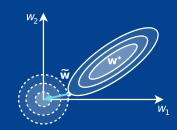




Lesson 09: Regularization, Optimization and Searching

CARSTEN EIE FRIGAARD

SPRING 2025









"A computer program is said to learn from experience E with respect to some class of tasks I 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 (F24) (no exercise. E24/F25)

Optimizers:

(no exercise).

Searching:

Gridsearch,

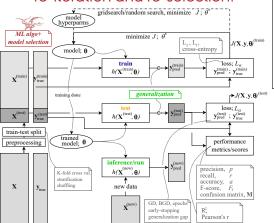
Randomsearch,

Exercise: L09/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!

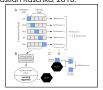


Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning Sebastian Raschka University of Wisconsin-Madison Abstract The correct use of model evaluation, model selection, and algorithm selection techniques is vital in academic machine learning research as well as in many of each technique with references to theoretical and empirical studies. Further, 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 bootstran technique are introduced for estimating the uncertainty of performance estimates, as an alternative to confidence intervals via normal approximation if bootstrapping is trade-off for choosing k is discussed, and practical tips for the optimal choice of it are given based on empirical evidence. Different statistical tests for algorithm comparisons are presented, and strategies for dealing with multiple comparisons alternative methods for algorithm selection, such as the combined F-test 5x2 crossvalidation and nested cross-validation, are recommended for comparing machine

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

Sebastian Raschka, 2018.

learning algorithms when datasets are small



ML Algorithm Selection and Model Selection

Manually Choosing an Algorithm and Tuning a Model..

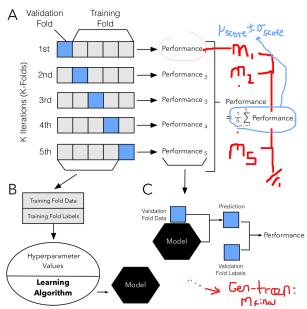
- algorithm selection:
 - choose an algo that 'fits' the problem, ...perhaps via searching??
- model selection:
 - looking for 'optimal' model capacity,
 - tuning model hyperparameters..
 - model weights regularizers..(for NN's)
 - gradient descent optimizers..(for NN's)
- model evaluation:
 - the performance 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!

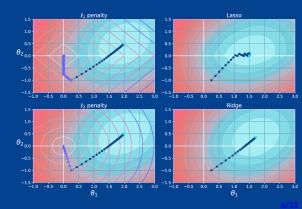


Model Evaluation

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

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

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

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

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, 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

$$\Omega(\mathbf{w}) = ||\mathbf{w}||_2^2 = \mathbf{w}^{\mathsf{T}}\mathbf{w}$$

$$ilde{J}_{ ext{ridge}}(\mathbf{X},\mathbf{y};\mathbf{w}) \ = J(\mathbf{X},\mathbf{y};\mathbf{w}) + lpha \mathbf{w}^{ op} \mathbf{w}$$

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 Euclidean 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 W_1

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 $[w_1, w_2] = (0, 0)$?

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

\mathcal{L}_1 Regularization

Lasso penalization

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

$$\Omega(\mathbf{w}) = ||\mathbf{w}||_1$$

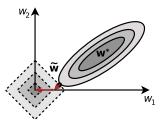
$$\tilde{J}_{\text{lasso}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha ||\mathbf{w}||_1$$

with the Manhattan norm

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

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

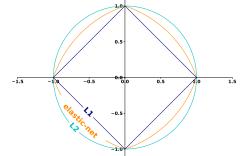
Elastic-net Penalization

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

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

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

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 $\theta=0$ than the unregularized minimum, but the weights do not get fully eliminated.

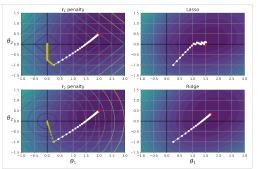


Figure 4-19, Lasso versus Ridge regularization



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

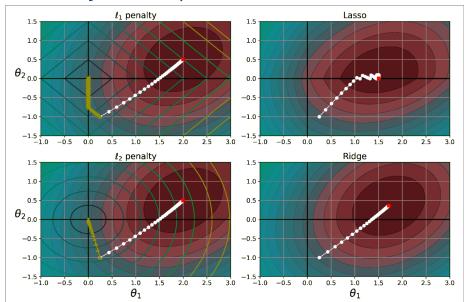
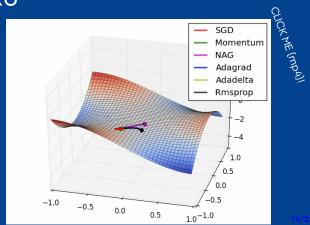
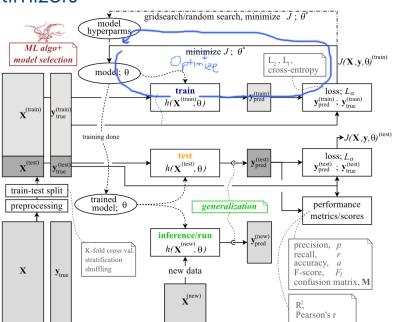


Figure 4-19. Lasso versus Ridge regularization

OPTIMIZERS





Momentum Optimization

Normal GD algo

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

but now with added (physical) momentum

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

Optimizer selection: (perhaps) via searching...

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.

- . 'Ibfgs' is an optimizer in the family of quasi-Newton methods.
- 'sad' 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 converce faster and perform better.

activation: ("identity", "logistic", "tanh", "relu"), default="relu"

Activation function for the hidden layer.

Or optimizers in Keras...



About Keras

Getting started

Models API

Lavers API

Callbacks API

Optimizers

Metrics

Losses Built-in small datasets

Utilities

Code examples

Why choose Keras?

Data preprocessing

Keras Applications

Developer guides

Keras API reference

» Keras API reference / Optimizers

Available optimizers

- SGD
- RMSprop
- Adam
- Adadelta Adagrad
- Adamax
- Nadam
- Etrl .

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.add(layers.Activation('softmax'))

model = keras.Sequential() model.add(lavers.Dense(64, kernel initializer='uniform', input shape=(10,)))

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')

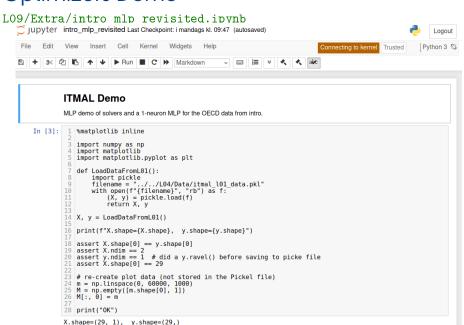


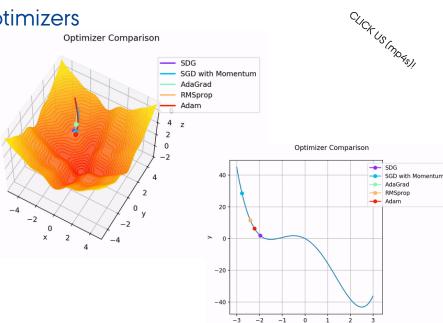
Optimizers □ Usage with compile

- fit() Usage in a custom to
- ▶ Learning rate decay
- scheduling
- Available optimizers
- □ Core Optimizer API apply gradients m weights property get weights metho

set_weights metho

Optimizers Demo





Sources: Imgur by Alec Radford and https://towardsdatascience.com/complete-quide-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

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

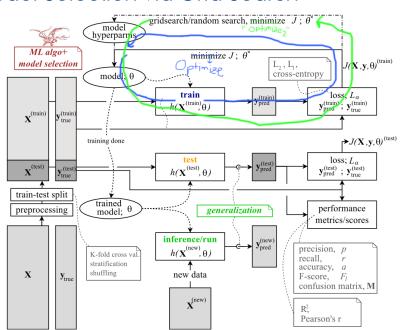
What ML Algorithm to choose at what Model Capacity?

- 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()
                                                prints..
                                                GaussianNB:
                                                               p=1.00
for i in models:
                                                SGDClassifier:
                                                               p = 0.93
    i.fit(X_train, y_train)
                                                               p = 0.98
                                                SVC:
    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}')
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
                                   ='12'.
 alpha =0.0001,
                        ll ratio
                                   =0.15,
 tol =None,
                        shuffle
                                   =True,
 verbose = 0.
                        epsilon
                                   =0.1.
                        power_t = 0.25,
 eta0
        =0.01.
 n_iter_no_change=5,
                        warm_start
                                   =False.
 fit_intercept
                =True.
                        max iter
                                   =None.
              =False,
                       n iter
                                   =None
 average
  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?

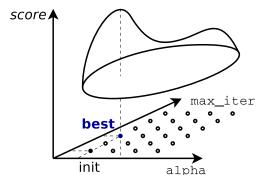
Gridsearch seen in 3D for the two hyperspace dimensions:

▶ $alpha \in [1, 2, 3, ..]$

(NOTE: linear range for this plot only,

▶ $max_iter \in [1, 2, 3, ..]$

should be 1, 10, 100 or similar.)



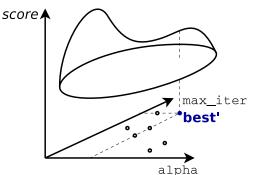
- \blacktriangleright why score and not J on z-axis?
- and what if there are many hyperparameters and many combinations? → Zzzzzzz!

Model Selection via Randomized Search

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

Replace GridSearchCV() with

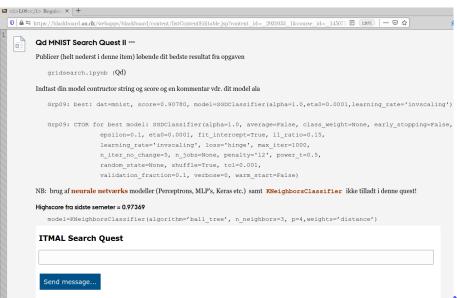
RandomizedSearchCV(n_iter=100,..)



- 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

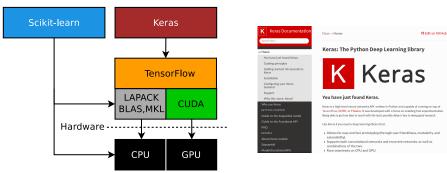


Extra Slides..

Keras and Tensorflow



Using the Keras API instead of Scikit-learn or TensorFlow

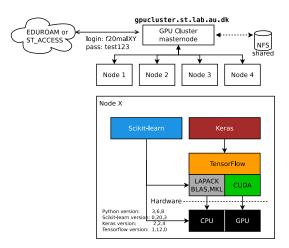


NOTE:

- 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=swmal09f25 (for SWMAL group 9) password=swmal09f25_123



GPU Cluster: Updated Architechture

