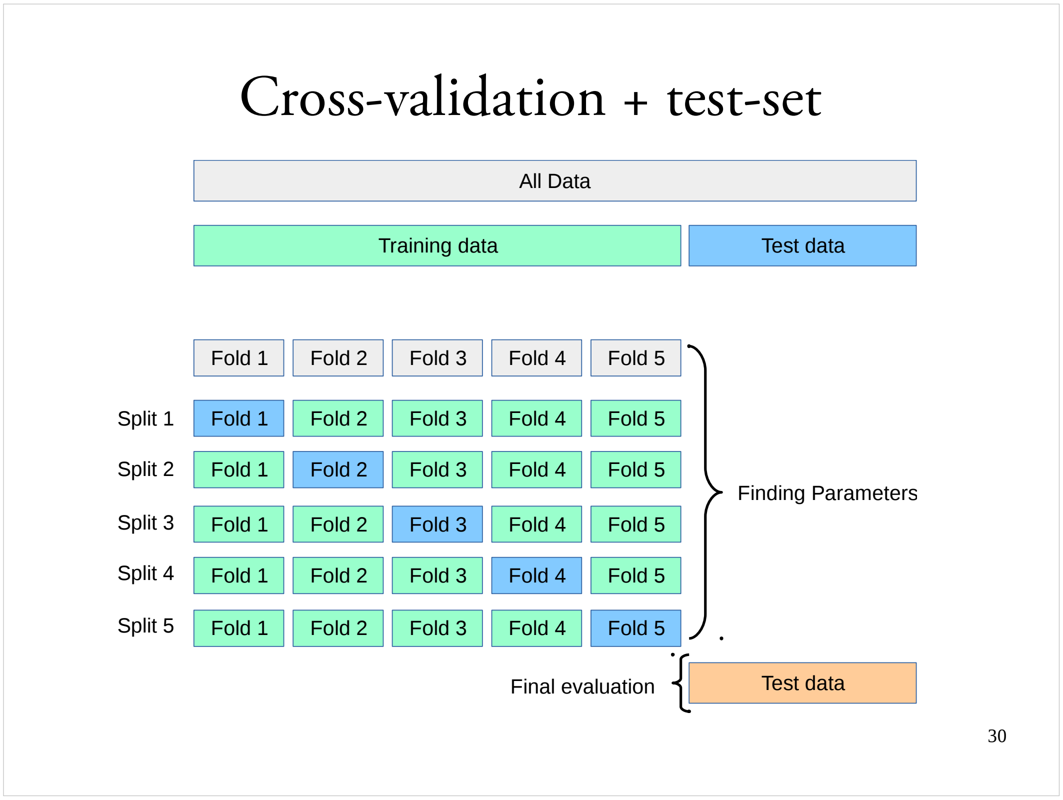
What are the pros and cons of this?

Each data point is in the test-set exactly once!

Takes 5 or 10 times longer!

Better data use (larger training sets).

Does that solve all problems? No, it replaces only one of the splits, usually the inner one!



Cross-validation + test-set

All Data

Training data | Test data

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

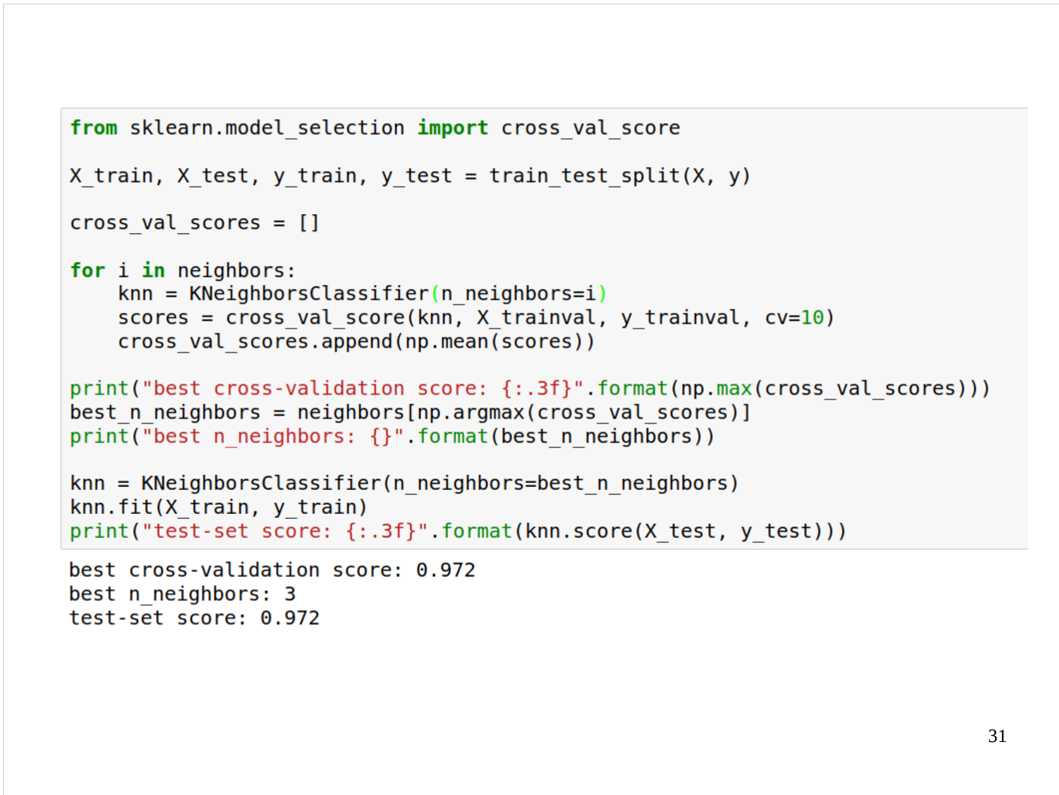
Finding Parameters

Final evaluation — Test data

Here is how the workflow looks like when we are using five-fold cross-validation together with a test-set split for adjusting parameters.

We start out by splitting of the test data, and then we perform cross-validation on the training set.

Once we found the right setting of the parameters, we retrain on the whole training set and evaluate on the test set.



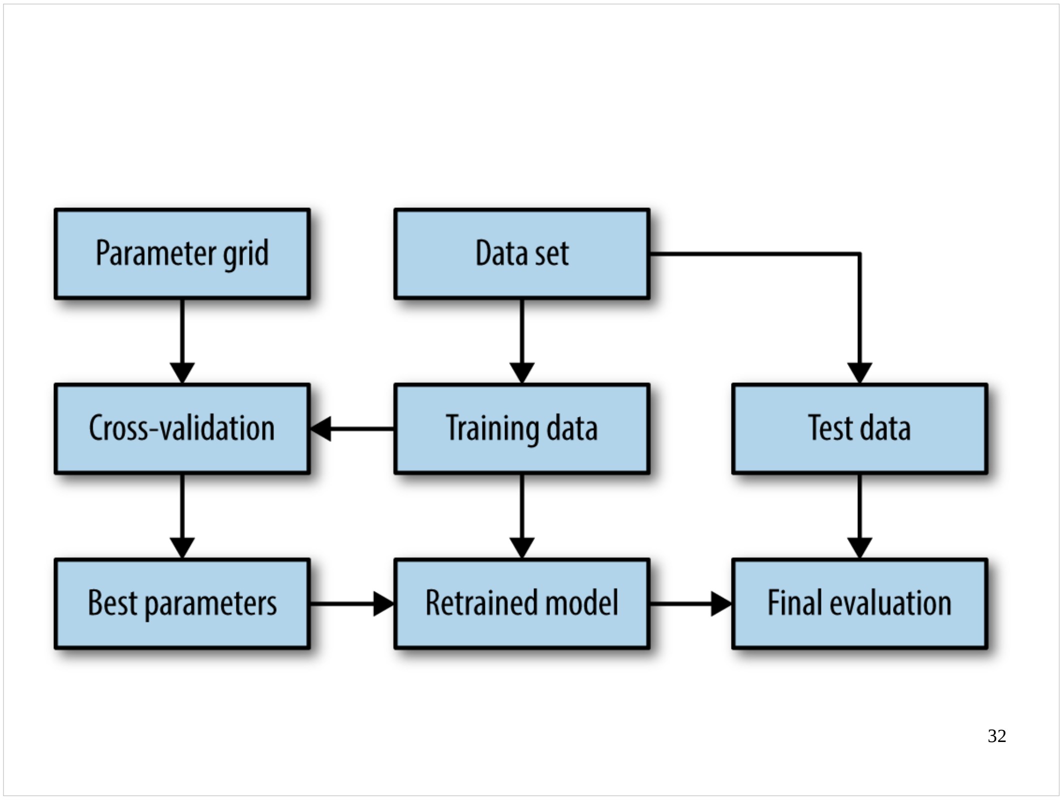
```python
from sklearn.model_selection import cross_val_score

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

cross_val_scores = []

for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn, X_trainval, y_trainval, cv=10)
    cross_val_scores.append(np.mean(scores))

print("best cross-validation score: {:.3f}".format(np.max(cross_val_scores)))
best_n_neighbors = neighbors[np.argmax(cross_val_scores)]
print("best n_neighbors: {}".format(best_n_neighbors))

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_train, y_train)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

```
best cross-validation score: 0.972
best n_neighbors: 3
test-set score: 0.972
```

Here is an implementation of this for k nearest neighbors.

We split the data, then we iterate over all parameters and for each of them we do cross-validation.

We had seven different values of n_neighbors, and we are running 10 fold cross-validation. How many models to we train in total?
10 * 7 + 1 = 71 (the one is the final model)



Parameter grid

Data set

Cross-validation

Training data

Test data

Best parameters

Retrained model

Final evaluation

Here is a conceptual overview of this way of tuning parameters, we start of with the dataset and a candidate set of parameters we want to try, labeled parameter grid, for example the number of neighbors.

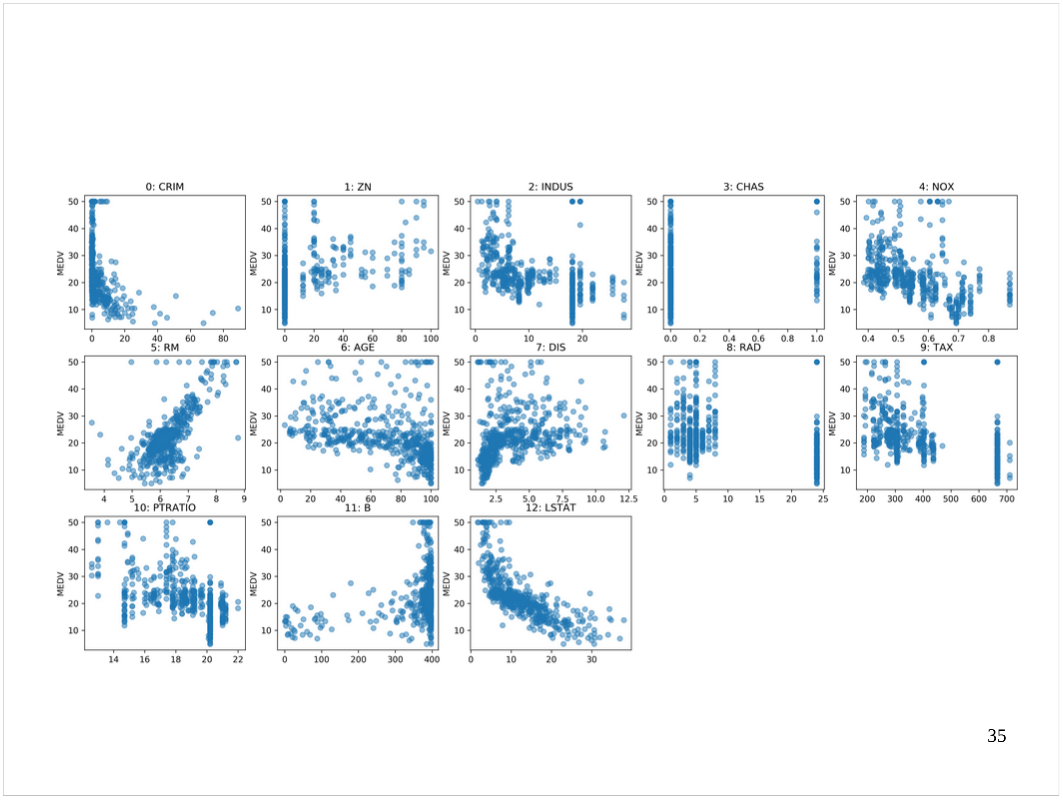
We split the dataset in to training and test set. We use cross-validation and the parameter grid to find the best parameters.

We use the best parameters and the training set to build a model with the best parameters,

and finally evaluate it on the test set.

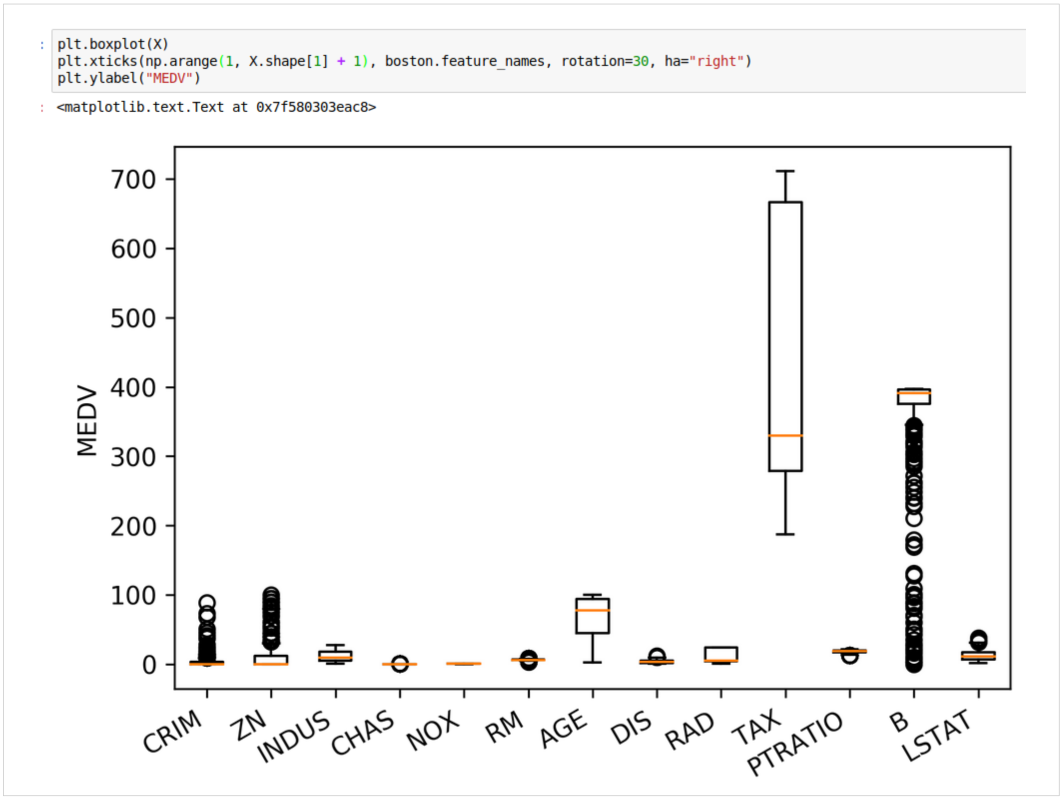
IPython Notebook:
Part 3 – Cross-validation and grid-search

Preprocessing



Let’s go back to the boston housing dataset. The idea was to predict house prices. Here are the features on the x axis and the response, so price, on the y axis.

What are some thing you can notice? (concentrated distributions, skewed distibutions, discrete variable, linear and non-linear effects, different scales)



```
plt.boxplot(X)
plt.xticks(np.arange(1, X.shape[1] + 1), boston.feature_names, rotation=30, ha="right")
plt.ylabel("MEDV")
```

```
<matplotlib.text.Text at 0x7f580303eac8>
```

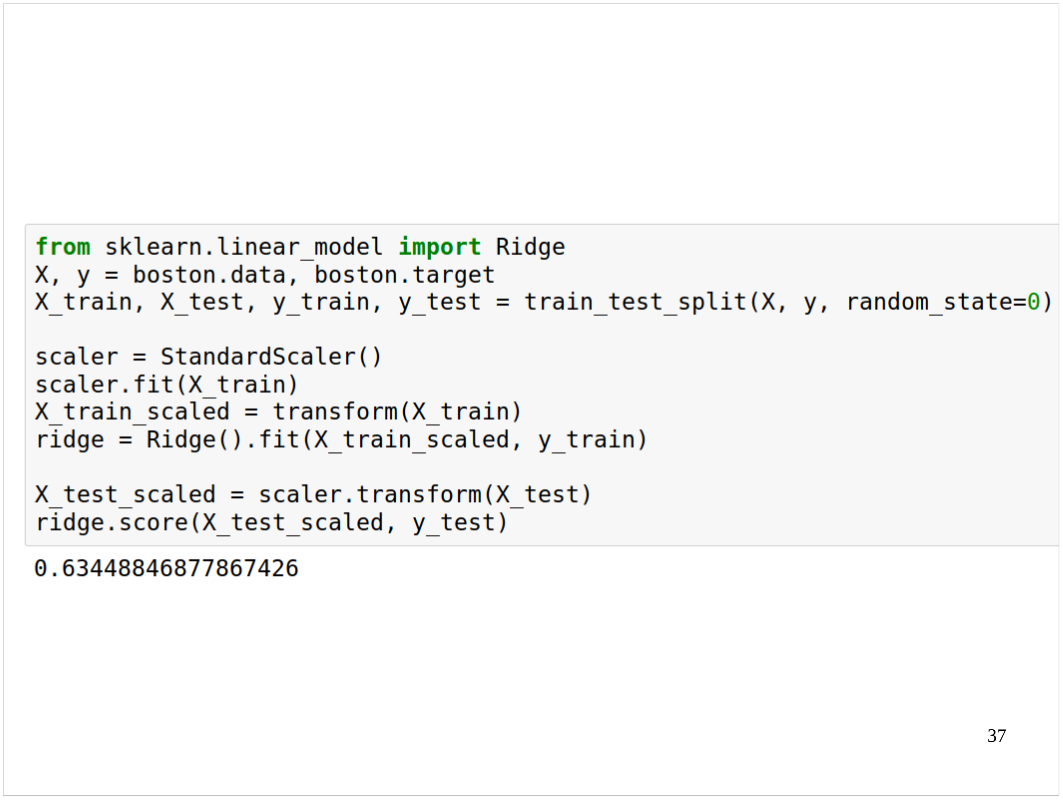
Let’s start with the different scales.

Many model want data that is on the same scale.

KNearestNeighbors: If the distance in TAX is between 300 and 400 then the distance difference in CHArS doesn’t matter!

Linear models: the different scales mean different penalty. L2 is the same for all!

We can also see non-gaussian distributions here btw!



```python
from sklearn.linear_model import Ridge
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = transform(X_train)
ridge = Ridge().fit(X_train_scaled, y_train)

X_test_scaled = scaler.transform(X_test)
ridge.score(X_test_scaled, y_test)
```

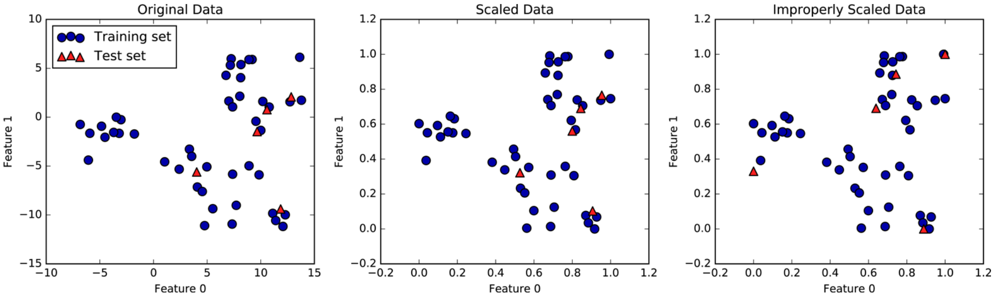
```
0.634488468778867426
```

Here’s how you do the scaling with StandardScaler in scikit-learn. Similar interface to models, but “transform” instead of “predict”. “transform” is always used when you want a new representation of the data.

Fit on training set, transform training set, fit ridge on scaled data, transform test data, score scaled test data.

The fit computes mean and standard deviation on the training set, transform subtracts the mean and the standard deviation.

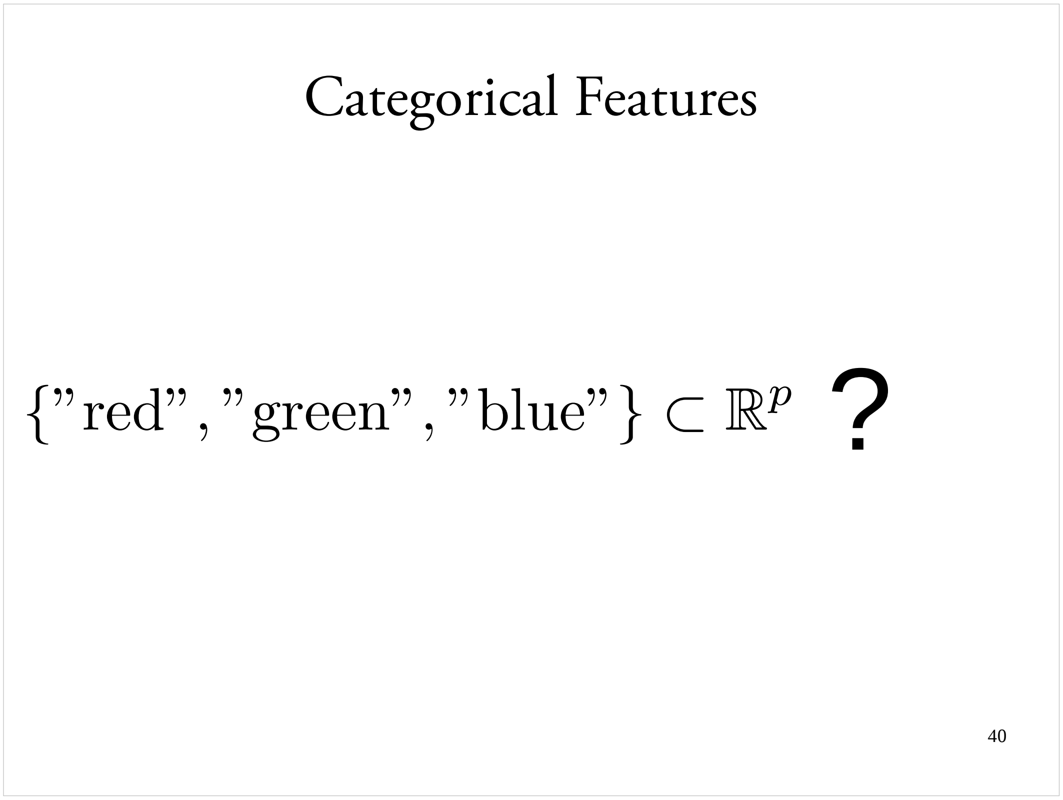
We fit on the training set and apply transform on both the training and the test set. That means the training set mean gets subtracted from the test set, not the test-set mean. That’s quite important.



Here’s an illustration why this is important using the min-max scaler. Left is the original data. Center is what happens when we fit on the training set and then transform the training and test set using this transformer. The data looks exactly the same, but the ticks changed. Now the data has a minimum of zero and a maximum of one on the training set. That’s not true for the test set, though. No particular range is ensured for the test-set. It could even be outside of 0 and 1. But the transformation is consistent with the transformation on the training set, so the data looks the same.

On the right you see what happens when you use the test-set minimum and maximum for scaling the test set. That’s what would happen if you’d fit again on the test set. Now the test set also has minimum at 0 and maximum at 1, but the data is totally distorted from what it was before. So don’t do that.

Categorical Features



Categorical Features

$\{''\mathrm{red}'', ''\mathrm{green}'', ''\mathrm{blue}''\} \subset \mathbb{R}^p$?

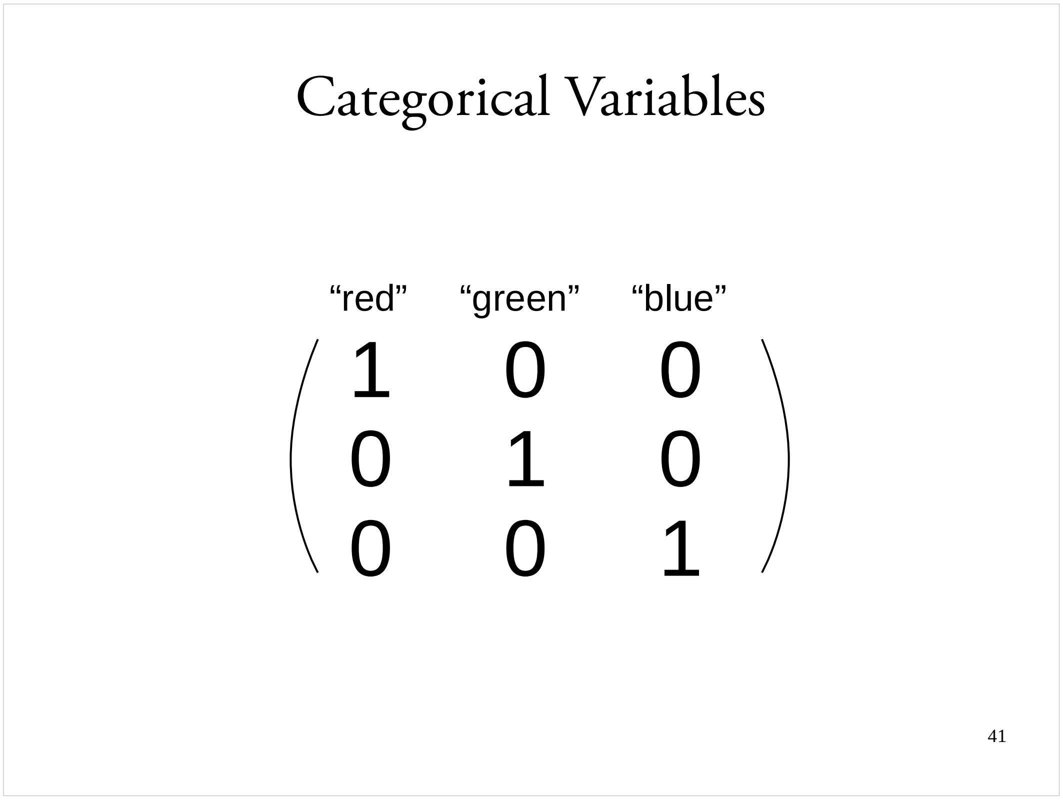
Before we can apply a machine learning algorithm, we first need to think about how we represent our data.

Earlier, I said x \in R^n. That’s not how you usually get data. Often data has units, possibly different units for different sensors, it has a mixture of continuous values and discrete values, and different measurements might be on totally different scales.

First, let me explain how to deal with discrete input variables, also known as categorical features. They come up in nearly all applications.

Let’s say you have three possible values for a given measurement, whether you used setup1 setup2 or setup3. You could try to encode these into a single real number, say 0, 1 and 2, or e, \pi, \tau.

However, that would be a bad idea for algorithms like linear regression.



Categorical Variables

$$\begin{array}{ccc} \text{“red”} & \text{“green”} & \text{“blue”} \\ \end{array}$$
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

If you encode all three values using the same feature, then you are imposing a linear relation between them, and in particular you define an order between the categories. Usually, there is no semantic ordering of the categories, and so we shouldn’t introduce one in our representation of the data.

Instead, we add one new feature for each category,

And that feature encodes whether a sample belongs to this category or not.

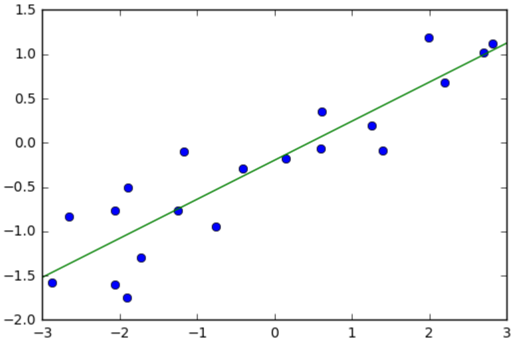
That’s called a one-hot encoding, because only one of the three features in this example is active at a time.

You could actually get away with n-1 features, but in machine learning that usually doesn’t matter.

IPython Notebook:
Part 4 – Preprocessing

Linear Models for Regression

Linear Models for Regression



$$\hat{y} = w^T \mathbf{x} + b = \sum_{i=1}^{p} w_i x_i + b$$

Linear Regression
Ordinary Least Squares

$$\hat{y} = w^T \mathbf{x} + b = \sum_{i=1}^{p} w_i x_i + b$$

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{p} ||w^T \mathbf{x}_i - y_i||^2$$

Unique solution if $\mathbf{X} = (\mathbf{x}_1, ...\mathbf{x}_n)^T$ has full rank.

Ridge Regression

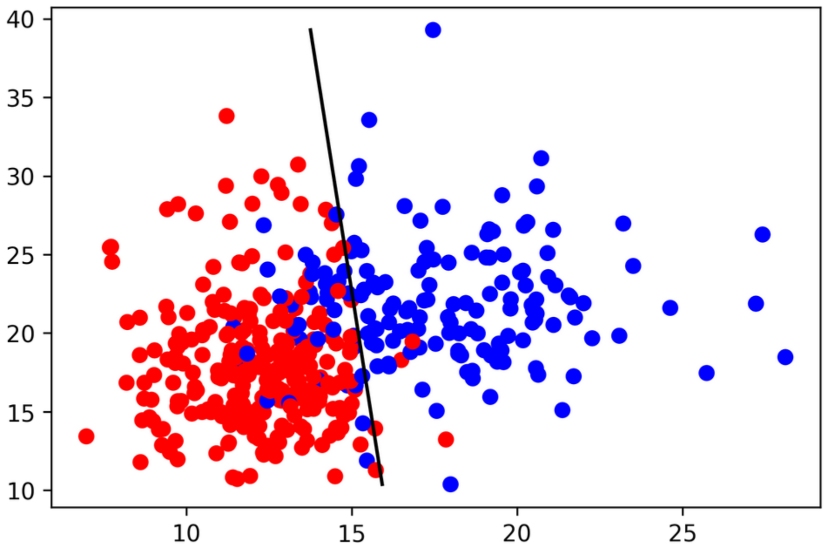
$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} ||w^T x_i - y_i||^2 + \alpha ||w||^2$$

Always has a unique solution.
Has tuning parameter alpha

IPython Notebook:
Part 5 – Linear Models for Regression

Linear Models for Classification

Linear models for **binary** classfiication

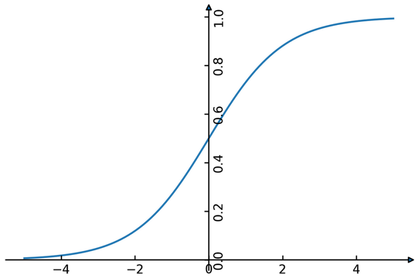


$$\hat{y} = \mathrm{sign}(w^T \mathbf{x} + b) = \mathrm{sign}(\sum_i w_i x_i + b)$$

Logistic Regression

$$\min_{w \in \mathbb{R}^p} - \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i) + 1)$$

$$p(y|\mathbf{x}) = \frac{1}{1 + e^{-w^T \mathbf{x}}}$$

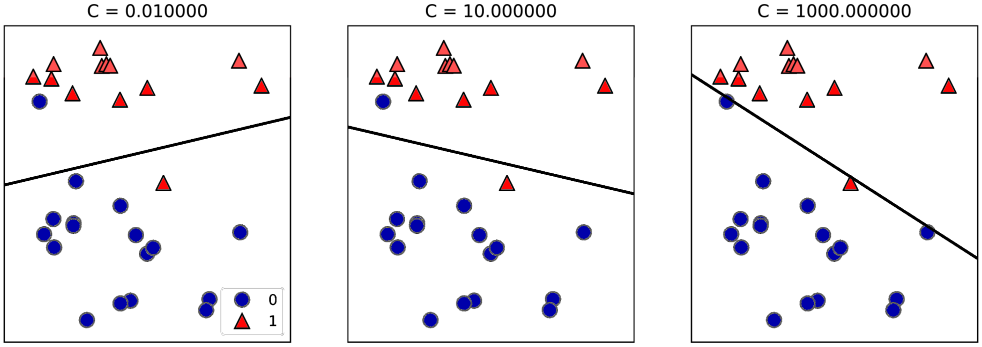


$$\hat{y} = \text{sign}(w^T \mathbf{x} + b)$$

Penalized Logistic Regression

$$\min_{w \in \mathbb{R}^p} -C \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i) + 1) + ||w||_2^2$$

C is inverse to alpha (or alpha / n_samples)


C = 0.010000
C = 10.000000
C = 1000.000000
0
1

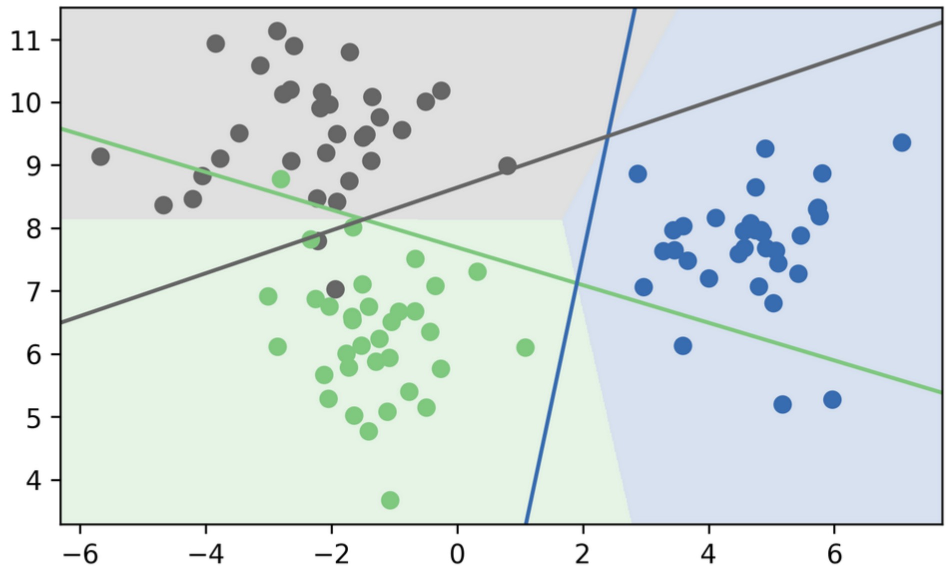
Multinomial Logistic Regression

Probabilistic multi-class model:

$$p(y = i|\mathbf{x}) = \frac{e^{-\mathbf{w}_i^T \mathbf{x}}}{\sum_{j \in Y} e^{-\mathbf{w}_j^T \mathbf{x}}}$$

$$\min_{w \in \mathbb{R}^p} - \sum_{i=1}^{n} \log(p(y = y_i | \mathbf{x}_i))$$

$$\hat{y} = \arg\max_{i \in Y} \mathbf{w}_i \mathbf{x}$$

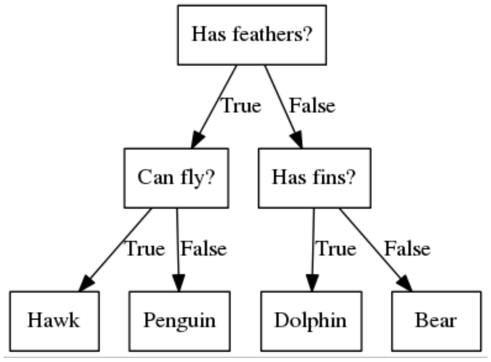


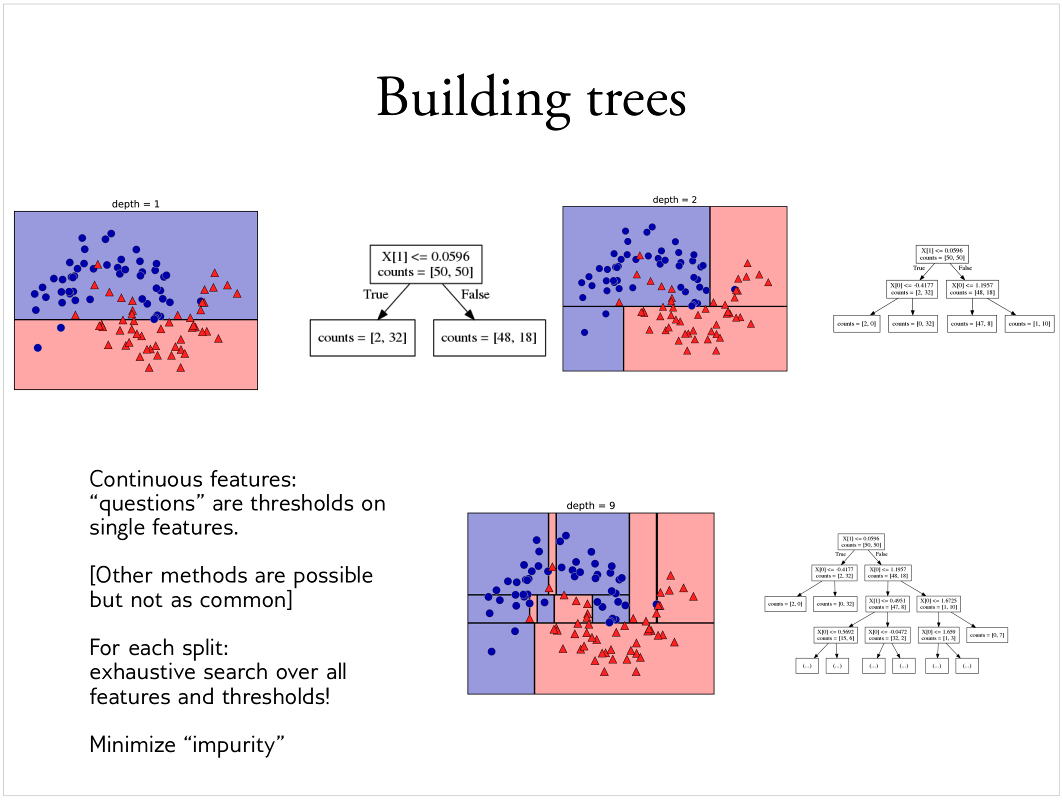
$$\hat{y} = \arg\max_{i \in Y} \mathbf{w}_i \mathbf{x}$$

IPython Notebook:
Part 6 – Linear Models for Classification

Decision Trees and Tree-based Models

Idea: series of binary questions





Building trees

Continuous features:
“questions” are thresholds on single features.

[Other methods are possible but not as common]

For each split:
exhaustive search over all features and thresholds!

Minimize “impurity”

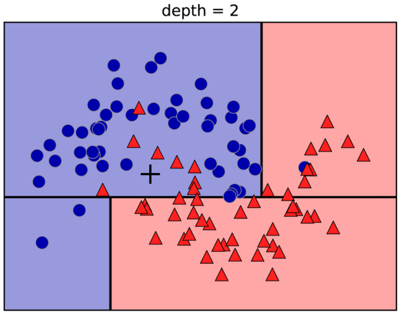
The second family of models I want to talk about is decision-tree based models.

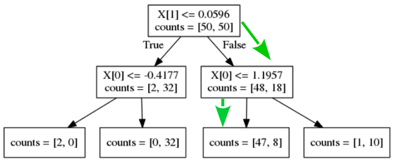
A decision tree is basically the same as a sequence of if-else branches, that ultimately lead to a decision. The point is that the question that are asked are learned, though.

A decision tree is build recursively by asking a series of questions of the form “is feature “i” greater than threshold t”. In each iteration, the question is chosen that yields the most information about the target variable. Then, the data is split according to this question, and we start again. This yields a hierarchical partitioning of the data, where each section of the partitioning becomes more and more “pure”, that is their content becomes more and more the same.

After you build the tree, you can make a prediction by checking which part of the partition a new point lies in and assigning the mean of the datapoints in this part.

Prediction



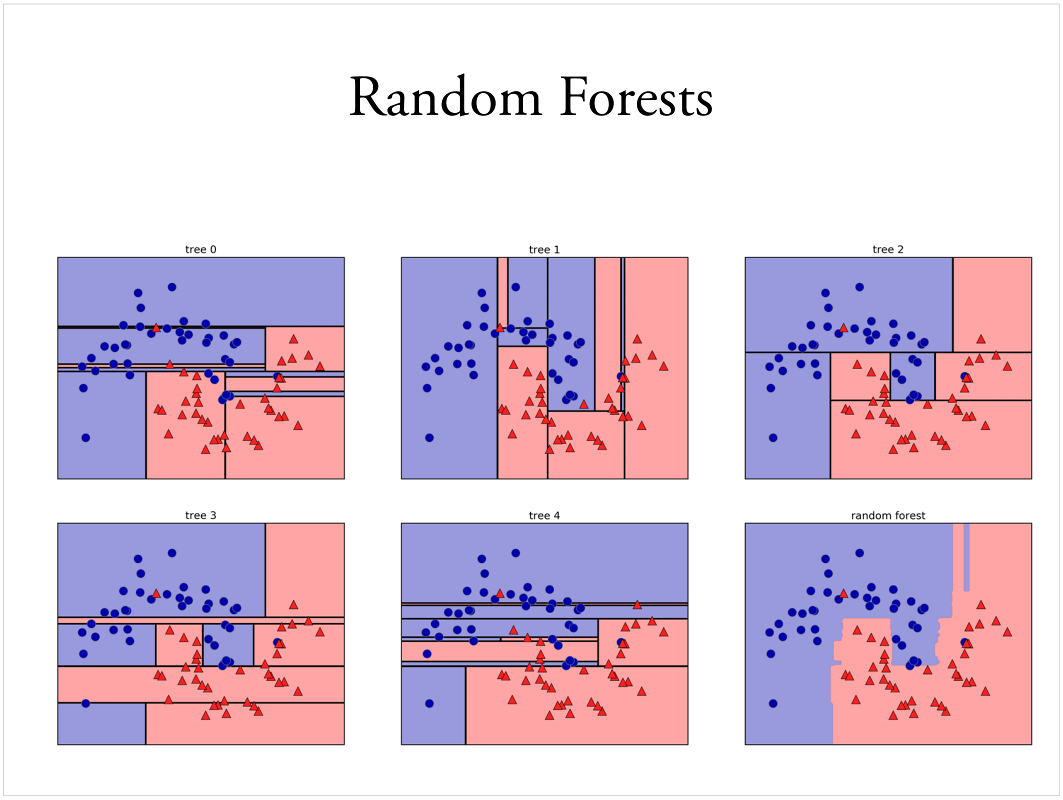


y=0

Traverse tree based on feature tests

Predict most common class in leaf

IPython Notebook:
Part 7 – Decision Trees



Random Forests

So decision trees are a great idea, but unfortunately they don’t work that well in practice. However, there is a modification of the algorithm that works very well, called random forest.

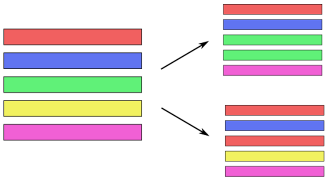
The idea behind random forest is that we build many decision trees, but we inject some randomness into each tree, so that they are all different.

Then, to make a prediction, we look at the prediction of all the decision trees and take the average.

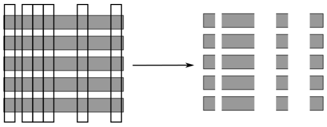
Randomize in two ways

- For each **tree**:

  Pick bootstrap sample of data



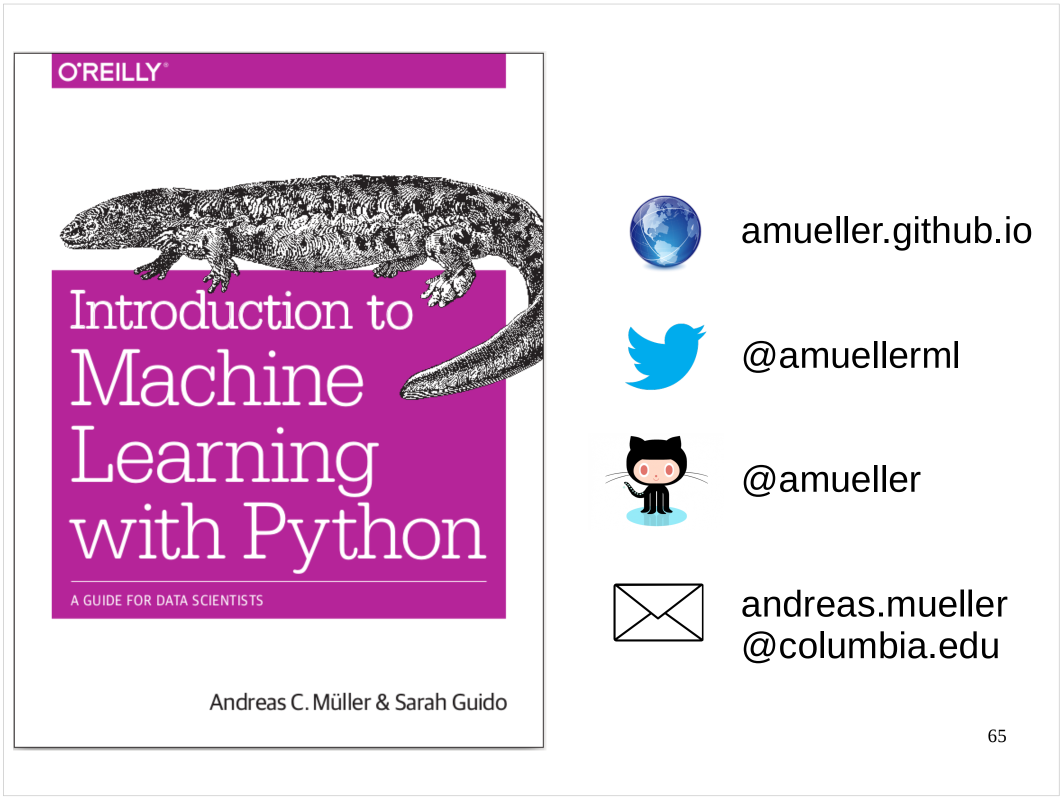
- For each **split**:
  Pick random sample of features



- More tree are always better

Tuning Random Forests

- Main parameter: max_features
  - around sqrt(n_features) for classification
  - Around n_features for regression
- n_estimators > 100
- Prepruning might help, definitely helps with model size!
- max_depth, max_leaf_nodes, min_samples_split again


O'REILLY
Introduction to Machine Learning with Python
A GUIDE FOR DATA SCIENTISTS
Andreas C. Müller & Sarah Guido

amueller.github.io

@amuellerml

@amueller

andreas.mueller
@columbia.edu

Buy my book. It’s about machine learning. It’s written for programmers, and should be a pretty easy read for anyone that knows a bit of numpy. I avoided adding to much math. If you want math, there are many awesome books to check out. This one is more about coding and how to get started with ml in python…. Which, you know, is why that’s the title…