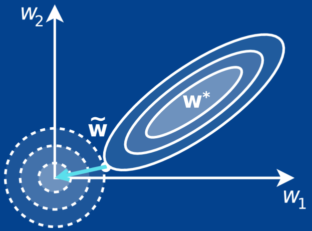
AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Lesson 09: Regularization, Optimization and Searching

CARSTEN EIE FRIGAARD

AUTUMN 2022



"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E." — Mitchell (1997).

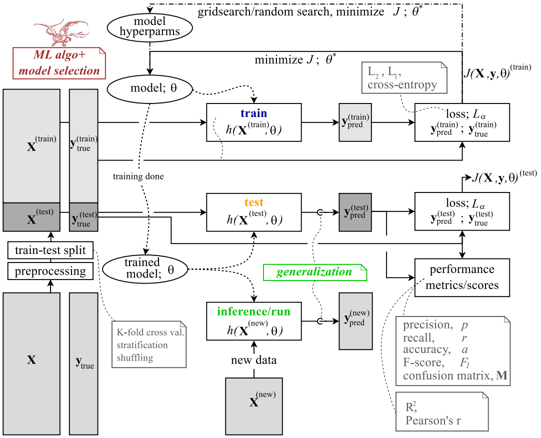
Agenda

- Resumé: ML Algorithm and Model Selection:
  k-fold Cross-Validation revisited,
- Regularization:
  Regulizers,
  Exercise: `L09/regulizers.ipynb`
- Optimizers:
  (no exercise).
- Searching:
  Gridsearch,
  Randomsearch,
  Exercise: `09/gridsearch.ipynb`

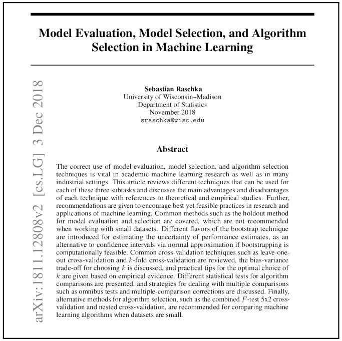
ML Algorithm Selection and Model Selection

Manually Choosing an Algorithm and Tuning a Model..

- algorithm selection.
- model selection,
- model evaluation,
- re-iteration and re-selection!



ML algo+ model selection
model hyperparms
gridsearch/random search, minimize J ; θ^*
minimize J ; θ^*
model; θ
L_2 , L_1 , cross-entropy
$J(\mathbf{X},\mathbf{y},\theta)^{(train)}$
train
$h(\mathbf{X}^{(train)},\theta)$
$\mathbf{y}^{(train)}_{pred}$
loss; L_α
$\mathbf{y}^{(train)}_{pred}$; $\mathbf{y}^{(train)}_{true}$
$\mathbf{X}^{(train)}$
$\mathbf{y}^{(train)}_{true}$
training done
$J(\mathbf{X},\mathbf{y},\theta)^{(test)}$
test
$h(\mathbf{X}^{(test)},\theta)$
$\mathbf{y}^{(test)}_{pred}$
loss; L_α
$\mathbf{y}^{(test)}_{pred}$; $\mathbf{y}^{(test)}_{true}$
$\mathbf{X}^{(test)}$
$\mathbf{y}^{(test)}_{true}$
train-test split
preprocessing
trained model; θ
generalization
performance metrics/scores
inference/run
$h(\mathbf{X}^{(new)},\theta)$
$\mathbf{y}^{(new)}_{pred}$
new data
$\mathbf{X}^{(new)}$
$\mathbf{X}$
$\mathbf{y}_{true}$
K-fold cross val.
stratification
shuffling
precision, p
recall, r
accuracy, a
F-score, F_1
confusion matrix, $\mathbf{M}$
R^2,
Pearson's r



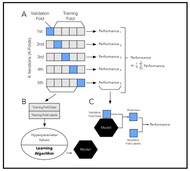
Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning

Sebastian Raschka
University of Wisconsin–Madison
Department of Statistics
November 2018
sraschka@wisc.edu

Abstract

The correct use of model evaluation, model selection, and algorithm selection techniques is vital in academic machine learning research as well as in many industrial settings. This article reviews different techniques that can be used for each of these three subtasks and discusses the main advantages and disadvantages of each technique with references to theoretical and empirical studies. Further, recommendations are given to encourage best yet feasible practices in research and applications of machine learning. Common methods such as the holdout method for model evaluation and selection are covered, which are not recommended when working with small datasets. Different flavors of the bootstrap technique are introduced for estimating the uncertainty of performance estimates, as an alternative to confidence intervals via normal approximation if bootstrapping is computationally feasible. Common cross-validation techniques such as leave-one-out cross-validation and k-fold cross-validation are reviewed, the bias-variance trade-off for choosing k is discussed, and practical tips for the optimal choice of k are given based on empirical evidence. Different statistical tests for algorithm comparisons are presented, and strategies for dealing with multiple comparisons such as omnibus tests and multiple-comparison corrections are discussed. Finally, alternative methods for algorithm selection, such as the combined F-test 5x2 cross-validation and nested cross-validation, are recommended for comparing machine learning algorithms when datasets are small.

"Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning", Sebastian Raschka, 2018.



ML Algorithm Selection and Model Selection

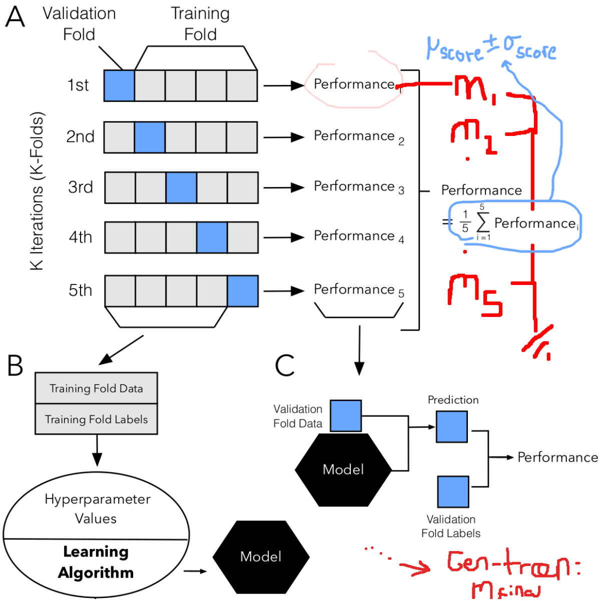
Manually Choosing an Algorithm and Tuning a Model..

- algorithm selection:
  - choose an algo that *'fits'* the problem,
    ...perhaps via searching??
- model selection:
  - looking for 'optimal' model capacity,
  - tuning model **hyperparameters**..
    - model weights **regularizers**..(for NN's)
    - gradient descent **optimizers**..(for NN's)
- model evaluation:
  - the performance metric `score` function,
  - how do you evaluate **generalization** performance?
  - holdout method (train-test split) and k-fold CV,
  - three-way split (train-validate-test split)..
- re-iteration and re-selection!

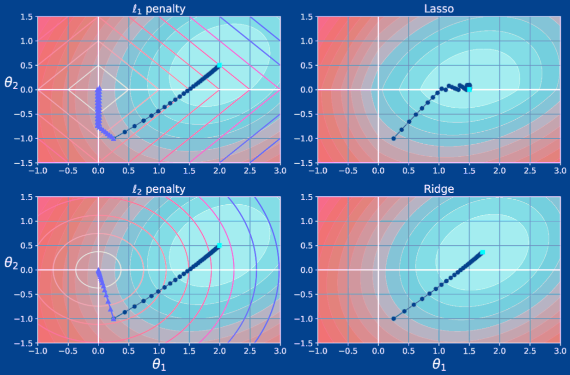
NOTE: Model selection: $\sim$ selection the best capacity/hyperparameter for a given model—NOT choosing the ML algo/model itself!

Model Evaluation

k-fold Cross-Validation Procedure, for *k*=5..



REGULARIZATION



Regularization

Adding a Penalty to the Cost Function

For a linear regressor, our cost function was

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \propto \text{MSE}(\mathbf{X}, \mathbf{y}; \mathbf{w})$$

But now enters a **penalty factor**, Ω, that scaled with α adds extra cost to J,

$$\tilde{J}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \alpha\Omega(\mathbf{w})$$

so this becomes **a-tug-of-war** between the two terms in $\tilde{J}$.

The effect of the added penalty is to:

- put a **contraint** on the norm of the weights, $\mathbf{w}$, disallowing 'em to grow wildely,
- leading to **reduced overfitting**, disabling the model to learn the background noise in the data.

$\mathcal{L}_2$ Regularization

Ridge Penalization

Aka Weight Decay, aka Tikhonov regularization

$$\begin{aligned}\Omega(\mathbf{w}) &= ||\mathbf{w}||_2^2 = \mathbf{w}^\top \mathbf{w} \\ \tilde{J}_{\text{ridge}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha \mathbf{w}^\top \mathbf{w}\end{aligned}$$

with $\mathbf{w} = [w_1 \; w_2 \; \cdots \; w_n]^\top$ without the bias element w_0 in the regulizer term, Ω, and recalling the Euclidiean norm

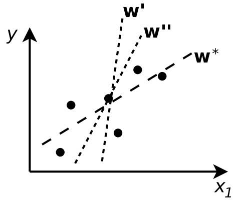
$$\mathcal{L}_2^2 : ||\mathbf{x}||_2^2 = \mathbf{x}^\top \mathbf{x}$$

NOTE: ..and give-or-take some additional $1/2$ or $1/n$ constant, that we do not care about.

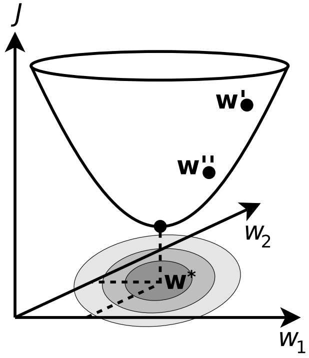
$\mathcal{L}_2$ Regularization

Ridge Penalization

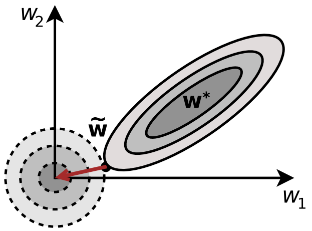
A graphical view for a linear regressor



1D featurespace



3D: ideal convex loss in $J - \mathbf{w}$ space.



2D: flat $w_2 - w_1$ view with some feature scaling.

The tug-of-war: what happens with $\tilde{\mathbf{w}}$,
if $\mathbf{w}^*$ is far from the origin $[w_1, w_2] = (0, 0)$?

$\mathcal{L}_1$ Regularization

Lasso penalization

Now, just replace the $\mathcal{L}_2$ with $\mathcal{L}_1$ and we have the Lasso regularizer

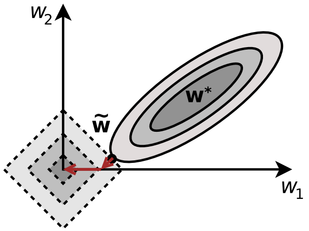
$$\begin{aligned} \Omega(\mathbf{w}) &= ||\mathbf{w}||_1 \\ \tilde{J}_{\text{lasso}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha ||\mathbf{w}||_1 \end{aligned}$$

with the Manhattan norm

$$\mathcal{L}_1 : ||\mathbf{x}||_1 = \sum_{i=1}^{n} \text{abs}(x_i)$$

and the $\mathcal{L}_1$ penalty tends to drive weights to zero:

- automatic feature selection,
- outputs a sparce model,
- i.e few nonzero w's.



$\mathcal{L}_1$ and $\mathcal{L}_2$ Regularization

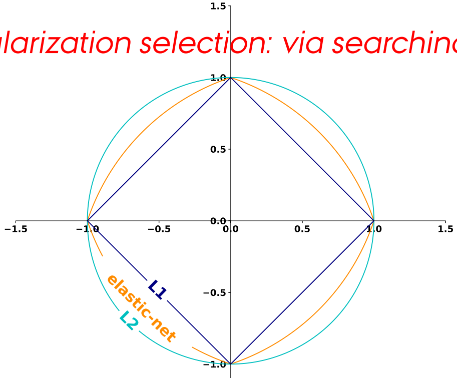
Elastic-net Penalization

And finally a combination of the two: an Elastic-net regularizer

$$\Omega(\mathbf{w}) = \beta||\mathbf{w}||_1 + (1-\beta)||\mathbf{w}||_2^2$$

$$\tilde{J}_{\text{elastic}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha\left(\beta||\mathbf{w}||_1 + (1-\beta)||\mathbf{w}||_2^2\right)$$

Regularization selection: via searching..



HOML Fig 4-19 Explained

You can get a sense of why this is the case by looking at Figure 4-19: on the top-left plot, the background contours (ellipses) represent an unregularized MSE cost function ($\alpha = 0$), and the white circles show the Batch Gradient Descent path with that cost function. The foreground contours (diamonds) represent the ℓ_1 penalty, and the triangles show the BGD path for this penalty only ($\alpha \to \infty$). Notice how the path first reaches $\theta_1 = 0$, then rolls down a gutter until it reaches $\theta_2 = 0$. On the top-right plot, the contours represent the same cost function plus an ℓ_1 penalty with $\alpha = 0.5$. The global minimum is on the $\theta_2 = 0$ axis. BGD first reaches $\theta_2 = 0$, then rolls down the gutter until it reaches the global minimum. The two bottom plots show the same thing but uses an ℓ_2 penalty instead. The regularized minimum is closer to $\boldsymbol{\theta} = \mathbf{0}$ than the unregularized minimum, but the weights do not get fully eliminated.

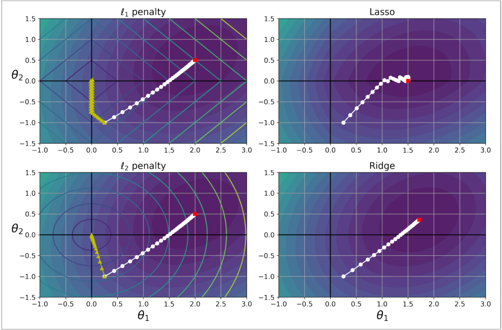


Figure 4-19. Lasso versus Ridge regularization

On the Lasso cost function, the BGD path tends to bounce across the gutter toward the end. This is because the slope changes abruptly at $\theta_2 = 0$. You need to gradually reduce the learning rate in order to actually converge to the global minimum.

HOML fig 4-19 Explained

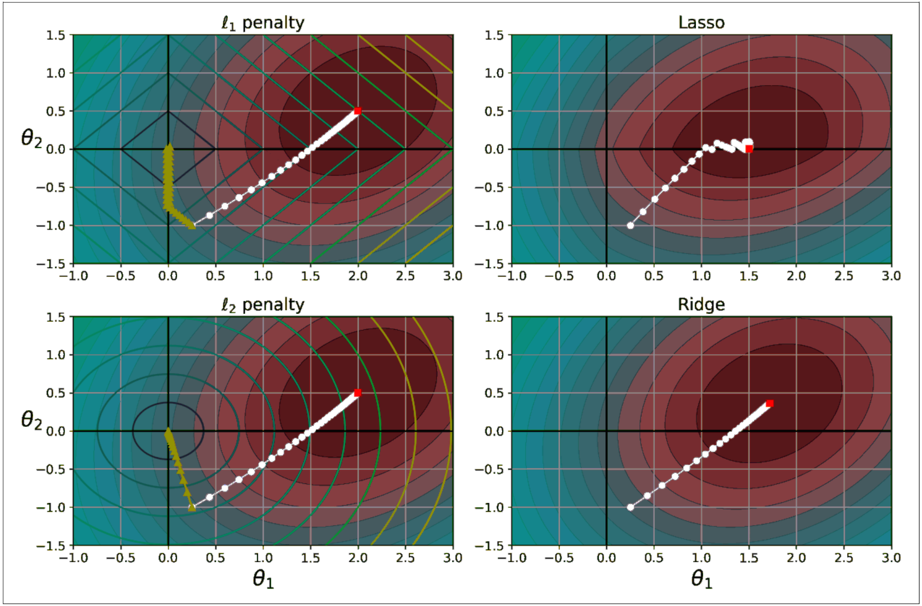
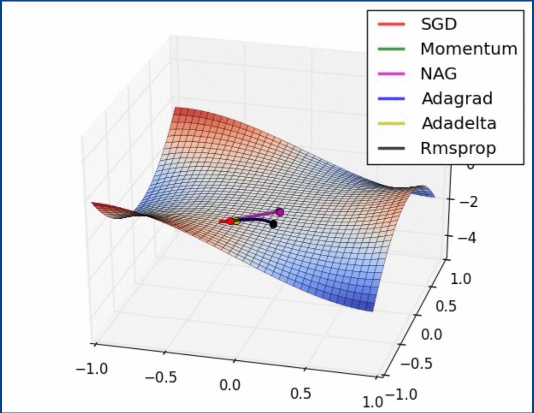
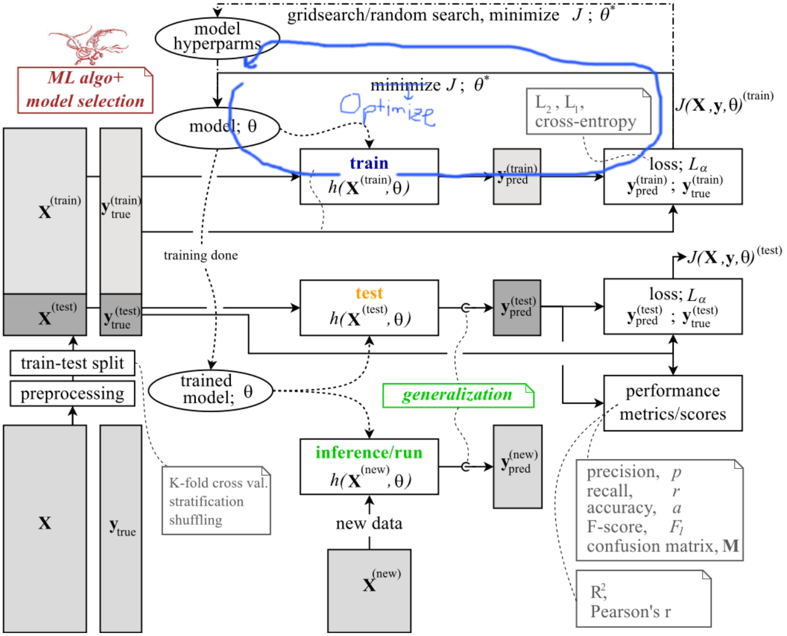


Figure 4-19. Lasso versus Ridge regularization

OPTIMIZERS


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

Optimizers


gridsearch/random search, minimize J ; θ^*
model hyperparms
ML algo+ model selection
minimize J ; θ^*
Optimize
L_2, L_1, cross-entropy
$J(\mathbf{X},\mathbf{y},\theta)^{(\mathrm{train})}$
model; θ
train
$h(\mathbf{X}^{(\mathrm{train})},\theta)$
$\mathbf{y}_{\mathrm{pred}}^{(\mathrm{train})}$
loss; L_α
$\mathbf{y}_{\mathrm{pred}}^{(\mathrm{train})}$; $\mathbf{y}_{\mathrm{true}}^{(\mathrm{train})}$
$\mathbf{X}^{(\mathrm{train})}$
$\mathbf{y}_{\mathrm{true}}^{(\mathrm{train})}$
training done
$J(\mathbf{X},\mathbf{y},\theta)^{(\mathrm{test})}$
test
$h(\mathbf{X}^{(\mathrm{test})},\theta)$
$\mathbf{y}_{\mathrm{pred}}^{(\mathrm{test})}$
loss; L_α
$\mathbf{y}_{\mathrm{pred}}^{(\mathrm{test})}$; $\mathbf{y}_{\mathrm{true}}^{(\mathrm{test})}$
$\mathbf{X}^{(\mathrm{test})}$
$\mathbf{y}_{\mathrm{true}}^{(\mathrm{test})}$
train-test split
preprocessing
trained model; θ
generalization
performance metrics/scores
inference/run
$h(\mathbf{X}^{(\mathrm{new})},\theta)$
$\mathbf{y}_{\mathrm{pred}}^{(\mathrm{new})}$
new data
$\mathbf{X}^{(\mathrm{new})}$
$\mathbf{X}$
$\mathbf{y}_{\mathrm{true}}$
K-fold cross val.
stratification
shuffling
precision, p
recall, r
accuracy, a
F-score, F_1
confusion matrix, $\mathbf{M}$
R^2,
Pearson's r

Optimizers

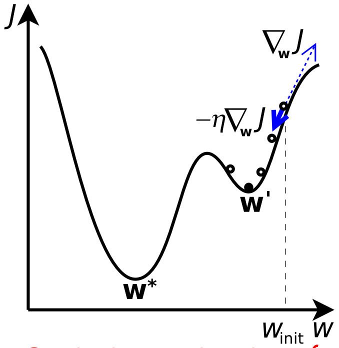
Momentum Optimization

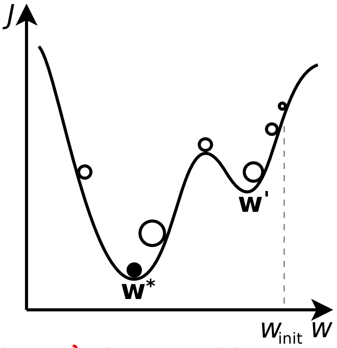
Normal GD algo

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} J$$

but now with added (physical) momentum

$$\begin{aligned} \mathbf{m} &\leftarrow \beta \mathbf{m} - \eta \nabla_{\mathbf{w}} J \\ \mathbf{w} &\leftarrow \mathbf{w} + \mathbf{m} \end{aligned}$$

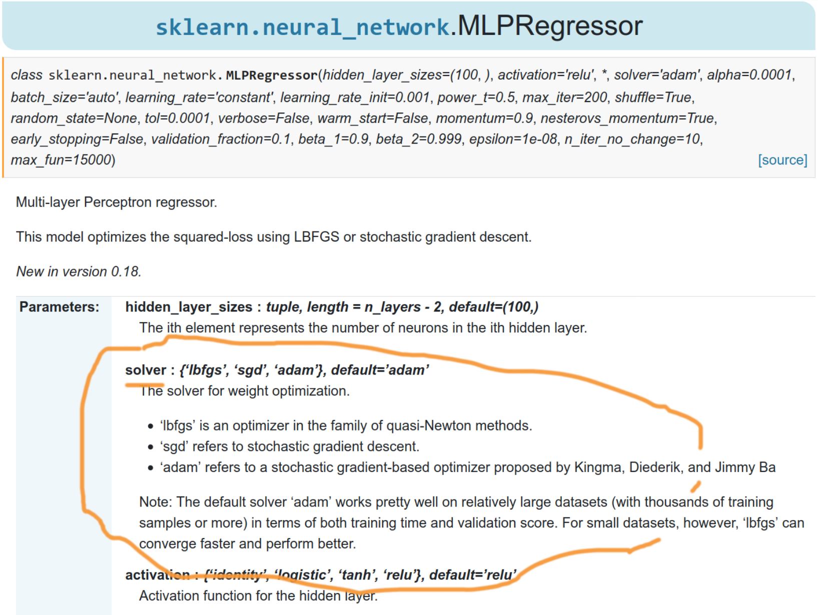




Optimizer selection: (perhaps) via searching..

Optimizers

Or solvers in Scikit-learn..



sklearn.neural_network.MLPRegressor

class `sklearn.neural_network.`**`MLPRegressor`**(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

Multi-layer Perceptron regressor.

This model optimizes the squared-loss using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*

The ith element represents the number of neurons in the ith hidden layer.

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*

The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

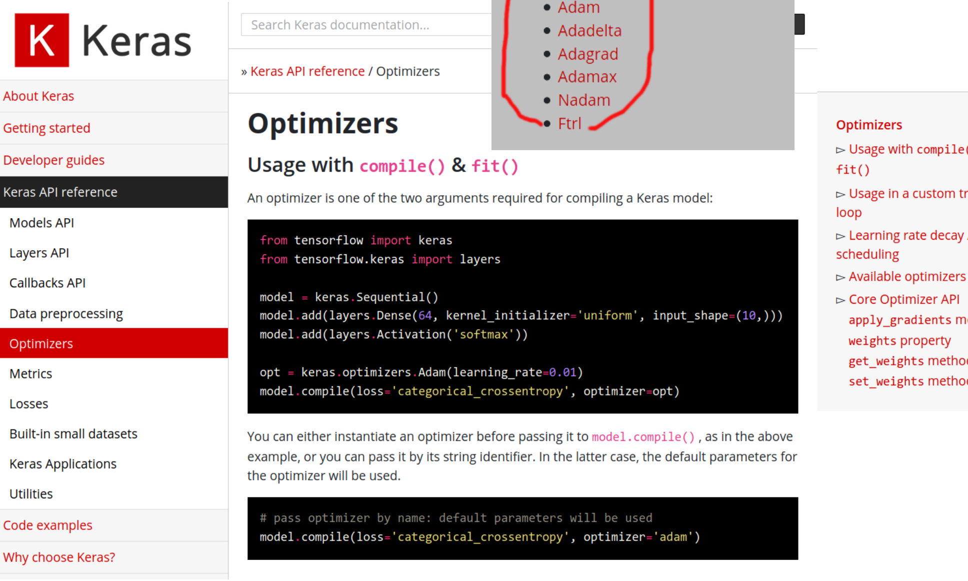
Note: The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*

Activation function for the hidden layer.

Optimizers

Or optimizers in Keras..



Keras

Search Keras documentation...

About Keras
Getting started
Developer guides
Keras API reference
- Models API
- Layers API
- Callbacks API
- Data preprocessing
- Optimizers
- Metrics
- Losses
- Built-in small datasets
- Keras Applications
- Utilities

Code examples
Why choose Keras?

» Keras API reference / Optimizers

Optimizers

Usage with compile() & fit()

An optimizer is one of the two arguments required for compiling a Keras model:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(layers.Activation('softmax'))

opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=opt)
```

You can either instantiate an optimizer before passing it to `model.compile()` , as in the above example, or you can pass it by its string identifier. In the latter case, the default parameters for the optimizer will be used.

```
# pass optimizer by name: default parameters will be used
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

Optimizers
- ▻ Usage with compile
fit()
- ▻ Usage in a custom tr
loop
- ▻ Learning rate decay
scheduling
- ▻ Available optimizers
- ▻ Core Optimizer API
  - apply_gradients m
  - weights property
  - get_weights metho
  - set_weights metho

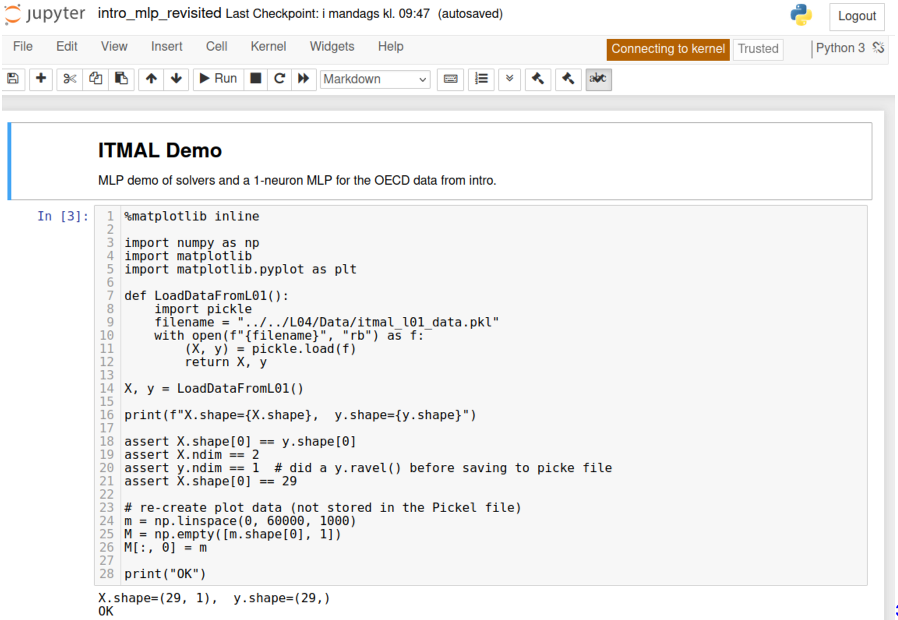
Available optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

K + TensorFlow

Optimizers Demo

`L09/Extra/intro_mlp_revisited.ipynb`


jupyter intro_mlp_revisited Last Checkpoint: i mandags kl. 09:47 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Connecting to kernel
Trusted
Python 3
Run
Markdown

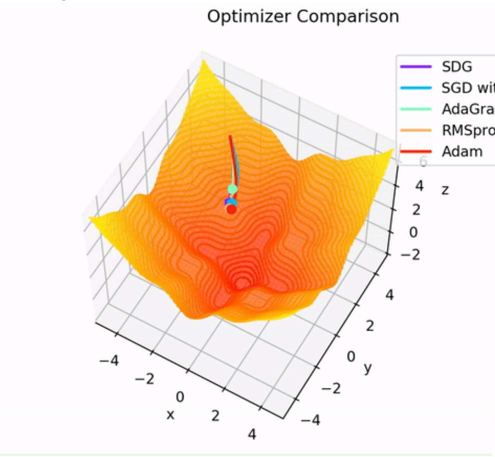
ITMAL Demo

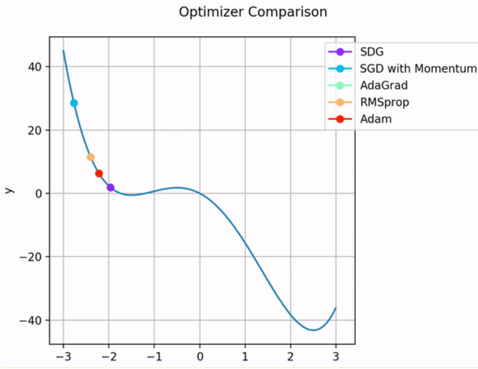
MLP demo of solvers and a 1-neuron MLP for the OECD data from intro.

In [3]:
```
%matplotlib inline

import numpy as np
import matplotlib
import matplotlib.pyplot as plt

def LoadDataFromL01():
    import pickle
    filename = "../../L04/Data/itmal_l01_data.pkl"
    with open(f"{filename}", "rb") as f:
        (X, y) = pickle.load(f)
        return X, y

X, y = LoadDataFromL01()

print(f"X.shape={X.shape},  y.shape={y.shape}")

assert X.shape[0] == y.shape[0]
assert X.ndim == 2
assert y.ndim == 1  # did a y.ravel() before saving to picke file
assert X.shape[0] == 29

# re-create plot data (not stored in the Pickel file)
m = np.linspace(0, 60000, 1000)
M = np.empty([m.shape[0], 1])
M[:, 0] = m

print("OK")
```

```
X.shape=(29, 1),  y.shape=(29,)
OK
```

Optimizers





Sources: Imgur by Alec Radford and
https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d

SEARCHING

ML Algorithm +
Model Selection via Searching

ML Models (or ML algorithms)

Models encountered so far

Some classifiers and regressors..

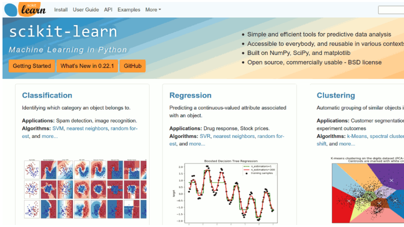
```
sklearn.neighbors.KNeighborsRegressor
sklearn.linear_model.LinearRegression
sklearn.linear_model.SGDClassifier
sklearn.linear_model.SGDRegressor
```

Perhaps..

```
sklearn.naive_bayes.GaussianNB
sklearn.naive_bayes.MultinomialNB
sklearn.svm.SVC
sklearn.svm.SVR
```

and to some degree..

```
sklearn.linear_model.LogisticRegression
sklearn.linear_model.Perceptron
sklearn.neural_network.MLPClassifier
sklearn.neural_network.MLPRegressor
keras.Sequential
```



scikit-learn

Machine Learning in Python

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various context
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Clustering

Automatic grouping of similar objects i

Applications: Customer segmentatio experiment outcomes
Algorithms: k-Means, spectral cluste shift, and more...

Or even more exotic models like..

- superviced ensemble: AdaBoost, Bagging, DecisionTree, RandomForest,..
- semi-supervised: ??
- unsupervised: K-means, manifolds, restricted Boltzmann machines,..
- clustering: K-means

ML Algorithm + Model Selection via Searching

What ML algorithm to choose?

- manual:

  algorithm characteristics, $\mathcal{O}$ complexity, etc.
  browsing through Scikit-learn documentation,
  ...and also based on data assumptions.
- semi-automatic:

  brute-force model search, and fun with python!

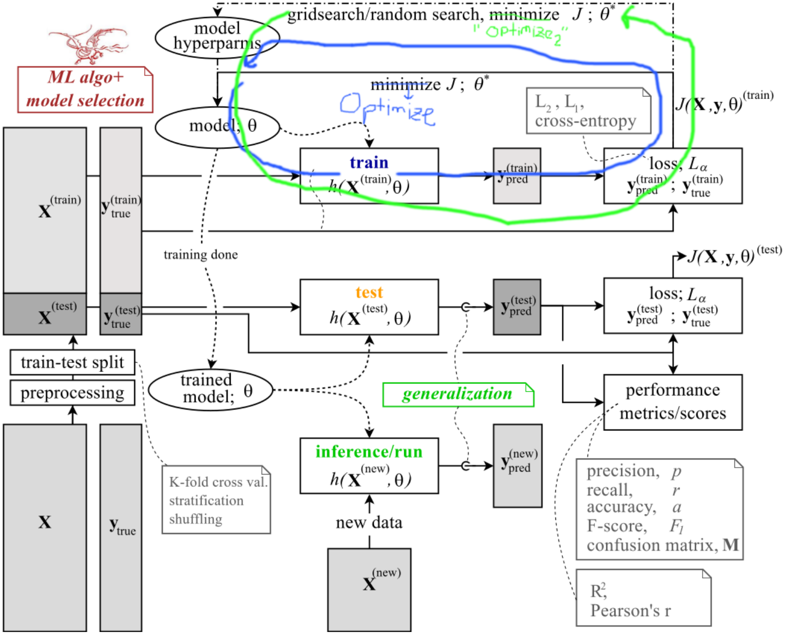
```
models = {
  SVC(gamma="scale"),
  SGDClassifier(tol=1e-3, eta0=0.1),
  GaussianNB()
}

for i in models:
    i.fit(X_train, y_train)
    y_pred_test = i.predict(X_test)
    p = precision_score(y_test, y_pred_test, average='micro')
    print(f'{type(i).__name__:13s}: precision={p:0.2f}')
```

prints..

```
GaussianNB:    p=1.00
SGDClassifier: p=0.93
SVC:           p=0.98
```

NOTE: Python set = `{a, b}`
Python dictionary= `{a:x, b:y}`

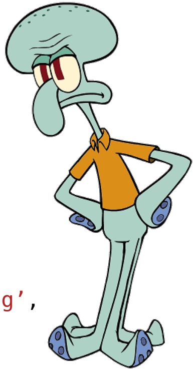
Model Selection via Grid Search



Model Selection via Grid Search

The hyperparamter-set for SGD linear regressor

```
class sklearn.linear_model.SGDRegressor(
  loss    ='squared_loss', penalty     ='l2',
  alpha   =0.0001,         l1_ratio    =0.15,
  tol     =None,           shuffle     =True,
  verbose =0,              epsilon     =0.1,
  eta0    =0.01,           power_t     =0.25,
  n_iter_no_change=5,      warm_start  =False,
  fit_intercept   =True,   max_iter    =None,
  average         =False,  n_iter      =None
  random_state    =None,   learning_rate='invscaling',
  early_stopping  =False,  validation_fraction=0.1
)
```



Search best hyperparameters in a (smaller) set, say

```
model = SGDRegressor()
tuning_parameters = {
  'alpha': [ 0.001, 0.01, 0.1],
  'max_iter': [1, 10, 100, 1000],
  'learning_rate':('constant','optimal','invscaling','adaptive')
}
..
grid_tuned = GridSearchCV(model, tuning_parameters, ..
```

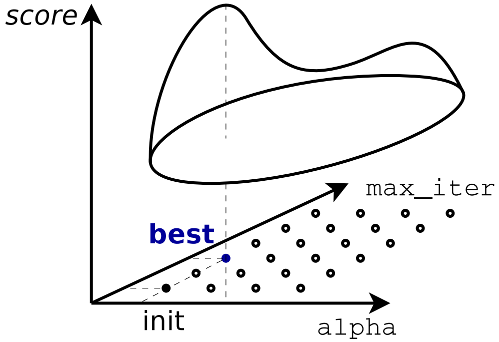
Model Selection via Grid Search

How to select 'best' set of hyperparameter—using bute force?

Gridsearch seen in 3D for the two hyperspace dimensions:

- `alpha` $\in [1, 2, 3, ..]$
- `max_iter` $\in [1, 2, 3, ..]$

(NOTE: linear range for this plot only, should be 1, 10, 100 or similar.)



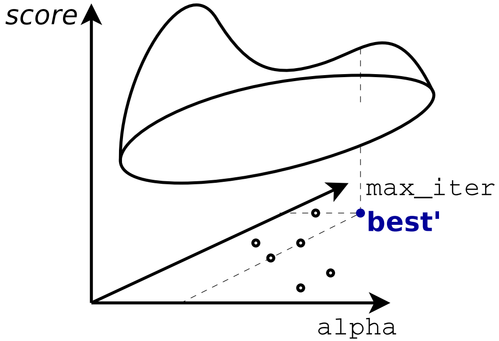
- why `score` and not J on $z-$axis?
- and what if there are many hyperparameters and many combinations? $\rightarrow$ Zzzzzzz!

Model Selection via Randomized Search

How to select 'best' set of hyperparameters—faster than brute force?

Replace `GridSearchCV()` with

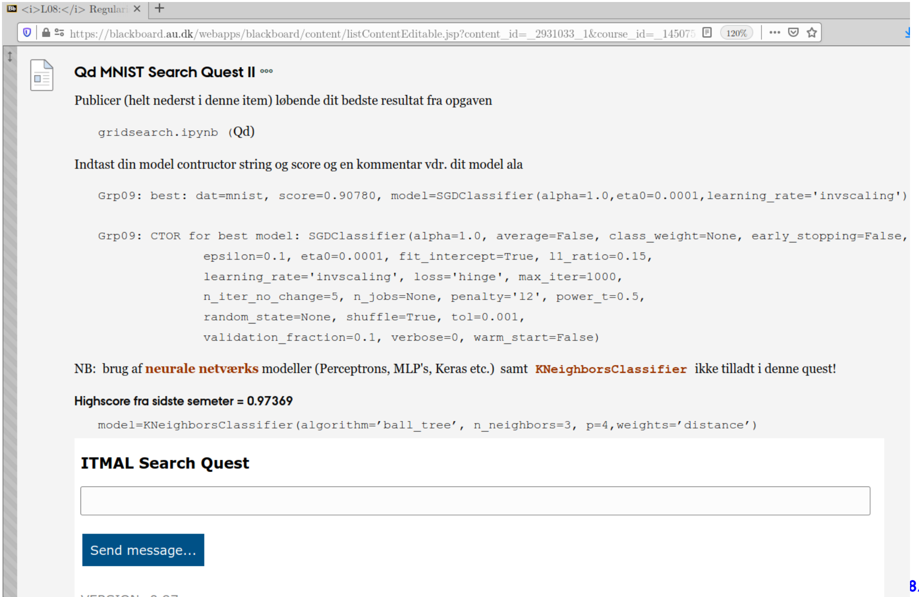
`RandomizedSearchCV(n_iter=100,..)`



- faster, but will not yield the (sub) optimal score maximum,
- ...but does it matter in a huge hyperparameter search-space?

Exercise: `L09/gridsearch.ipynb`

Qd MNIST Search Quest II: Husk at publicer på Brightspace



<i>L08:</i> Regulari
https://blackboard.au.dk/webapps/blackboard/content/listContentEditable.jsp?content_id=_2931033_1&course_id=_145075
120%

Qd MNIST Search Quest II

Publicer (helt nederst i denne item) løbende dit bedste resultat fra opgaven

`gridsearch.ipynb` (Qd)

Indtast din model contructor string og score og en kommentar vdr. dit model ala

```
Grp09: best: dat=mnist, score=0.90780, model=SGDClassifier(alpha=1.0,eta0=0.0001,learning_rate='invscaling')

Grp09: CTOR for best model: SGDClassifier(alpha=1.0, average=False, class_weight=None, early_stopping=False,
              epsilon=0.1, eta0=0.0001, fit_intercept=True, l1_ratio=0.15,
              learning_rate='invscaling', loss='hinge', max_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

NB: brug af **neurale netværks** modeller (Perceptrons, MLP's, Keras etc.) samt `KNeighborsClassifier` ikke tilladt i denne quest!

Highscore fra sidste semeter = 0.97369

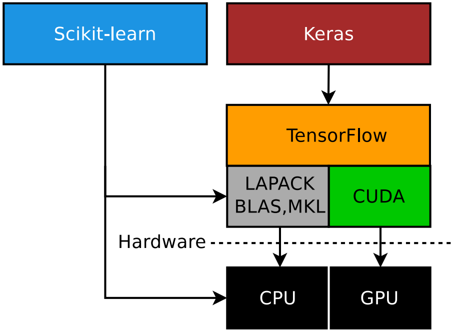
```
model=KNeighborsClassifier(algorithm='ball_tree', n_neighbors=3, p=4,weights='distance')
```

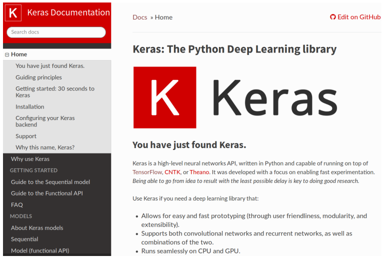
ITMAL Search Quest

Extra Slides..

Keras and Tensorflow

Using the Keras API instead of Scikit-learn or TensorFlow





Keras: The Python Deep Learning library

You have just found Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

NOTE:

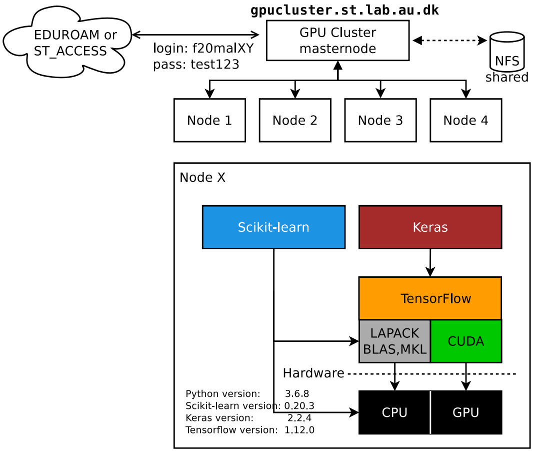
- documentation: `https://keras.io/`
- keras provides a `fit-predict`-interface,
- many similiarities to Scikit-learn,
- but also many differences!

High-Performace-Computing (HPC)

Running on the ASE GPU cluster:

  login=itmal09-e21   (for ITMAL group 9)

  password=imal09-e21_123


gpucluster.st.lab.au.dk
EDUROAM or ST_ACCESS
login: f20malXY
pass: test123
GPU Cluster masternode
NFS shared
Node 1
Node 2
Node 3
Node 4
Node X
Scikit-learn
Keras
TensorFlow
LAPACK BLAS,MKL
CUDA
Hardware
CPU
GPU
Python version: 3.6.8
Scikit-learn version: 0.20.3
Keras version: 2.2.4
Tensorflow version: 1.12.0