Automated Machine Learning
(AutoML)

M. Lindauer F. Hutter

University of Freiburg

Lecture 2:
Design Spaces in Machine Learning

Where are we? The big picture

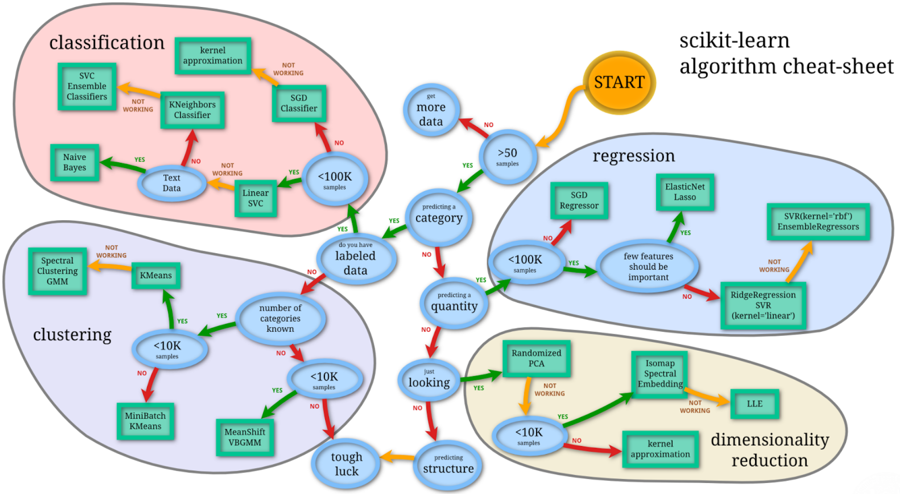
- Introduction
- → Background
  - → Design spaces in ML
  - Evaluation and visualization
- Hyperparameter optimization (HPO)
  - Bayesian optimization
  - Other black-box techniques
  - Speeding up HPO with multi-fidelity optimization
- Pentecost (Holiday) – no lecture
- Architecture search I + II
- Meta-Learning
- Learning to learn & optimize
- Beyond AutoML: algorithm configuration and control
- Project announcement and closing

Learning Goals

After this lecture, you will be able to . . .

- identify design decisions of machine learning algorithms
- explain different types of design decisions and there relations
- create design spaces
- discuss the pros and cons of different design space approaches
- explain design spaces for neural architecture search

Simple Design Decisions: Selection of Algorithm



Simple Design Decisions: Selection of Algorithm

- categorical design decison: $\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \dots, \mathcal{A}_n\}$
  - random forest (RF), support vector machine (SVM), gradient boosting (GB), ...
  - there is no ordering between algorithms
  - set notation

Simple Design Decisions: Selection of Algorithm

- categorical design decison: $\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \ldots, \mathcal{A}_n\}$
  - random forest (RF), support vector machine (SVM), gradient boosting (GB), ...
  - there is no ordering between algorithms
  - set notation
- if we would run all of them and each takes (on average) t seconds: $t \cdot n$ seconds

Simple Design Decisions: Selection of Algorithm

- categorical design decison: $\{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \dots, \mathcal{A}_n\}$
  - random forest (RF), support vector machine (SVM), gradient boosting (GB), ...
  - there is no ordering between algorithms
  - set notation
- if we would run all of them and each takes (on average) t seconds: $t \cdot n$ seconds
- in addition, choose pre-processing algorithm: $\{\mathcal{A}_1^P, \mathcal{A}_2^P, \mathcal{A}_3^P, \dots \mathcal{A}_l^P\}$
  - PCA, feature selection, random kitchen sinks, ...
- if we only use one preprocessor and one ML algorithm, exhaustive search would require: $t \cdot n \cdot l$

Lecture Overview

Design Space of Support Vector Machines

Home | Installation | Documentation | Examples

Previous: sklearn.svm.O... | Next: sklearn.svm.SVR | Up: API Reference

scikit-learn v0.20.3
Other versions

Please **cite us** if you use the software.

sklearn.svm.SVC
Examples using sklearn.svm.SVC

sklearn.svm.SVC

class `sklearn.svm.` **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how gamma, coef0 and degree affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Parameters:

C : ***float, optional (default=1.0)***
Penalty parameter C of the error term.

kernel : ***string, optional (default='rbf')***
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : ***int, optional (default=3)***
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : ***float, optional (default='auto')***
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

Current default is 'auto' which uses 1 / n_features, if `gamma='scale'` is passed then it uses 1 / (n_features * X.var()) as value of gamma. The current default of gamma, 'auto', will change to 'scale' in version 0.22. 'auto_deprecated', a deprecated version of 'auto' is used as a default indicating that no explicit value of gamma was passed.

coef0 : ***float, optional (default=0.0)***
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking : ***boolean, optional (default=True)***
Whether to use the shrinking heuristic.

probability : ***boolean, optional (default=False)***
Whether to enable probability estimates. This must be enabled prior to calling fit, and will slow

Design Space of Support Vector Machines

Hyperparameter Optimization (HPO; informal)

Given

- a dataset
- a set of hyperparameters of a machine learning algorithms
- a cost metric (e.g., predictive error)

we want to find the hyperparameter configuration
that minimizes the cost metric wrt the dataset.

Hyperparameter Types of SVM

C	float hyperparameter
Kernel	categorical hyperparameter
Degree	integer hyperparamerter
gamma	float hyperparamerter
	...

Hyperparameter Types

categorical: set of values (not sorted) with uniform distance

- `kernel` {`linear`, `rbf`, `poly`, `sigmoid`}

Hyperparameter Types

categorical: set of values (not sorted) with uniform distance

- `kernel {linear, rbf, poly, sigmoid}`

ordinal: list of ordered values with uniform distance between neighbors

- no example in SVM design space
- `size [small, medium, large]`

Hyperparameter Types

categorical set of values (not sorted) with uniform distance

- `kernel {linear, rbf, poly, sigmoid}`

ordinal list of ordered values with uniform distance between neighbors

- no example in SVM design space
- `size [small, medium, large]`

integer bounded range of integers

- `degree [1, 5]`

Hyperparameter Types

categorical: set of values (not sorted) with uniform distance
- `kernel {linear, rbf, poly, sigmoid}`

ordinal: list of ordered values with uniform distance between neighbors
- no example in SVM design space
- `size [small, medium, large]`

integer: bounded range of integers
- `degree [1, 5]`

float: bounded range of floats
- `gamma_value [0.0001, 8.0]`

Design Space and Configuration

Design/Configuration Space

The combination of several hyperparameter ranges $\mathbf{\Lambda}_i$ for the i-th hyperparameter creates a design space: $\mathbf{\Lambda} = \mathbf{\Lambda}_1 \times \mathbf{\Lambda}_2 \times \mathbf{\Lambda}_3 \ldots \times \mathbf{\Lambda}_n$

Design Space and Configuration

Design/Configuration Space

The combination of several hyperparameter ranges $\mathbf{\Lambda}_i$ for the i-th hyperparameter creates a design space: $\mathbf{\Lambda} = \mathbf{\Lambda}_1 \times \mathbf{\Lambda}_2 \times \mathbf{\Lambda}_3 \ldots \times \mathbf{\Lambda}_n$

For example, the design space of a SVM would include:

```
kernel categorical {linear, rbf, poly, sigmoid}
degree integer [1, 5]
gamma_value float [0.0001, 8.0]
```

Design Space and Configuration

Design/Configuration Space

The combination of several hyperparameter ranges $\mathbf{\Lambda}_i$ for the i-th hyperparameter creates a design space: $\mathbf{\Lambda} = \mathbf{\Lambda}_1 \times \mathbf{\Lambda}_2 \times \mathbf{\Lambda}_3 \ldots \times \mathbf{\Lambda}_n$

For example, the design space of a SVM would include:

```
kernel categorical {linear, rbf, poly, sigmoid}
degree integer [1, 5]
gamma_value float [0.0001, 8.0]
```

Configuration

An element of the configuration space $\lambda \in \mathbf{\Lambda}$ instantiates each hyperparameter λ_i with a value. For example:

```
{kernel: rbf, gamma_value: 1, degree: 2}
```

Concept of Defaults

- We assume that each algorithm has a default
  - often a robust configuration if you don't want to change it
  - defaults often provided by developer, e.g.,
    in its documentation or the corresponding paper

Concept of Defaults

- We assume that each algorithm has a default
  - often a robust configuration if you don't want to change it
  - defaults often provided by developer, e.g., in its documentation or the corresponding paper
- We use the default to start our search for HPO
  - if we know a reasonable configuration, we should start with a random configuration?
  - Goal: find something which is better than the default

Concept of Defaults

- We assume that each algorithm has a default
  - often a robust configuration if you don't want to change it
  - defaults often provided by developer, e.g., in its documentation or the corresponding paper
- We use the default to start our search for HPO
  - if we know a reasonable configuration, we should start with a random configuration?
  - Goal: find something which is better than the default
  - Pro: Can help us to start in good region of the design space
  - Contra: We might start being trapped in local optimum.

Concept of Defaults

- We assume that each algorithm has a default
  - often a robust configuration if you don't want to change it
  - defaults often provided by developer, e.g., in its documentation or the corresponding paper
- We use the default to start our search for HPO
  - if we know a reasonable configuration, we should start with a random configuration?
  - Goal: find something which is better than the default
  - Pro: Can help us to start in good region of the design space
  - Contra: We might start being trapped in local optimum.

Example: the default `kernel` of the SVM could be the RBF-kernel

`kernel categorical {linear, rbf, poly, sigmoid}[rbf]`

Conditional Dependencies

SVM Example

```
kernel categorical {linear, rbf, poly, sigmoid}[rbf]
degree integer [1, 5][3]
gamma_value float [0.0001, 8.0][2.0]
```

- Some hyperparameters depend on each other
  - `degree` is only active if `kernel` is set to `poly`
  - `gamma_value` is only active if `kernel` is set to `rbf`

Conditional Dependencies

SVM Example

```
kernel categorical {linear, rbf, poly, sigmoid}[rbf]
degree integer [1, 5][3]
gamma_value float [0.0001, 8.0][2.0]
```

- Some hyperparameters depend on each other
  - `degree` is only active if `kernel` is set to `poly`
  - `gamma_value` is only active if `kernel` is set to `rbf`

$\rightsquigarrow$ model such conditional dependencies in configuration space:

```
degree | kernel in {poly}
gamma_value | kernel in {rbf}
```

Remarks (I): Conditional Dependencies

Duplicates of Hyperparameters?

- Sometimes algorithms have the same sub-hyperparameter (with slightly different meanings)
- Two approaches:
  - Duplicate hyperparameter and use conditionals
    $\rightsquigarrow$ larger configuration space
  - a single hyperparameter
    $\rightsquigarrow$ optimizer has to learn dependencies on its own
- Not well studied which way is better under which conditions

Remarks (II): Conditional Dependencies

Imputation

- Inactive hyperparameters can be handled in different ways
- Most trivial approach: imputation of deactivated hyperparameters
  1. Impute with default value
  2. Impute with non-existing value

Remarks (II): Conditional Dependencies

Imputation

- Inactive hyperparameters can be handled in different ways
- Most trivial approach: imputation of deactivated hyperparameters
  1. Impute with default value
  2. Impute with non-existing value
- Risk:
  - confusing for some optimizers (in particular model-based optimizers)

Remarks (II): Conditional Dependencies

Imputation

- Inactive hyperparameters can be handled in different ways
- Most trivial approach: imputation of deactivated hyperparameters
  1. Impute with default value
  2. Impute with non-existing value
- Risk:
  - confusing for some optimizers (in particular model-based optimizers)
- One of the open challenges: best way to handle conditionals

Forbidden Constraints

- Combinations of settings can be forbidden
- For example: $a \leq b$

Forbidden Constraints

- Combinations of settings can be forbidden
- For example: $a \leq b$

$\leadsto$ Try to avoid such constraints
because sampling in constrained spaces gets much harder

- Sometimes constraints can be rewritten:

```
a float [0,1][0]
b float [0,1][0]
a <= b
```

Rewrite:

```
a float [0,1][0]
c float [0,1][0]
```

with $b = a + c \leadsto$ b might be larger than 1!

Expert Knowledge: Transformations

- Expert knowledge can help to guide hyperparameter optimization
- For example, some hyperparameters might not be sampled uniformly

Expert Knowledge: Transformations

- Expert knowledge can help to guide hyperparameter optimization
- For example, some hyperparameters might not be sampled uniformly

For example, regularization hyperparameter of SVM:

```
C float [0.001, 1000.0][1.0]
```

- the distance between 999.9 and 1000.0 should not be the same as between 0.001 and 0.101

Expert Knowledge: Transformations

- Expert knowledge can help to guide hyperparameter optimization
- For example, some hyperparameters might not be sampled uniformly

For example, regularization hyperparameter of SVM:

```
C float [0.001, 1000.0][1.0]
```

- the distance between 999.9 and 1000.0 should not be the same as between 0.001 and 0.101

$\leadsto$ We might want to sample here from from a log-scale

```
C float [0.001, 1000.0][1.0] log
```

Design Space of Support Vector Machines

Home Installation Documentation Examples

Previous sklearn.svm.O... | Next sklearn.svm.SVR | Up API Reference

scikit-learn v0.20.3
Other versions

Please **cite us** if you use the software.

`sklearn.svm.SVC`
Examples using `sklearn.svm.SVC`

`sklearn.svm`.SVC

class `sklearn.svm.` **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how gamma, coef0 and degree affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Parameters:

C : ***float, optional (default=1.0)***
Penalty parameter C of the error term.

kernel : ***string, optional (default='rbf')***
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : ***int, optional (default=3)***
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

gamma : ***float, optional (default='auto')***
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

Current default is 'auto' which uses 1 / n_features, if `gamma='scale'` is passed then it uses 1 / (n_features * X.var()) as value of gamma. The current default of gamma, 'auto', will change to 'scale' in version 0.22. 'auto_deprecated', a deprecated version of 'auto' is used as a default indicating that no explicit value of gamma was passed.

coef0 : ***float, optional (default=0.0)***
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking : ***boolean, optional (default=True)***
Whether to use the shrinking heuristic.

probability : ***boolean, optional (default=False)***
Whether to enable probability estimates. This must be enabled prior to calling fit, and will slow

Configuration Space of SVM

```
Hyperparameters:
  C float [0.001, 1000.0][1.0] log
  coef0 float [0.0, 10.0][0.0]
  degree integer [1, 5][3]
  gamma categorical {auto, value}[auto]
  gamma_value float [0.0001, 8.0][1.0]
  kernel categorical {linear, rbf, poly, sigmoid}[poly]
  shrinking categorical {true, false}[true]
Conditions:
  coef0 | kernel in {poly, sigmoid}
  degree | kernel in {poly}
  gamma | kernel in {rbf, poly, sigmoid}
  gamma_value | gamma in {value}
```

Optimization of Random Seeds?

Is the random seed a valid design decision?

Output: trained model

If the output of your AutoML tool is a trained model:

- your goal is to obtain the best trained model

$\rightsquigarrow$ tune your random seed!

Optimization of Random Seeds?

Is the random seed a valid design decision?

Output: trained model

If the output of your AutoML tool is a trained model:

- your goal is to obtain the best trained model

$\rightsquigarrow$ tune your random seed!

Output: best configuration

If the output of your AutoML tool is the best configuration:

- your goal is to obtain a configuration that performs well after refitting

$\rightsquigarrow$ don't tune your random seed!

$\rightsquigarrow$ obtain configuration that performs well on many random seeds

Lecture Overview

Algorithm: Randomized Regression Tree

Algorithm 1: `BuildTree()`

Input : $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i \in \{1 \dots |D|\}}$, attributes A

if *current tree depth is larger than threshold* t_d **then**
  return *Leaf with* $\mathbf{y}$

if *samples size* $|D|$ *is smaller than threshold* t_n **then**
  return *Leaf with* $\mathbf{y}$

Algorithm: Randomized Regression Tree

Algorithm 2: `BuildTree()`

Input : $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i \in \{1 \ldots |D|\}}$, attributes A

```
if current tree depth is larger than threshold t_d then
    return Leaf with y
if samples size |D| is smaller than threshold t_n then
    return Leaf with y
A' ← subsample k attributes from A;
let v be the best split value of attribute a ∈ A' according to criterion c;
```

Algorithm 3: `BuildTree()`

Input : $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i \in \{1 \dots |D|\}}$, attributes A

if *current tree depth is larger than threshold* t_d **then**
  return *Leaf with* $\mathbf{y}$
if *samples size* $|D|$ *is smaller than threshold* t_n **then**
  return *Leaf with* $\mathbf{y}$
$A' \leftarrow$ subsample k attributes from A;
let v be the *best split value* of attribute $a \in A'$ according to criterion c;
Create edges with constraint $\mathbf{x}^{(i)}.a \leq v$ and $\mathbf{x}^{(i)}.a > v$;
BuildTree($\{(\mathbf{x}^{(i)}, y^{(i)}) \in D | \mathbf{x}^{(i)}.a \leq v\}$, A);
BuildTree($\{(\mathbf{x}^{(i)}, y^{(i)}) \in D | \mathbf{x}^{(i)}.a > v\}$, A);
return *current node*

Algorithm: Regression Random Forest

Algorithm 4: `BuildForest()`

Input : $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i \in \{1 \ldots |D|\}}$, attributes A

for $i \in \{1 \ldots n\}$ **do**
  $D' \leftarrow$ bootstrap D with d points;
  $T_i \leftarrow$ BuildTree(D', A);

Algorithm: Regression Random Forest

Algorithm 6: `BuildForest()`

Input : $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i \in \{1 \ldots |D|\}}$, attributes A

for $i \in \{1 \ldots n\}$ **do**
  $D' \leftarrow$ bootstrap D with d points;
  $T_i \leftarrow$ BuildTree(D', A);

Algorithm 7: `Predict()`

Input : Data point x

for $i \in \{1 \ldots n\}$ **do**
  $y_i \leftarrow T_i$.predict(x);
$y \leftarrow \frac{1}{n} \sum_i^n y_i$;

Configuration Space of Regression Random Forest

Task: [5min]

- What are design decisions of a regression random forest?
- What could be a reasonable configuration space?

Configuration Space of Regression Random Forest

Task: [5min]

- What are design decisions of a regression random forest?
- What could be a reasonable configuration space?

```
n_trees integer [1, 100][10] log
d_bootstrap float [0.1, 1.0][1.0]
c_criterion categorical {mse, mae}
k_attributes float [0.1, 1.0][0.6]
t_n integer [1,10][1]
t_d integer [2,1024][1024]log
```

Levels of Programming by Optimization [Hoos 2012]

Level 0 Already exposed hyperparameters

Levels of Programming by Optimization [Hoos 2012]

Level 0 Already exposed hyperparameters

Level 1 Make hardwired design choices accessible

Levels of Programming by Optimization [Hoos 2012]

Level 0 Already exposed hyperparameters

Level 1 Make hardwired design choices accessible

Level 2 Design choices are already considered during software development

Levels of Programming by Optimization [Hoos 2012]

- Level 0 Already exposed hyperparameters
- Level 1 Make hardwired design choices accessible
- Level 2 Design choices are already considered during software development
- Level 3 Seek design choices in software development

Levels of Programming by Optimization [Hoos 2012]

Level 0 Already exposed hyperparameters

Level 1 Make hardwired design choices accessible

Level 2 Design choices are already considered during software development

Level 3 Seek design choices in software development

Remark: The field of search-based software engineering is closely related to AutoML.

Lecture Overview

Hyperparameter Optimization

Hyperparameter Optimization (HPO)

Let

- λ be the hyperparameters of an ML algorithm A with domain Λ,
- D_{opt} be a training set which is split into D_{train} and D_{valid}
- $\mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the loss of A_λ trained on D_{train} and evaluated on D_{valid}.

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this loss:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Hyperparameter Optimization

Hyperparameter Optimization (HPO)

Let

- λ be the hyperparameters of an ML algorithm A with domain Λ,
- D_{opt} be a training set which is split into D_{train} and D_{valid}
- $\mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the loss of A_λ trained on D_{train} and evaluated on D_{valid}.

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this loss:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remark:

- $\arg\min$ returns a set of optimal points of a given function. It suffices to find one element of this set and thus use $\in$ instead of $=$.

Extend HPO

AutoML includes

- Hyperparameter Optimization (HPO)
- Algorithm selection
- ... (and more)

Extend HPO

AutoML includes

- Hyperparameter Optimization (HPO)
- Algorithm selection
- . . . (and more)

$\rightsquigarrow$ How to select an algorithm and a hyperparameter configuration?

CASH: Combined Algorithm Selection and Hyperparameter Optimization

Let

- $\mathbf{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$ be a set of algorithms
- $\mathbf{\Lambda}$ be a set of hyperparameters of each machine learning algorithm $\mathcal{A}_i$
- D_{opt} be a training set which is split into D_{train} and D_{valid}
- $\mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the loss of A_λ trained on D_{train} and evaluated on D_{valid}.

we want to find the best combination of algorithm $\mathcal{A} \in \mathbf{A}$ and its hyperparameter configuration $\lambda \in \mathbf{\Lambda}$ minimizing:

$$(\mathcal{A}^*, \lambda^*) \in \operatorname*{arg\,min}_{\mathcal{A}\in\mathbf{A}, \lambda\in\mathbf{\Lambda}} \mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Representation of CASH

- top-level hyperparameter to select algorithm
- conditional constraints for all algorithm-specific hyperparameters

Representation of CASH

- top-level hyperparameter to select algorithm
- conditional constraints for all algorithm-specific hyperparameters

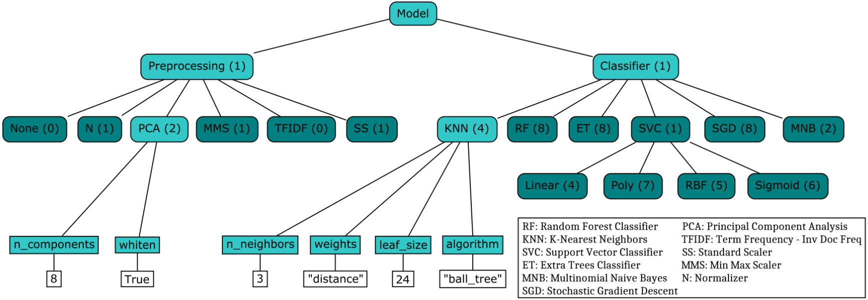
```
algo categorical {SVM, RF, DNN}[RF]

n_tree integer [10,100][10]
n_tree | algo in {RF}

gamma float {0.0001,8}[1]
gamma | algo in {SVM]

...
```

Representation of CASH



Source: [Komer et al. 2019]

Lecture Overview

Bounded Configuration Spaces

So far, we assumed that each hyperparameter has a pre-defined domain.

Problems:

- for pipelines, we might don't know the size of the pipeline
  - some components could be part of the pipeline multiple times

Bounded Configuration Spaces

So far, we assumed that each hyperparameter has a pre-defined domain.

Problems:

- for pipelines, we might don't know the size of the pipeline
  - some components could be part of the pipeline multiple times
- sometimes we don't know a good range for a hyperparameter
  - too large range $\rightsquigarrow$ very hard optimization problem
  - too small range $\rightsquigarrow$ we might miss high-performance areas

Bounded Configuration Spaces

So far, we assumed that each hyperparameter has a pre-defined domain.

Problems:

- for pipelines, we might don't know the size of the pipeline
    - some components could be part of the pipeline multiple times
- sometimes we don't know a good range for a hyperparameter
    - too large range $\leadsto$ very hard optimization problem
    - too small range $\leadsto$ we might miss high-performance areas

$\leadsto$ How can we design such configuration spaces?

Pipeline: Boolean Options

```
component_1 categorical {True, False}[True]
component_2 categorical {True, False}[False]
component_3 categorical {True, False}[False]
...
```

- Hard to optimize/not applicable if
  - we also have to find the ordering of the components
  - if there is an upper bound on the number of components
    $\leadsto$ leads to many invalid configurations
  - each component should be chosen more than once

Pipeline: Fixed Pipelines Size

```
step_1 categorical {comp1, comp2, comp3, ..., none}[comp1]
step_2 categorical {comp1, comp2, comp3, ..., none}[comp2]
step_3 categorical {comp1, comp2, comp3, ..., none}[none]
...
```

- encode each step of the pipeline with a choice between all possible components
- Hard to optimize/not applicable if
  - we don't know a good upper bound of the pipeline size
  - there is an upper bound on how often a component can be chosen

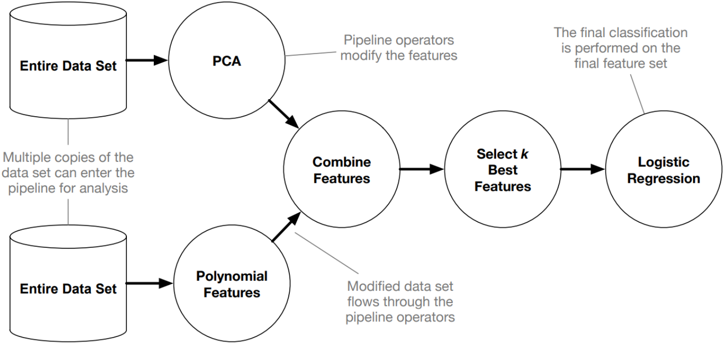
Pipeline: Fixed Pipelines Size

```
step_1 categorical {comp1, comp2, comp3, ..., none}[comp1]
step_2 categorical {comp1, comp2, comp3, ..., none}[comp2]
step_3 categorical {comp1, comp2, comp3, ..., none}[none]
...
```

- encode each step of the pipeline with a choice between all possible components
- Hard to optimize/not applicable if
  - we don't know a good upper bound of the pipeline size
  - there is an upper bound on how often a component can be chosen

- If we don't need to specify the maximum length of the pipeline this design is equivalent to search in a tree of possible pipelines

Pipeline: TPOT [Olson et al. 2016]



- TPOT searches in a space of tree-based pipelines
- Pipeline can potentially grow in size
- Challenge: avoid illegal pipelines

Bounds: Distributions instead of Bounds (HyperOpt [Bergstra et al. 2013])

- Instead of a range, HyperOpt also allows for user-defined distributions
  - sampling of gamma (of SVM) could be done according to log-normal distribution (with given statistics)
- Advantage:
  allows for flexible definition of expert knowledge
- Disadvantage:
  hard to find a good distribution if expert knowledge is limited

Bounds: Increasing Bounds [Shahriari et al. 2015]

1. start with peaked distributions
   s.t. it is very unlikely to sample outside of fairly narrow bounds
2. increase width of distribution over time
   to search in larger areas over time

Bounds: Increasing Bounds [Shahriari et al. 2015]

1. start with peaked distributions
   s.t. it is very unlikely to sample outside of fairly narrow bounds
2. increase width of distribution over time
   to search in larger areas over time

Remark:

- similar ideas can be used for safe optimization
  - quite important if a failed configuration incurs great (monetary) costs, e.g., a robot is destroyed because of its configuration [Sui et al. 2015]

Lecture Overview

Design of Neural Networks

To train a deep neural network, we have many crucial design decisions

Design of Neural Networks

To train a deep neural network, we have many crucial design decisions

- number of layers
- number of neurons in each layer
- activation functions
- skip connections
- global architecture (MLP, ResNet, DenseNet, ...)
- regularization
  - batch norm, weight decay, dropout, mixup, cut-out, ...
- optimizer hyperparameters
  - type of optimizer (SGD, Adam, ...)
  - learning rate
  - momentum
  - learning rate schedule
- ...

$\rightsquigarrow$ joint global optimization of hyperparameters and architecture!

Neural Architecture Search (NAS)

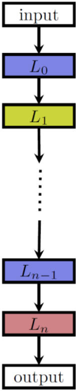
Let

- $\mathcal{A}$ be a neural network
- $\mathbf{\Lambda}$ be a design space
  defining the architecture of a deep neural network
- D_{opt} be a training set which is split into D_{train} and D_{valid}
- $\mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the loss of A_λ trained on D_{train} and evaluated on D_{valid}.

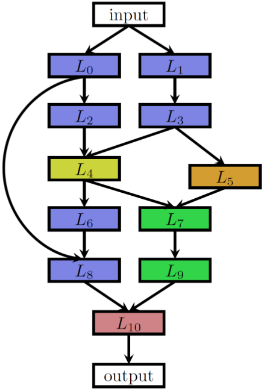
we want to find the architecture $\lambda \in \mathbf{\Lambda}$ minimizing:

$$\lambda^* \in \arg\min_{\lambda \in \mathbf{\Lambda}} \mathcal{L}(\mathcal{A}(\lambda), \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Global NAS



Chain-structured space
(different colours:
different layer types)



More complex space
with multiple branches
and skip connections

Neural Architecture Search (NAS) – Remarks

- yet another hyperparameter problem?
  - $\rightarrow$ we will see in later sessions how we exploit expert knowledge about neural networks to go beyond black-box HPO

Neural Architecture Search (NAS) – Remarks

- yet another hyperparameter problem?
  $\rightarrow$ we will see in later sessions how we exploit expert knowledge about neural networks to go beyond black-box HPO

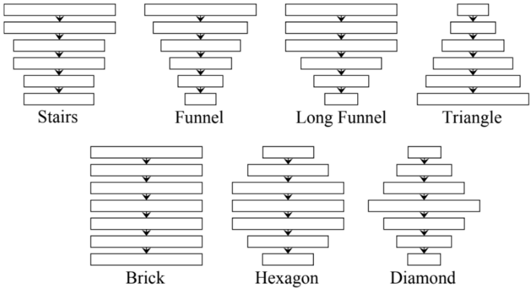
- Current practice:
  - hyperparameters (e.g., of the optimizer) are tuned manual or independently from the architecture

Neural Architecture Search (NAS) – Remarks

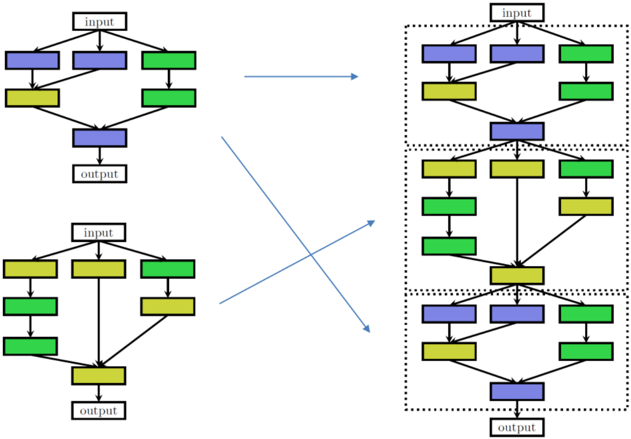
- yet another hyperparameter problem?
  - → we will see in later sessions how we exploit expert knowledge about neural networks to go beyond black-box HPO
- Current practice:
  - hyperparameters (e.g., of the optimizer) are tuned manual or independently from the architecture
- Better practice:
  - jointly optimize hyperparameters and architecture design [Zela et al. 2018]

Shapes of Deep Neural Networks [Kotila 2017]

- Many networks designed by humans follow a pattern
- Whether this is a good idea is not well studied
- Advantage: The number of hyperparameters is smaller
  - E.g., instead of tuning the number of neurons in each layer ($\rightsquigarrow$ one hyperparameter per layer), a few hyperparameters to define shape



Cell NAS

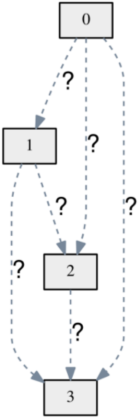


Two possible cells

Architecture composed of stacking together individual cells

$\rightsquigarrow$ Search for cells and repeat these in the final architecture n times.

Flow of Tensors through Operators



- each node is a tensor (i.e., a latent representation of the input data)
- between nodes operators change the data (e.g., convolution or max pooling)
    - includes no-op operators to deactivate edges

Source: [Liu et al. 2019]

Learning Goals

Now, you should be able to ...

- identify design decisions of machine learning algorithms
- explain different types of design decisions and there relations
- create design spaces
- discuss the pro and cons of different design space approaches
- explain design spaces for neural architecture search

Literature [These are links]

- [Programming by Optimization. Hoos 2012]
- [AutoWEKA and CASH. Thornton et al. 2013]
- [TPOT. Olson and Moore. 2019]
- [Hyperopt-Sklearn. Komer et al. 2019]
- [Unbounded Bayesian Optimization via Regularization. Shahriari et al. 2016]
- [DARTS: Differentiable Architecture Search. Liu et al. 2019]