Machine Learning, Computer Vision & Open Source

Andreas Müller
Columbia University, scikit-learn

Alfred P. Sloan
FOUNDATION

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

What is machine learning?

So let’s start with what machine learning is. 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 an existing 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.

Types of machine learning:

- supervised
- unsupervised
- reinforcement

There are three main branches of machine learning, called supervised learning, unsupervised learning and reinforcement learning.

I will only talk about supervised learning today, which is arguably the most commonly used and successful one right now.

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.

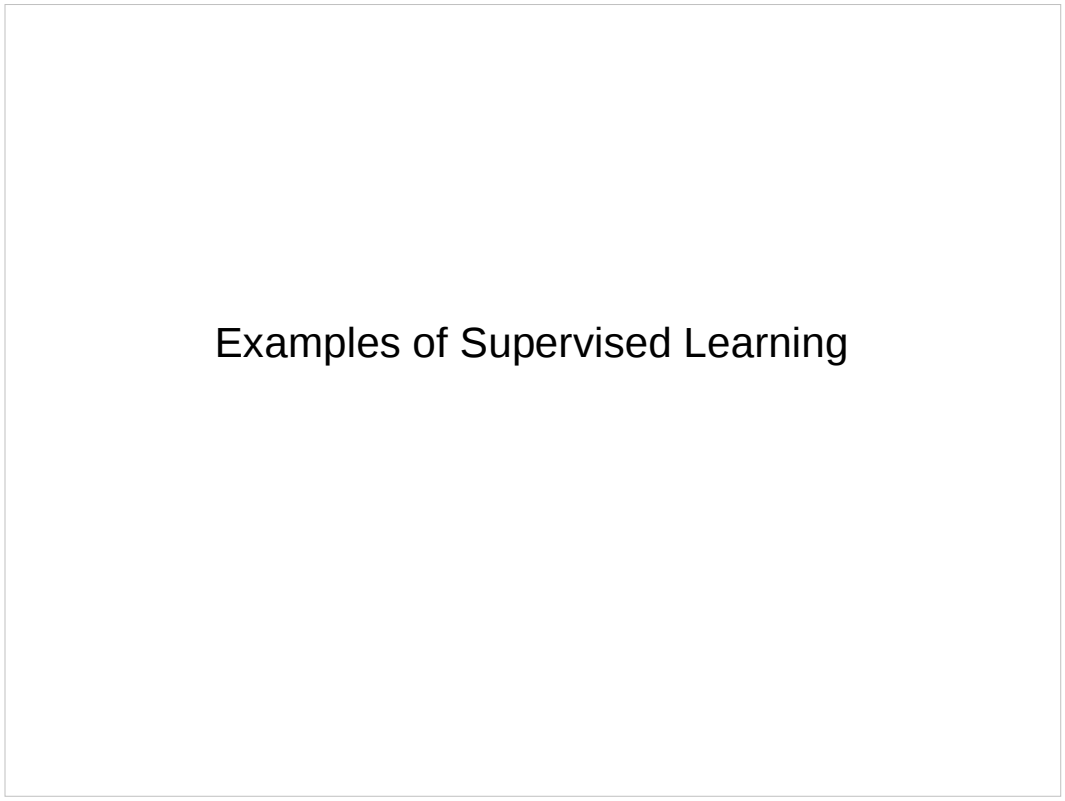
We assume these samples are drawn iid from some unknown joint distribution p(x, y).

You can think of this as there being some (not necessarily deterministic) process that goes from x_i to y_i, but that we don't know.

The goal is to learn a function f so that for new inputs x , and unobserved y, f(x) is approximately y.

This Is very closely related to function approximation.

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



There are two main categories of use-cases of supervised learning: learning from the past to predict in the future, and automating a manual task.

For example, you could have data set of diagnoses made by a doctor, and you could use machine learning to learn to predict the diagnosis from test results. This would replace or augment the human labor from the doctor.

Similarly, you could do this for content moderation or spam detection.

When trying to learn from the past, sometimes there is no human annotation necessary – it's automatically generated during the process. For example if you want to predict whether a user will click on a particular ad on your platform, you just need to log user behavior. Then, given the past behavior of your users, where you observed whether they clicked or not, you can build a model to predict future behavior.

Classification and Regression

Classification:

- y discrete

Regression:

- y continuous

As I said, I want to focus on supervised learning, though.
There are two basic kind of supervised learning, 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.

For example if you want to decide what disease a patient has, this is classification. If you want to predict a patients heart rate or blood sugar level, that's regression.

This distinction into classification and regression might seem a bit trivial, but measuring "how good" a prediction is is quite different for discrete and continuous y.

Generalization

Not only $f(x_i) \approx y_i$

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

I want to talk in more depth about supervised learning, and I want to get a little more formal.

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 only on this training data, but on new data drawn from this distribution. This is called generalization, and this is a core property of machine learning. 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.

(regularized) Empirical Risk Minimization

Goal: $\min_{f \in F} \mathbb{E}_{x,y} \ell(f(x), y)$

Let’s get a little bit more formal about this. Our goal is to find a function f that has minimal expected loss on the data distribution, where loss could be something like mean squared error or accuracy of predictions.

However, there is no way to measure that, because the data distribution is unknown. We only have the datasets, which is an iid sample.

Empirical Risk Minimization

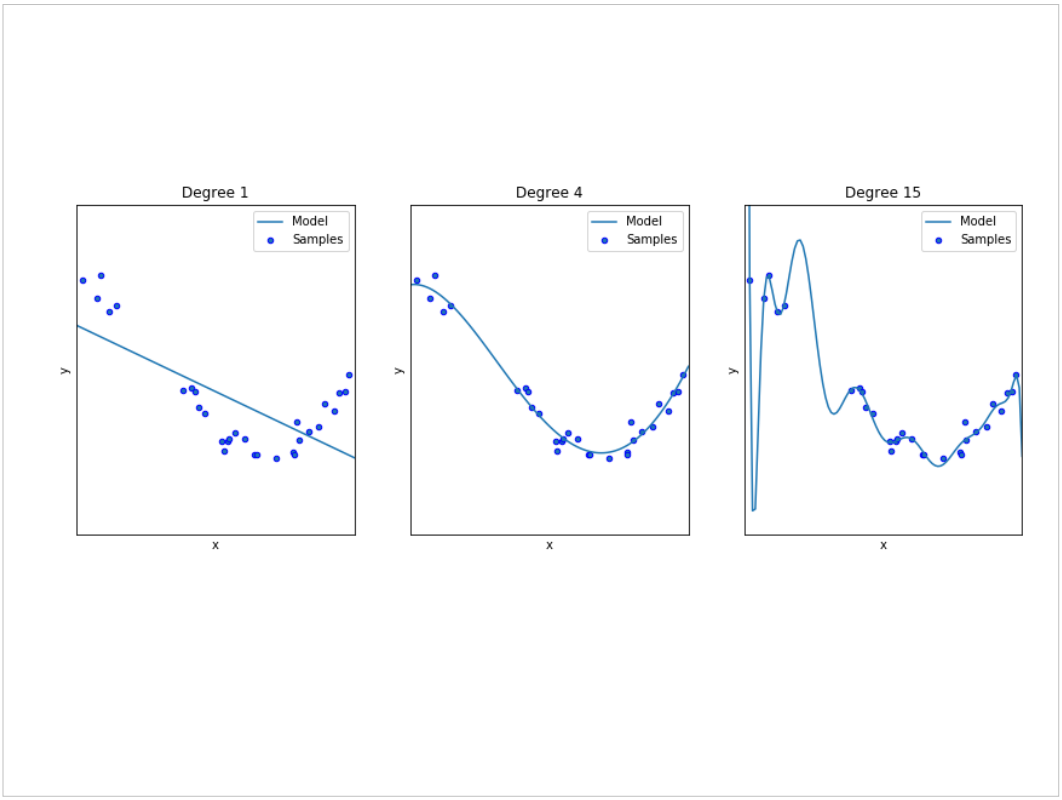
Goal: $\min_{f \in F} \mathbb{E}_{x,y} \ell(f(x), y)$

solvable: $\min_{f \in F} \sum_i \hat{\ell}(f(x_i), y_i)$

So what we are doing instead, is we minimize what is known as the empirical risk, which is just the loss summed over the training data.

You might notice that I also put a hat on the ell. In classification, we're often interested in the number of accurately classified points. However, that loss function is non-convex and non-differentiable, and so when we try to optimize the problem at the bottom, we use a substitute loss function that is better behaved, like a convex upper bound.

Depending on ell hat and the family F, this problem might be reasonably easy to solve. But there is one more problem. If F is very large, there might be multiple perfect solutions, in particular there can be solutions that are too complex for the problem.



Here you can see this problem when using polynomial regression. We one input feature on the x axis and a target on the y axis, with the blue dots the data and the blue line different fits to the data.

On the left is a linear fit, in the center a degree 4 fit, and on the right a degree 15 fit. Clearly the fit on the right has the least error on the training set, because it's closest to all the points. But it's also a very complex function, that we might not expect to generalize well to new data. This is called overfitting.

Because we are interested in generalization, we want to penalize functions that fit the data well, but that are too complex.

Regularized Empirical Risk Minimization

Goal: $\min_{f \in F} \mathbb{E}_{x,y} \ell(f(x), y)$

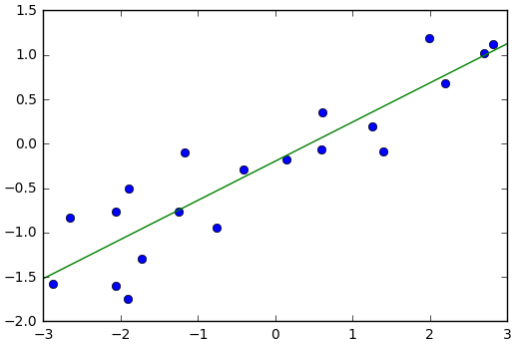
solvable: $\min_{f \in F} \sum_i \hat{\ell}(f(x_i), y_i)$

better: $\min_{f \in F} \sum_i \hat{\ell}(f(x_i), y_i) + \lambda R(f)$

That leads us to “regularized empirical risk minimization”, which adds a so called regularizer R to the optimization problem. This is basically the most common formalism for machine learning. We want to minimize a loss over the training set, but we also don’t want to choose a function that is too complex. The choice of R depends on what the family F is, but usually there is a free parameter lambda, which allows us to trade off between fitting the data and a simple model.

Many different machine learning models correspond to different choices of ell hat, the family F and the regularizer R.

Linear Regression as ERM



$$\min_{w \in \mathbb{R}^n} \sum_i ||w^T x_i - y_i||^2$$

Let's make this more concrete by looking at an example: linear regression.
That's even simpler than polynomial fitting, but one of the really simple and successful methods in statistics and machine learning.

Linear regression, or ordinary least squares means finding a vector w such that w * x – y is minimized.
This is basically only the first part of the principle I just showed you, the data fitting part.
The family of models is parametrized by w, and the ell hat is the squared euclidean distance.

if you look at this in 2d, it looks like a great idea.
This is mostly because linear models in 2d are inherently simple.
However, if you go to higher dimensional spaces, linear models can become quite complex.

Regularized Linear Regression: Ridge

$$\min_{w \in \mathbb{R}^n} \sum_i ||w^T x_i - y_i||^2 + \lambda ||w||^2$$

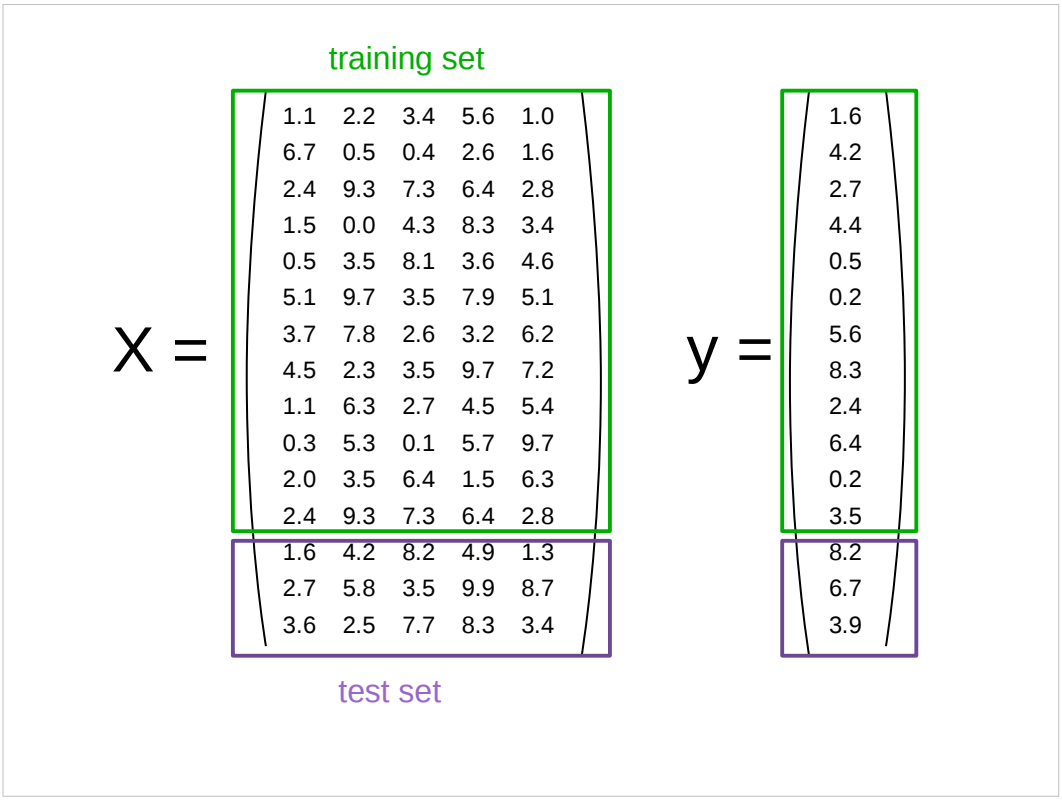
If you have more features than observations, you can find a linear model that fits any possible outcome y perfectly! It's an underspecified linear system.

But I don't want to give up, I still want to learn. And I can. I just need to add in the second part, the regularization, that tells me to keep the model simple.

For linear regression, the easiest way to do that is by adding the l2 norm of w to the objective, and we get what's called ridge regression. This basically says "I want to fit the data, but don't make w too long" - or in other words, don't make any particular entry of w too large.

This will make the optimization problem well-specified, and lead to a more "simple" model that generalizes better.

You might notice that the formula also has a lambda in it, which balances between data fitting and regularization, and you probably want to ask: what's lambda? Well, it's a real number, and you have to pick it, and I'll tell you later how to do it.



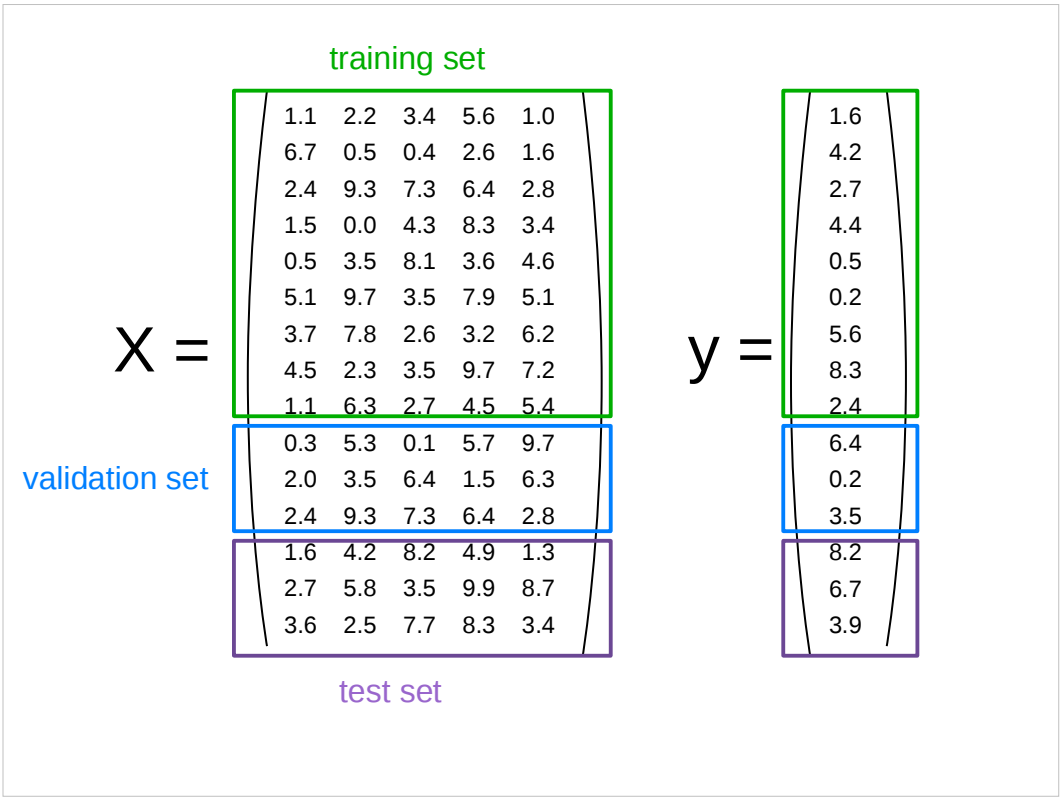
In the polynomial regression task, we could see whether we think a model will generalize well to new data from eyeballing it. That's clearly not a good way to test a model, and usually impossible in higher dimensions.

The standard method to evaluate the generalization capabilities is with a hold-out set, also known as test-set or validation set.

The way this works it that we split our initial dataset into one part that is used for actually building the model, which we call the training set, and another part, the test set, which will be our "pretend future data". The essential point here is that we have the true outputs, or ground truth, for this data, so we can measure how well our model did on this "new" data.

So first we split the data, then we build a model on the training part, and then we evaluate it on the test part.

And that provides an unbiased estimate of generalization performance.



However, we also had the problem of picking the regularization parameter lambda.

We could use the test set to adjust lambda, but then the test set will not provide an unbiased estimate of the generalization performance any more.

So what we have to do is split the data again, into a training set, a validation set and a test set.

The training set is used to build the model, the validation set is used to select between models and adjust tuning parameters like the regularization parameter, and the test set is used to estimate the generalization performance of the model.

Question?

So now that we have all the basics down, let's look at some more complex models.

Neural Networks

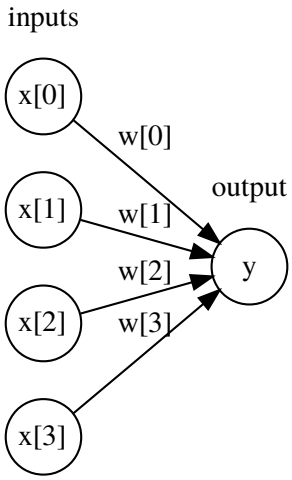
History

- Nearly everything we talk about today existed ~1990
- What changed?
  - More data
  - Faster computers (GPUs)
  - Some improvements in algorithms.

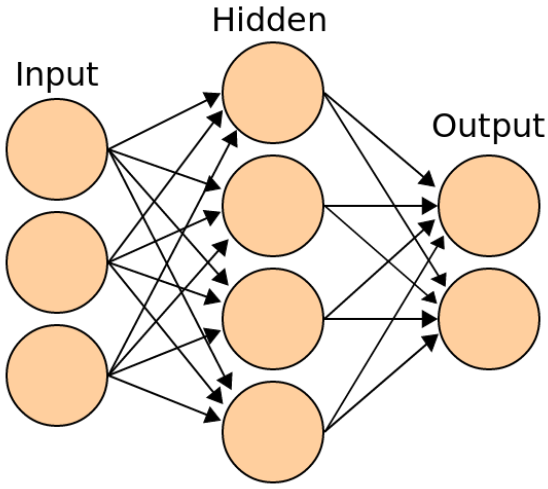
Supervised Neural Networks

- Non-linear models for classification and regression
- Work well for very large datasets
- Non-convex optimization
- Notoriously slow to train – need for GPUs
- Use dot products etc → require preprocessing, similar to SVM or linear models, unlike trees
- MANY variants (Convolutional nets, Gated Recurrent neural networks, Long-Short-Term Memory, recursive neural networks, variational autoencoders, general adverserial networks, neural turing machines...)

Linear regression drawn as neural net



Basic Architecture (for making predictions)



First let's describe how to make a prediction given a model.

Input denotes single sample, here three input features.

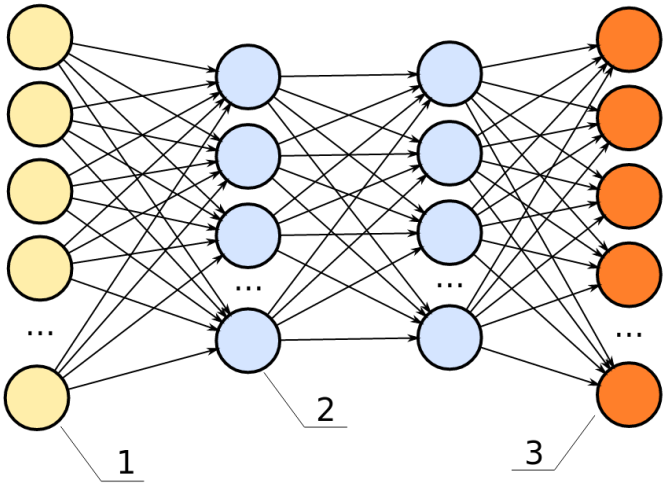
Hidden layer here 4 units is matrix multiply with W, b added (size of b is 4 here), followed by the univariate non-linear function f – sigmoid, tanh, rectifying linear function.

Output is a matrix multiplication with different weight matrix W', b' added (size 2 here), followed by another non-linear function g. The function g for the output layer is often different: identity for regression, soft-max for classification.

We want to learn W (3x4) W' (4x2), b (4,), b' (2,).

Can think of it as logistic regression with learning a non-linear basis transformation.

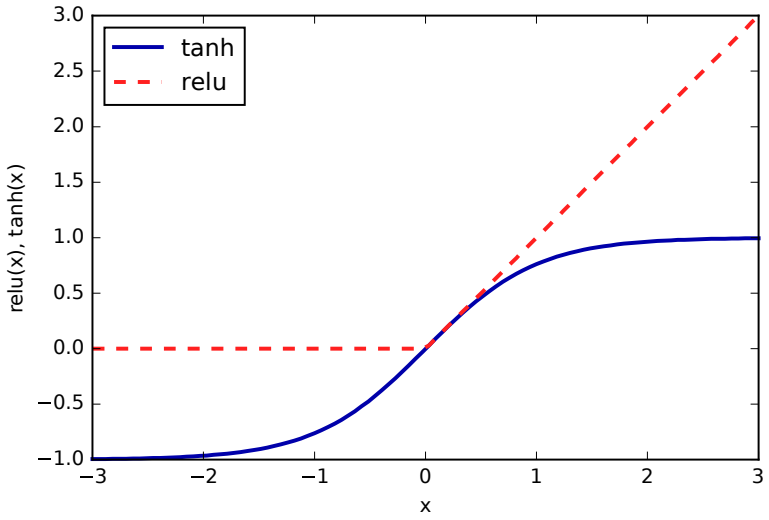
Can have arbitrary many layers



Hidden layers usually all have the same non-linear function, weights are different for each layer.
Many layers → “deep learning”.
This is called a multilayer perceptron, feed-forward neural network, vanilla feed-forward neural network.

For regression usually single output neuron with linear activation.
For classification one-hot-encoding of classes, n_classes many output variables with softmax.

Nonlinear activation function



Choices for activation function f of hidden layers.

Traditional tanh (or logistic sigmoid, not shown, but similar).

Tanh squashes to open interval (-1, 1).

Relu – recent trend, linear function x=y for positive, constant for negative values. Bias allows shifting the cut-off (~ linear splines)

Training objective

$$h(\mathbf{x}) = f(W_1\mathbf{x} + \mathbf{b}_1)$$

$$o(\mathbf{x}) = g(W_2 h(\mathbf{x}) + \mathbf{b}_2) = g(W_2 f(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

$$\min_{W_1, W_2, \mathbf{b}_1, \mathbf{b}_2} \sum_{i=1}^{N} \ell(y_i, o(\mathbf{x}_i))$$

Could add regularization

$$= \min_{W_1, W_2, \mathbf{b}_1, \mathbf{b}_2} \sum_{i=1}^{N} \ell(y_i, g(W_2 f(W_1\mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2))$$

ℓ squared loss for regression
cross-entropy loss (multi-class log-loss) for classification

Backpropagation

- For gradient based-method need $\frac{\partial o(\mathbf{x})}{\partial W_i}$ $\frac{\partial o(\mathbf{x})}{\partial \mathbf{b}_i}$

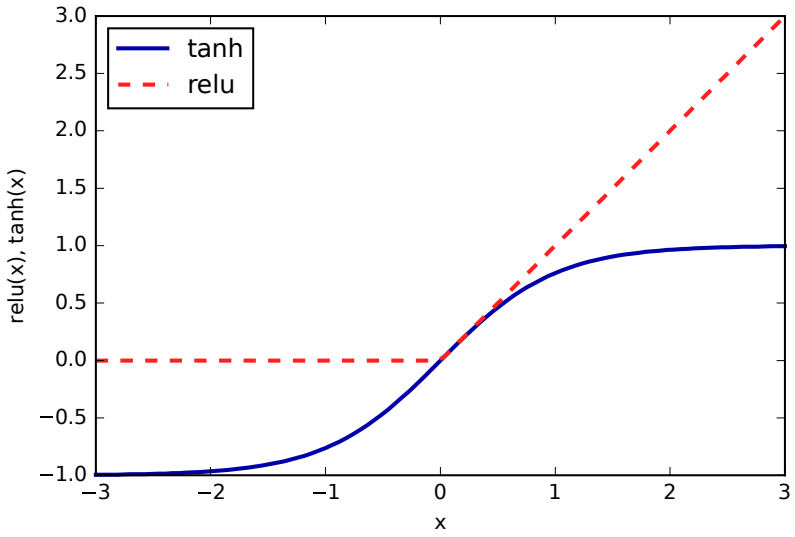
$$\mathrm{net}(\mathbf{x}) := W_1 \mathbf{x} + b_1$$

$$\frac{\partial o(\mathbf{x})}{\partial W_1} = \underbrace{\frac{\partial o(\mathbf{x})}{\partial h(\mathbf{x})}}_{\text{backpropagation of gradient of layer above.}} \underbrace{\frac{\partial h(\mathbf{x})}{\partial \mathrm{net}(\mathbf{x})}}_{\text{Gradient of Non-linearity f}} \underbrace{\frac{\partial \mathrm{net}(\mathbf{x})}{\partial W_1}}_{\text{Input to 1st layer x}}$$

Backpropagation = Chain Rule + Dynamic Programming

- Easy to write down for last layer
- Example for squared loss (g is identity for regression)
- Can use the chain rule to compute other gradients
- Bottom layers require partial derivatives of upper layers → reuse results
  - Backpropagation: dynamic programming + chain rule
  - “backward pass” compute partial derivatives starting at the last layer.
  - You should try to go through that yourself once

But wait!



- relu is not differentiable.
- But it’s differentiable almost anywhere.
- “subgradient descent” - little issues in practice

Optimizing W, b

$$W_i \leftarrow W_i - \eta \sum_{j=1}^{n} \frac{\ell(\mathbf{x}_j, y_j)}{W_i}$$ batch

$$W_i \leftarrow W_i - \eta \sum_{j=k}^{k+m} \frac{\ell(\mathbf{x}_j, y_j)}{W_i}$$ minibatch

$$W_i \leftarrow W_i - \eta \frac{\ell(\mathbf{x}_j, y_j)}{W_i}$$ Online / stochastic

- Standard solvers: l-bfgs, newton, cg
- Problem: Hessian too expensive, can to l-bfgs
- Computing gradients over whole dataset expensive
- Stochastic Gradient Descent to rescue
- Actually use mini-batches

Learning Heuristics

- Constant η not good
- Can decrease η
- Better: adaptive η for each entry if Wi
- State-of-the-art: adam (with magic numbers)

https://arxiv.org/pdf/1412.6980.pdf

http://sebastianruder.com/optimizing-gradient-descent/

Convolutional Neural Networks

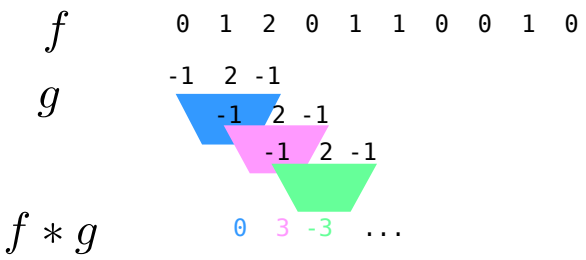
Idea

- Translation invariance
- Weight sharing

Definition of Convolution

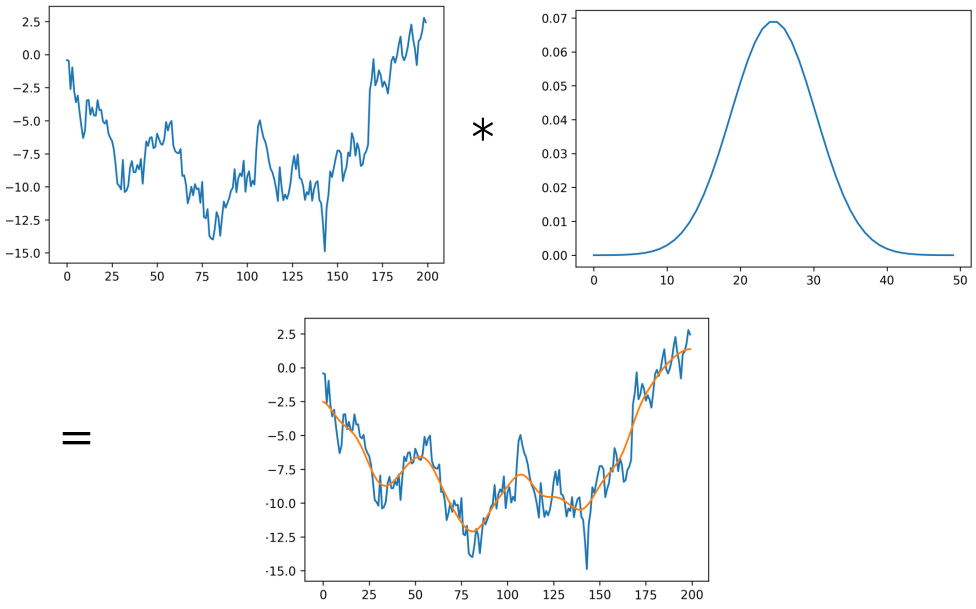
$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$$

$$= \sum_{m=-\infty}^{\infty} f[n-m]\, g[m].$$

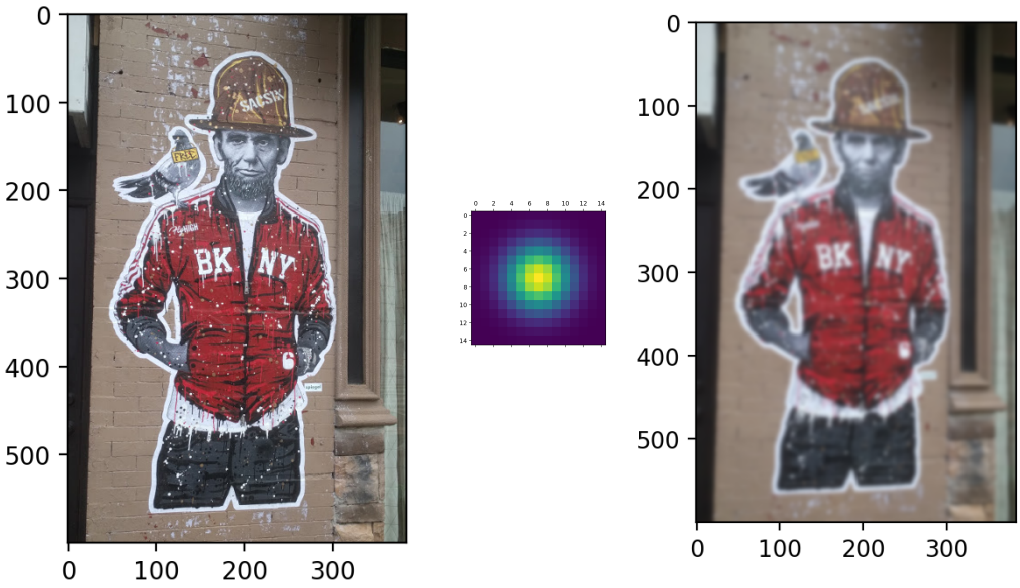


- The definition is symmetric in f, but usually one is the input signal, say f, and g is a fixed “filter” that is applied to it.
- You can imagine the convolution as g sliding over f. If the support of g is smaller than the support of f (it’s a shorter non-zero sequence) then you can think of it as each entry in f * g depending on all entries of g multiplied with a local window in f.
- Not that the output is shorter than the input by half the size of g. this is called a **valid** convolution.
- We could also extend f with zeros, and get a result that is larger than f by half the size of g, that’s called a **full** convolution. We can also just pad a little bit and get something that is of the same size as f.
- Also not that the filter g is flipped as it’s indexed with -m

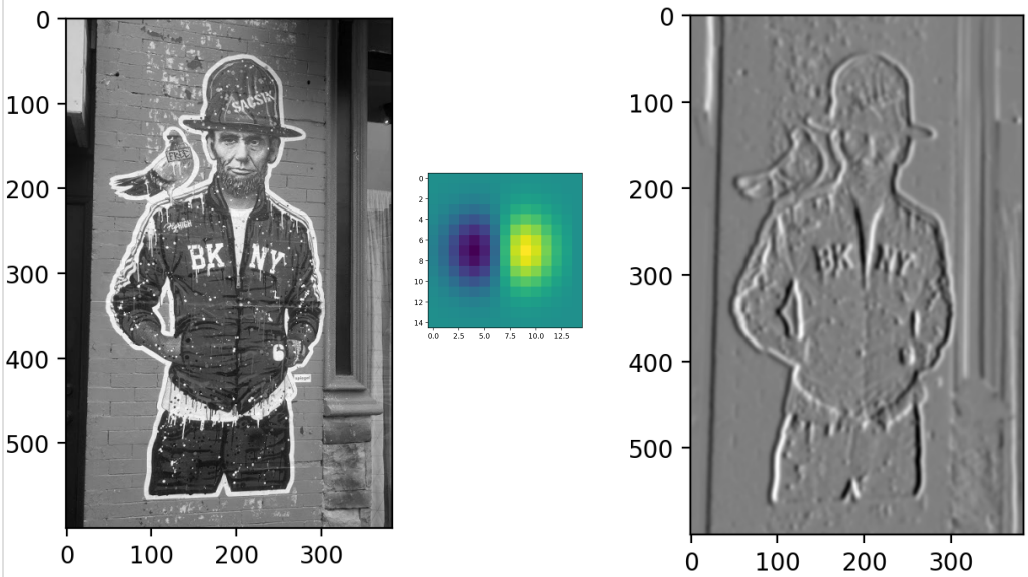
1d example: Gaussian smoothing



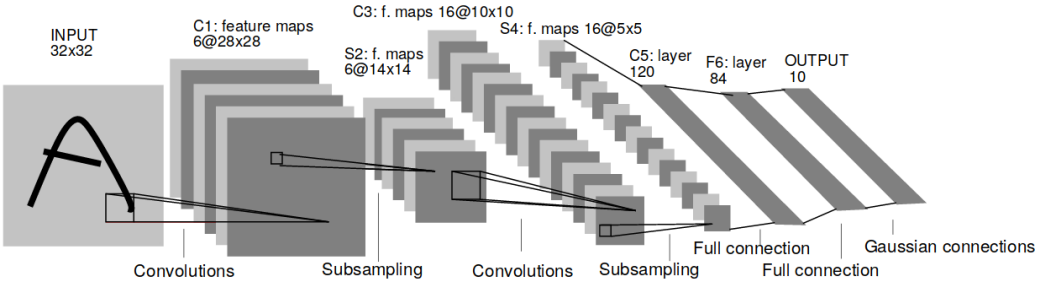
2d smoothing



2d Gradients



Convolutional Neural Networks



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.

Gradient-based learning applied to document recognition

Here is the architecture of an early convolutional net form 1998. The basic architecture in current networks is still the same.

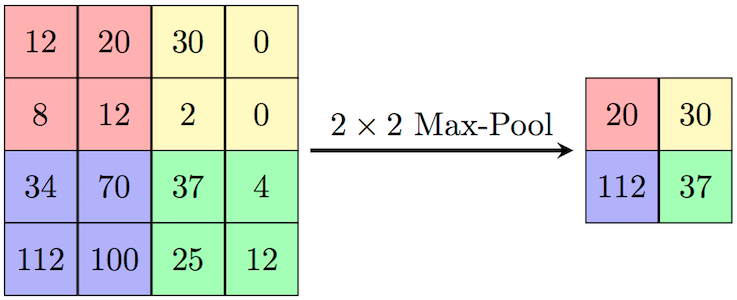
You can multiple layers of convolutions and resampling operations. You start convolving the image, which extracts local features. Each convolutions creates new “feature maps” that serve as input to later convolutions.

To allow more global operations, after the convolutions the image resolution is changed. Back then it was subsampling, today it is max-pooling.

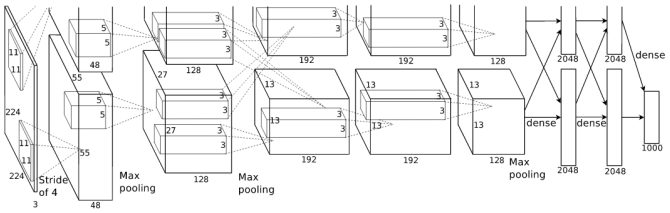
So you end up with more and more feature maps with lower and lower resolution.

At the end, you have some fully connected layers to do the classificattion.

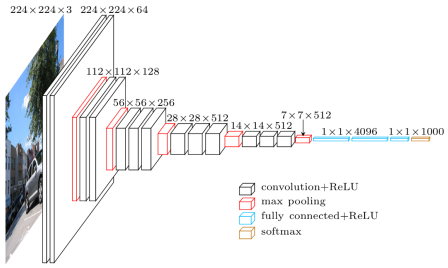
Max pooling



Need to remember position of maximum for back-propagation.
Again not differentiable → subgradient descent



Alex Net



VGG 16

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

conv <receptive field size>-<number of channels>

Here are two more recent architectures, AlexNet from 2012 and VGG net from 2015.

These nets are typically very deep, but often have very small convolutions. In VGG there are 3x3 convolutions and even 1x1 convolutions which serve to summarize multiple feature maps into one.

There is often multiple convolutions without pooling in between but pooling is definitely essential.

Structured Prediction

Structured Prediction

$$y = (y_1, y_2, \ldots y_{n_k})$$

Applications: Multi-Label Classification

	Politics	Sports	Finance	Domestic	Religion
News Story1	1	0	0	1	1
News Story2	0	1	0	1	0
News Story3	0	0	1	0	0

Applications: Multi-Label Classification

	Politics	Sports	Finance	Domestic	Religion
News Story1	1	0	0	1	1
News Story2	0	1	0	1	0
News Story3	0	0	1	0	0

	Owns Car	Smokes	Married	Self-Employed	Has Kids
Customer1	1	0	1	0	1
Customer2	1	1	0	1	0
Customer3	0	1	1	0	0

Applications: Sequence Tagging

Applications: Sequence Tagging

Stroke cat.

Stroke cat.

Stroke cat.

Open trash can.

Put cat in trash can.

Applications: Image Segmentation

The Essence of Structured Prediction

$$f(x, w) := \arg\max_{y \in \mathcal{Y}} g(x, y, w)$$

The Essence of Structured Prediction

$$f(x, w) := \arg\max_{y \in \mathcal{Y}} g(x, y, w)$$

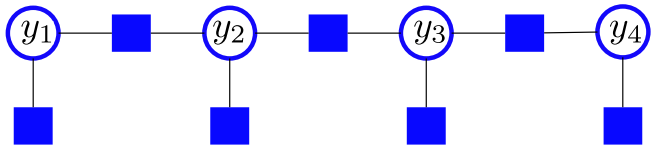
If you like:

$$\arg\max_{y \in \mathcal{Y}} p(y|x, w)$$

Pairwise Structured Models

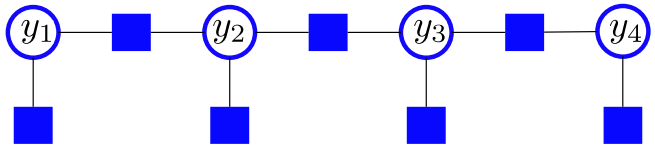
$$\arg\max_{y_1, y_2, \ldots, y_n} w^T \psi(x, y)$$

$$= \arg\max_{y_1, y_2, \ldots, y_n} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j) \in E} w_{i,j}^T \psi(x, y_i, y_j)$$



The Devil is in the Inference

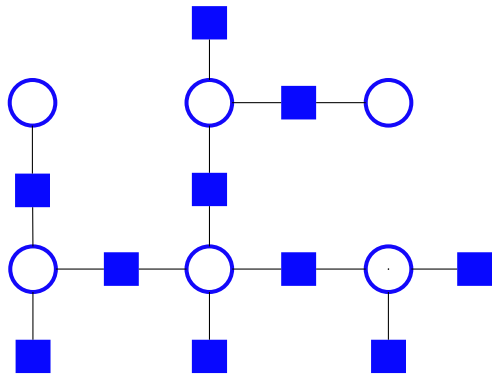
$$\arg\max_{y_1,y_2,\ldots,y_n} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j)\in E} w_{i,j}^T \psi(x, y_i, y_j)$$



Easy: Dynamic Programming

The Devil is in the Inference

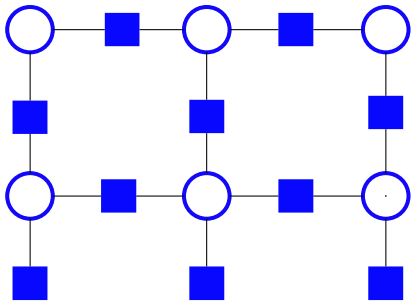
$$\arg\max_{y_1,y_2,\ldots,y_n} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j)\in E} w_{i,j}^T \psi(x, y_i, y_j)$$



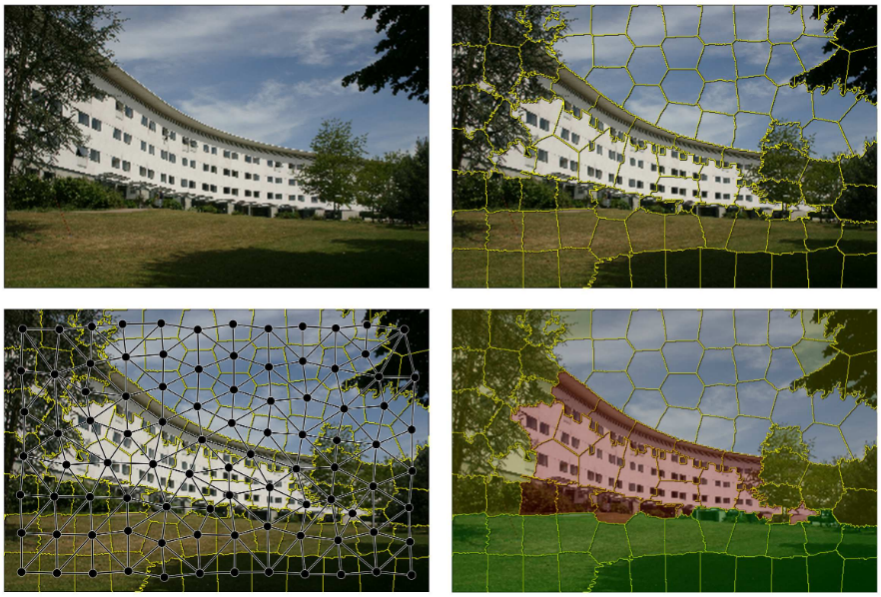
Easy: Dynamic Programming

The Devil is in the Inference

$$\underset{y_1,y_2,\ldots,y_n}{\arg\max} \sum_{I} w_i^T \psi(x, y_i) + \sum_{(i,j)\in E} w_{i,j}^T \psi(x, y_i, y_j)$$



HARD!
AD3, QPBO, LP, Loopy BP, ….



The Open Source Idea

Open exchange

We can learn more from each other when information is open. A free exchange of ideas is critical to creating an environment where people are allowed to learn and use existing information toward creating new ideas.

Participation

When we are free to collaborate, we create. We can solve problems that no one person may be able to solve on their own.

Rapid prototyping

Rapid prototypes can lead to rapid failures, but that leads to better solutions found faster. When you're free to experiment, you can look at problems in new ways and look for answers in new places. You can learn by doing.

Meritocracy

In a meritocracy, the best ideas win. In a meritocracy, everyone has access to the same information. Successful work determines which projects rise and gather effort from the community.

New BSD License

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

  a. Redistributions of source code must retain the above copyright notice,
     this list of conditions and the following disclaimer.
  b. Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in the
     documentation and/or other materials provided with the distribution.
  c. Neither the name of the Scikit-learn Developers nor the names of
     its contributors may be used to endorse or promote products
     derived from this software without specific prior written
     permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.

Permissive vs Share-alike

BSD / MIT / Apache

Hadoop

Python (+libraries)

Apache web server

Do whatever you want
= Industry friendly

GPL / LGPL

Linux

Most linux tools (GNU)

Derived software must be licensed the same way.
Derived web-services must be open source.

No “free riders”

So if you want to do machine learning in python, your go-to library is scikit-learn. And I'm only saying that because I'm one of the maintainer ;)

The rest of my talk I want to spend on how to apply scikit-learn to do machine learning, and discuss some methods and best practices.

Scikit-learn is built upon numpy and scipy, and the main data structures it works with are numpy arrays, and I hope you're all familiar with them.

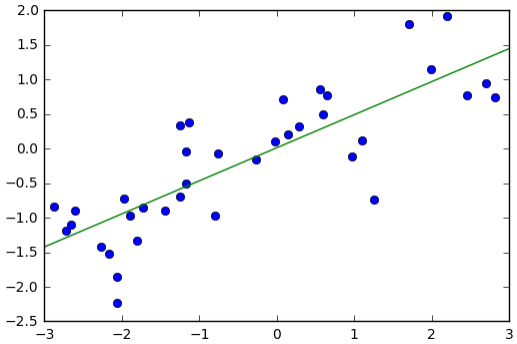
Classification
Regression
Clustering
Semi-Supervised Learning
Feature Selection
Feature Extraction
Manifold Learning
Dimensionality Reduction
Kernel Approximation
Hyperparameter Optimization
Evaluation Metrics
Out-of-core learning
……

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train)

print(lr.predict(X_test))
print(lr.score(X_test, y_test))
```

```
[-1.07669396 -0.94489099  0.29582604  1.32379186 -1.38209551 -1.42220547
  0.86691958 -0.63314097  0.79944587  1.28044585  0.64491641  0.23934466
 -0.73008772]
0.465661173258
```



We import linear regression and instantiate the class
Then we can fit the model on the training data and the training targets with the fit method.
This estimates slope and offset of the linear regression, which are stored in the lr object.
We can access them as lr.coef_ and lr.intercept_.

We can then make prediction using this linear regression model using the ``predict`` function.
Here are the predictions for the training set.

We can also make predictions for the unseen data, the test set.

To evaluate how well we do on either the training set or the test set, we can use the score method, which for regression computes the R^2.
So here you can see the three central methods of scikit-learn: fit, predict and score.


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

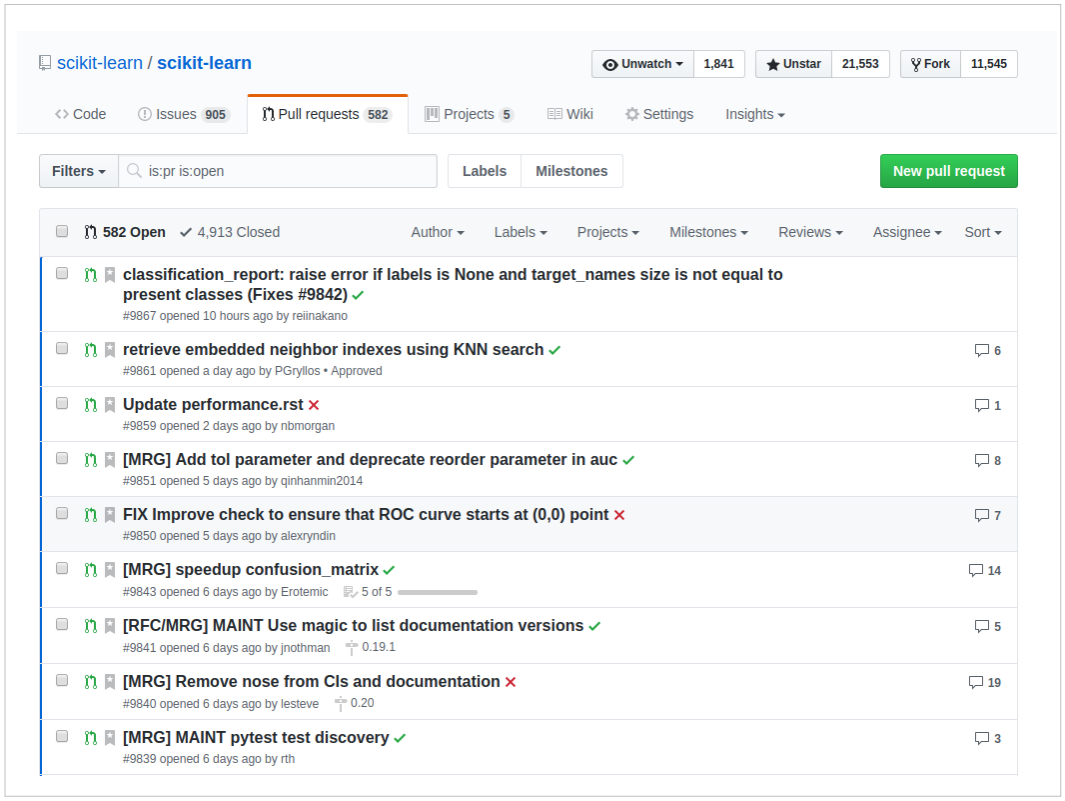

Vlad Niculae
vene


Virgile Fritsch
VirgileFritsch


Vincent Michel
vmichel

Wei Li
wellinear

Yaroslav Halchenko
yarikoptic


scikit-learn / scikit-learn
Unwatch 1,841
Unstar 21,553
Fork 11,545
Code
Issues 905
Pull requests 582
Projects 5
Wiki
Settings
Insights
Filters
is:pr is:open
Labels
Milestones
New pull request
582 Open
4,913 Closed
Author
Labels
Projects
Milestones
Reviews
Assignee
Sort
classification_report: raise error if labels is None and target_names size is not equal to present classes (Fixes #9842)
#9867 opened 10 hours ago by reiinakano
retrieve embedded neighbor indexes using KNN search
#9861 opened a day ago by PGryllos • Approved
6
Update performance.rst
#9859 opened 2 days ago by nbmorgan
1
[MRG] Add tol parameter and deprecate reorder parameter in auc
#9851 opened 5 days ago by qinhanmin2014
8
FIX Improve check to ensure that ROC curve starts at (0,0) point
#9850 opened 5 days ago by alexryndin
7
[MRG] speedup confusion_matrix
#9843 opened 6 days ago by Erotemic
5 of 5
14
[RFC/MRG] MAINT Use magic to list documentation versions
#9841 opened 6 days ago by jnothman
0.19.1
5
[MRG] Remove nose from CIs and documentation
#9840 opened 6 days ago by lesteve
0.20
19
[MRG] MAINT pytest test discovery
#9839 opened 6 days ago by rth
3

```
-    y_pred = y_pred[ind]
-    y_true = y_true[ind]
-    # also eliminate weights of eliminated items
-    sample_weight = sample_weight[ind]
+    # If labels are not consecitive integers starting from zero, then
+    # yt, yp must be converted into index form
+    need_index_conversion = not (
```

jnothman 5 days ago **Owner**

Surely `np.all(labels == np.arange(len(labels)))` is just as fast for a reasonable number of classes and much more readable?

Erotemic 3 days ago • edited Contributor

It is both faster and more readable, when I originally wrote this I didn't consider that the ordering of the labels was significant, so the `np.diff` was a quick fix after realizing this. Your solution is simpler and better.

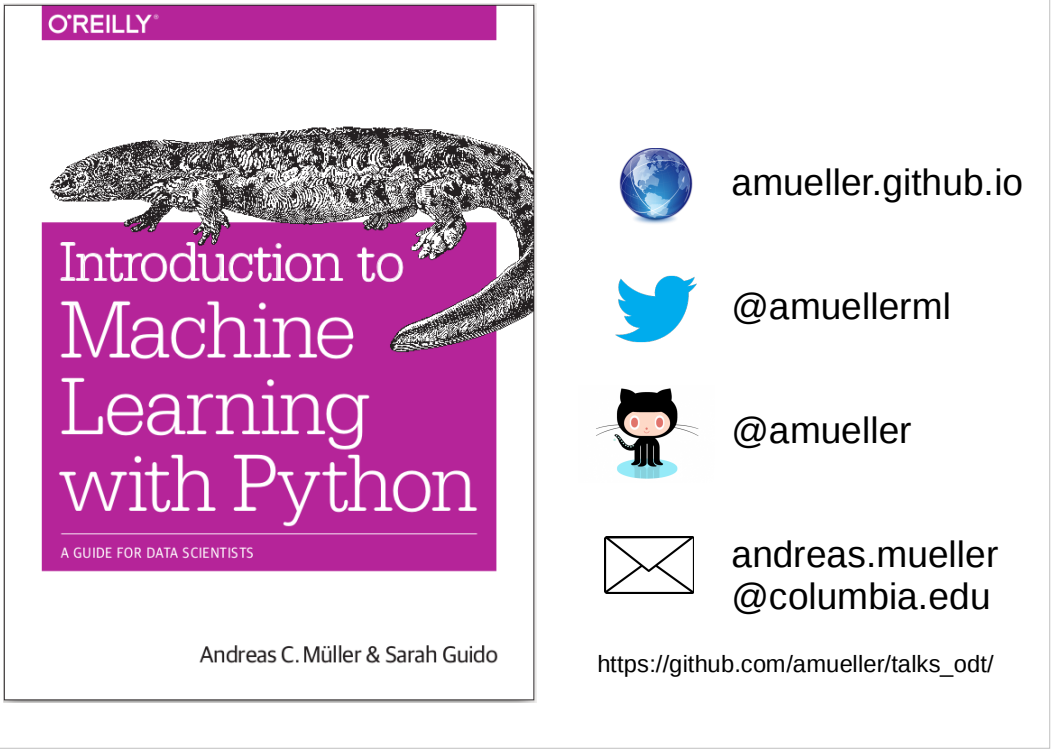
```
In [1]: labels = np.arange(100)

In [2]: %timeit labels.min() == 0 and np.all(np.diff(labels) == 1)
8.3 µs ± 162 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

In [3]: %timeit np.all(np.arange(len(labels)) == labels)
3.6 µs ± 47.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Reply…

Start a new conversation



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…

I hope my rant was a bit informative, or at least entertaining. And A big shout out to the organizers for putting together such a great conference, and thanks again for having me!