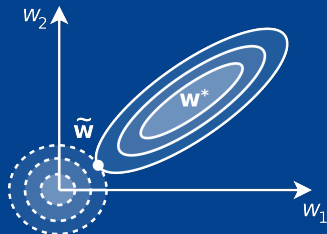


MACHINE LEARNING

LESSON 7: Optimization and Searching

CARSTEN EIE FRIGAARD
SPRING 2019



L07: Optimization and Searching

Agenda

- ▶ BB | Hand-ins Deadlines | [Selected J1 hand-ins](#)

L07: Optimization and Searching

Agenda

- ▶ BB | Hand-ins Deadlines | [Selected J1 hand-ins](#)
- ▶ GPU Cluster via EDUROAM or VPN/au-access
(but loss of port 22)

L07: Optimization and Searching

Agenda

- ▶ BB | Hand-ins Deadlines | [Selected J1 hand-ins](#)
- ▶ GPU Cluster via EDUROAM or VPN/au-access
(but loss of port 22)
- ▶ Regularization and Optimization
Preprocessing of data (no-exe),
Optimizers (no-exe),
Regulizers,
Exercise: [L07/regulizers.ipynb](#)

L07: Optimization and Searching

Agenda

- ▶ BB | Hand-ins Deadlines | [Selected J1 hand-ins](#)
- ▶ GPU Cluster via EDUROAM or VPN/au-access
(but loss of port 22)
- ▶ Regularization and Optimization
Preprocessing of data (no-exe),
Optimizers (no-exe),
Regulizers,
Exercise: [L07/regulizers.ipynb](#)
- ▶ Searching
Model selection and model searching (no-exe),
Gridsearch,
Randomsearch,
Exercise: [L07/gridsearch.ipynb](#)

Preprocessing of Data

Scaling, Standardization, Normalization...

Why the need for preprocessing?

*Standardization of datasets is a **common requirement for many machine learning estimators** [...]; they might **behave badly** if the individual features do not more or less look like standard normally distributed data. [...]*

[<https://scikit-learn.org/stable/modules/preprocessing.html>]

Preprocessing of Data

Scaling, Standardization, Normalization...

Why the need for preprocessing?

*Standardization of datasets is a **common requirement for many machine learning estimators** [..]; they might **behave badly** if the individual features do not more or less look like standard normally distributed data. [..]*

[<https://scikit-learn.org/stable/modules/preprocessing.html>]

Standardization of a feature vector \mathbf{x} , giving \mathbf{x}' mean zero, and standard deviation one

$$\mathbf{x}' = \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$$

Preprocessing of Data

Scaling, Standardization, Normalization...

Why the need for preprocessing?

*Standardization of datasets is a **common requirement for many machine learning estimators** [..]; they might **behave badly** if the individual features do not more or less look like standard normally distributed data. [..]*

[<https://scikit-learn.org/stable/modules/preprocessing.html>]

Standardization of a feature vector \mathbf{x} , giving \mathbf{x}' mean zero, and standard deviation one

$$\mathbf{x}' = \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$$

What kind of estimators needs standardization?

→ **Neural networks (NNs)** in particular!

Preprocessing of Data

Scaling, Standardization, Normalization...

Why the need for preprocessing?

*Standardization of datasets is a **common requirement for many machine learning estimators** [..]; they might **behave badly** if the individual features do not more or less look like standard normally distributed data. [..]*

[<https://scikit-learn.org/stable/modules/preprocessing.html>]

Standardization of a feature vector \mathbf{x} , giving \mathbf{x}' mean zero, and standard deviation one

$$\mathbf{x}' = \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}$$

What kind of estimators needs standardization?

→ **Neural networks (NNs)** in particular!

Regularization and optimization:

→ **WARM-UP** for the coming use of NNs!

Preprocessing of Data

Scaling, Standardization, Normalization...

Common preprocessing methods

- ▶ standardization, aka mean/std scaling, $\frac{x-\mu}{\sigma}$
`sklearn.preprocessing.StandardScaler`

Preprocessing of Data

Scaling, Standardization, Normalization...

Common preprocessing methods

- ▶ standardization, aka mean/std scaling, $\frac{x-\mu}{\sigma}$
`sklearn.preprocessing.StandardScaler`
- ▶ min/max, or abs scaling, say $x \in [0; 1]$
`sklearn.preprocessing.MinMaxScaler`
`sklearn.preprocessing.MaxAbsScaler`

Preprocessing of Data

Scaling, Standardization, Normalization...

Common preprocessing methods

- ▶ standardization, aka mean/std scaling, $\frac{x-\mu}{\sigma}$
`sklearn.preprocessing.StandardScaler`
- ▶ min/max, or abs scaling, say $x \in [0; 1]$
`sklearn.preprocessing.MinMaxScaler`
`sklearn.preprocessing.MaxAbsScaler`
- ▶ normalization, giving unit norm
`sklearn.preprocessing.normalize`

Preprocessing of Data

Scaling, Standardization, Normalization...

Common preprocessing methods

- ▶ standardization, aka mean/std scaling, $\frac{x-\mu}{\sigma}$
`sklearn.preprocessing.StandardScaler`
- ▶ min/max, or abs scaling, say $x \in [0; 1]$
`sklearn.preprocessing.MinMaxScaler`
`sklearn.preprocessing.MaxAbsScaler`
- ▶ normalization, giving unit norm
`sklearn.preprocessing.normalize`

Potential problems:

- ▶ outliers,
- ▶ NaNs (Not-a-Number)

Preprocessing of Data

Scaling, Standardization, Normalization in a Pipeline

Use preprocessing as first state in a fit-predict **pipeline**

Preprocessing of Data

Scaling, Standardization, Normalization in a Pipeline

Use preprocessing as first state in a fit-predict **pipeline**

```
1  from sklearn.preprocessing import StandardScaler
2  from sklearn.pipeline import make_pipeline
3  from sklearn.naive_bayes import GaussianNB
4
5  mypipeline = make_pipeline(
6      StandardScaler(),
7      GaussianNB()
8  )
9
10 ..
11 mypipeline.fit(X_train, y_train)
12 ..
13 mypipeline.predict(X_test)
```

[[GITMAL],L07/Extra/standardization_demo.ipynb]

Preprocessing of Data

Scaling, Standardization, Normalization in a Pipeline

Use preprocessing as first state in a fit-predict **pipeline**

```
1  from sklearn.preprocessing import StandardScaler
2  from sklearn.pipeline import make_pipeline
3  from sklearn.naive_bayes import GaussianNB
4
5  mypipeline = make_pipeline(
6      StandardScaler(),
7      GaussianNB()
8  )
9
10 ..
11 mypipeline.fit(X_train, y_train)
12 ..
13 mypipeline.predict(X_test)
```

[[GITMAL],L07/Extra/standardization_demo.ipynb]

Remember: scale train and test equally!

Regularization

Adding a Penalty to the Cost Function

For a linear regressor, our cost function was

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

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 , adding extra cost to J , and scaled with α

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

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 , adding extra cost to J , and scaled with α

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

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

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 , adding extra cost to J , and scaled with α

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

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

The effect of the added penalty is to:

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

\mathcal{L}_2 Regularization

Ridge Penalisation

Aka Weight Decay, aka Tikhonov regularization

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

$$\tilde{J}_{\text{ridge}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha \mathbf{w}^\top \mathbf{w}$$

\mathcal{L}_2 Regularization

Ridge Penalisation

Aka Weight Decay, aka Tikhonov regularization

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

$$\tilde{J}_{\text{ridge}}(\mathbf{X}, \mathbf{y}; \mathbf{w}) = J(\mathbf{X}, \mathbf{y}; \mathbf{w}) + \alpha \mathbf{w}^\top \mathbf{w}$$

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

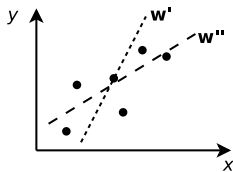
$$\mathcal{L}_2^2 : \|\mathbf{x}\|_2^2 = \mathbf{x}^\top \mathbf{x}$$

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

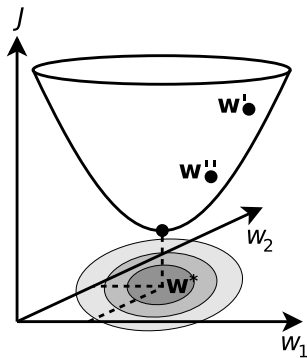


2D $y - x$ space,
here only 1D shown.

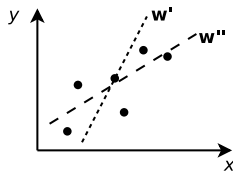
\mathcal{L}_2 Regularization

Ridge Penalization

A graphical view for a linear regressor



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

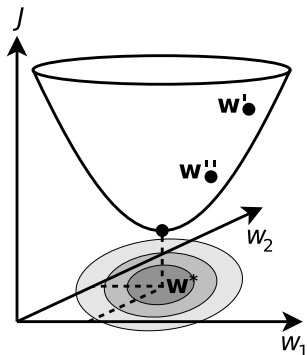


2D $y - x$ space,
here only 1D shown.

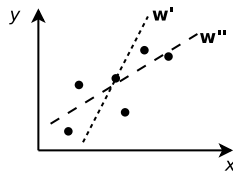
\mathcal{L}_2 Regularization

Ridge Penalization

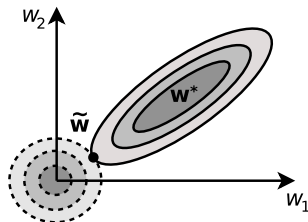
A graphical view for a linear regressor



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



2D $y - x$ space,
here only 1D shown.

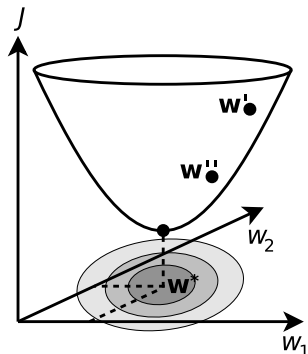


2D: flat $w_2 - w_1$ view with
some feature scaling.

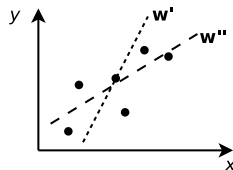
\mathcal{L}_2 Regularization

Ridge Penalization

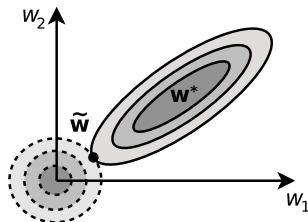
A graphical view for a linear regressor



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



2D $y - x$ space,
here only 1D shown.



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

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

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

\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

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

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

Elastic-net Penalisation

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

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

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

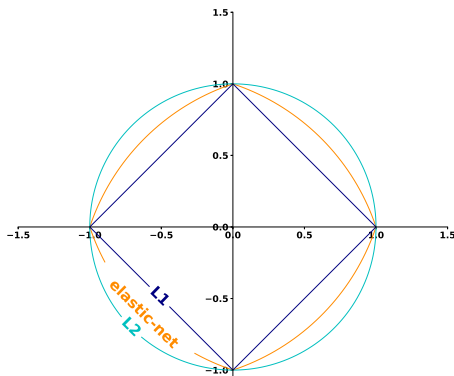
\mathcal{L}_1 and \mathcal{L}_2 Regularization

Elastic-net Penalisation

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

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

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



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

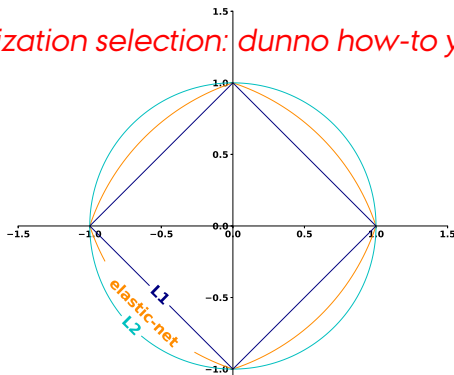
Elastic-net Penalisation

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

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

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

Regularization selection: dunno how-to yet!??



Optimizers

Momentum Optimization

Normal GD algo

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

Optimizers

Momentum Optimization

Normal GD algo

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

but now with added (physical) momentum

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

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{m}$$

Optimizers

Momentum Optimization

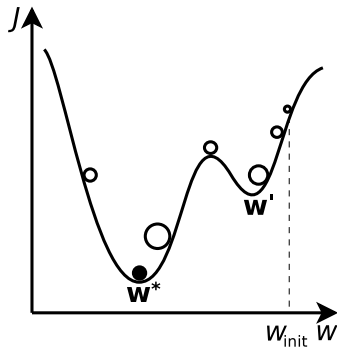
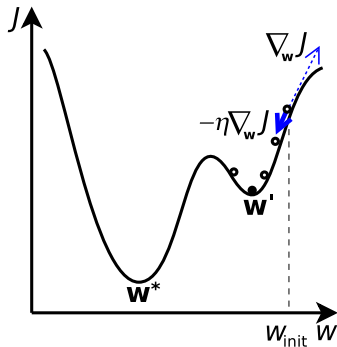
Normal GD algo

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

but now with added (physical) momentum

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

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{m}$$

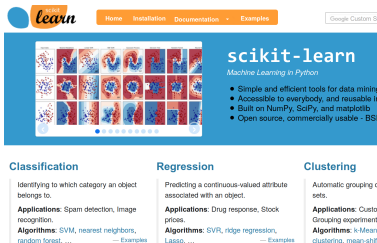


ML Models

Models Encountered so far

Some classifiers and regressors..

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



The screenshot shows the scikit-learn website. At the top, there is a navigation bar with links for Home, Installation, Documentation, and Examples. The main header features the scikit-learn logo and the tagline "Machine Learning in Python". Below this, a grid of 16 small images displays various machine learning model outputs, such as handwritten digit recognition and face detection. To the right of the grid, a list of bullet points highlights the library's features: simple and efficient tools for data mining, accessibility to everybody, built on NumPy, SciPy, and matplotlib, and being open source and commercially usable. The page is divided into three columns: Classification, Regression, and Clustering. Each column provides a brief definition, applications, and algorithms. The Classification column describes identifying categories, with applications like spam detection and algorithms like SVM and random forest. The Regression column describes predicting continuous values, with applications like drug response and algorithms like SVR and Lasso. The Clustering column describes automatic grouping of sets, with applications like customer grouping and algorithms like k-Means and mean-shift.

scikit-learn
Machine Learning in Python

- Simple and efficient tools for data mining
- Accessible to everybody, and reusable in
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD

Classification	Regression	Clustering
Identifying to which category an object belongs to.	Predicting a continuous-valued attribute associated with an object.	Automatic grouping of sets.
Applications: Spam detection, image recognition.	Applications: Drug response, Stock prices.	Applications: Customer grouping, experimental design.
Algorithms: SVM, nearest neighbors, random forest, ...	Algorithms: SVR, ridge regression, Lasso, ...	Algorithms: k-Means, hierarchical clustering, mean-shift.

ML Models

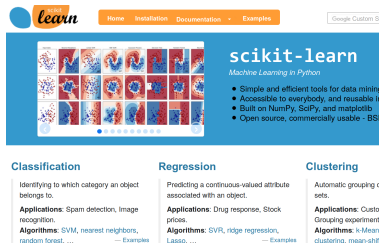
Models Encountered so far

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
```



The screenshot shows the scikit-learn website. At the top, there is a navigation bar with links for Home, Installation, Documentation, and Examples. The main header features the scikit-learn logo and the tagline "Machine Learning in Python". Below this, a grid of 16 small images displays various machine learning model outputs, such as handwritten digit recognition and face detection. To the right of the grid, the text "scikit-learn" is prominently displayed, followed by "Machine Learning in Python" and a list of bullet points highlighting the library's features: "Simple and efficient tools for data mining", "Accessible to everybody, and reusable in various contexts", "Built on NumPy, SciPy, and matplotlib", and "Open source, commercially usable - BSD license". Below the header, the page is divided into three columns: "Classification", "Regression", and "Clustering". Each column contains a brief description of the task, a list of applications, and a list of algorithms. For example, the "Classification" section describes identifying categories and lists applications like spam detection and image recognition, along with algorithms like SVM and random forest. The "Regression" section describes predicting continuous values and lists applications like drug response and stock prices, with algorithms like SVR and ridge regression. The "Clustering" section describes automatic grouping of data sets and lists applications like customer segmentation and grouping experiments, with algorithms like k-Means and hierarchical clustering.

scikit-learn
Machine Learning in Python

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

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of data sets.

Applications: Customer segmentation, Grouping experiments.

Algorithms: k-Means, hierarchical clustering, mean-shift

ML Models

Models Encountered so far

Some classifiers and regressors..

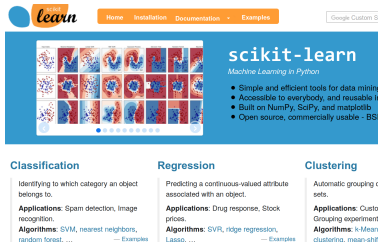
```
sklearn.neighbors.KNeighborsRegressor  
sklearn.linear_model.LinearRegression  
sklearn.linear_model.SGDClassifier  
sklearn.linear_model.SGDRegressor
```

Perhaps...

```
sklearn.naive_bayes.GaussianNB  
sklearn.naive_bayes.MultinomialNB
```

Not really or not in depth

```
sklearn.linear_model.Perceptron  
sklearn.linear_model.LogisticRegression  
sklearn.svm.SVC  
sklearn.svm.SVR  
sklearn.neural_network.MLPClassifier  
sklearn.neural_network.MLPRegressor
```



The screenshot shows the scikit-learn website. At the top, there is a navigation bar with links for Home, Installation, Documentation, and Examples. The main header features the scikit-learn logo and the tagline "Machine Learning in Python". Below this, a grid of 16 small images displays various machine learning model outputs, such as handwritten digit recognition and face detection. To the right of the grid, the text "scikit-learn" is prominently displayed, followed by "Machine Learning in Python" and a list of bullet points highlighting the library's features: "Simple and efficient tools for data mining", "Accessible to everybody, and reusable in various contexts", "Built on NumPy, SciPy, and matplotlib", and "Open source, commercially usable - BSD license". Below the main header, the page is divided into three columns: "Classification", "Regression", and "Clustering". Each column contains a brief description of the task, a list of applications, and a list of algorithms. For example, the "Classification" column describes identifying categories and lists applications like spam detection and image recognition, along with algorithms like SVM and random forest. The "Regression" column describes predicting continuous values and lists applications like drug response and stock prices, with algorithms like SVR and Lasso. The "Clustering" column describes automatic grouping of data sets and lists applications like customer segmentation and grouping experiments, with algorithms like k-Means and mean-shift.

scikit-learn
Machine Learning in Python

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

Classification	Regression	Clustering
Identifying to which category an object belongs to.	Predicting a continuous-valued attribute associated with an object.	Automatic grouping of data sets.
Applications: Spam detection, image recognition.	Applications: Drug response, Stock prices.	Applications: Customer segmentation, Grouping experiments.
Algorithms: SVM, nearest neighbors, random forest, ...	Algorithms: SVR, ridge regression, Lasso, ...	Algorithms: k-Means, hierarchical clustering, mean-shift.

ML Models

Models Encountered so far

Some classifiers and regressors..

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

Perhaps...

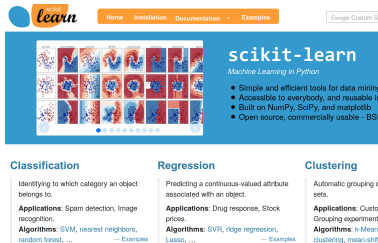
```
sklearn.naive_bayes.GaussianNB  
sklearn.naive_bayes.MultinomialNB
```

Not really or not in depth

```
sklearn.linear_model.Perceptron  
sklearn.linear_model.LogisticRegression  
sklearn.svm.SVC  
sklearn.svm.SVR  
sklearn.neural_network.MLPClassifier  
sklearn.neural_network.MLPRegressor
```

Or even more exotic models like..

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



The screenshot shows the scikit-learn website. At the top, there's a navigation bar with links: Home, Installation, Documentation, and Examples. The main header features the scikit-learn logo and the text "Machine Learning in Python". Below this, there are three columns: Classification, Regression, and Clustering. Each column contains a brief description, applications, and algorithms. The Classification column mentions spam detection and image recognition. The Regression column mentions drug response and stock prices. The Clustering column mentions automatic grouping of sets. At the bottom right, there's a small image of a person with a surprised expression.

scikit-learn
Machine Learning in Python

- Simple and efficient tools for data mining
- Accessible to everybody, and reusable in
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD

Classification	Regression	Clustering
Identifying to which category an object belongs to.	Predicting a continuous-valued attribute associated with an object.	Automatic grouping of sets.
Applications: Spam detection, image recognition.	Applications: Drug response, Stock prices.	Applications: Customer grouping, experimental design.
Algorithms: SVM, nearest neighbors, random forest, ...	Algorithms: SVR, ridge regression, Lasso, ...	Algorithms: k-Means, hierarchical clustering, mean-shift.

ML Model Selection and Searching

What ML model to choose?

Model selection

- ▶ manual:

- model characteristics, \mathcal{O} complexity, etc.
 - browsing through Scikit-learn documentation,
 - ...and also based on data assumptions.

ML Model Selection and Searching

What ML model to choose?

Model selection

- ▶ manual:
 - model characteristics, \mathcal{O} complexity, etc.
 - browsing through Scikit-learn documentation,
 - ...and also based on data assumptions.
- ▶ semi-automatic:
 - model search, and fun with python!

ML Model Selection and Searching

What ML model to choose?

Model selection

► manual:

model characteristics, \mathcal{O} complexity, etc.
browsing through Scikit-learn documentation,
...and also based on data assumptions.

► semi-automatic:

model search, and fun with python!

```
1 models = {  
2     SVC(gamma="scale"),  
3     SGDClassifier(tol=1e-3, eta0=0.1),  
4     GaussianNB()  
5 }  
6  
7 for i in models:  
8     i.fit(X_train, y_train)  
9     y_pred_test = i.predict(X_test)  
10    p = precision_score(y_test, y_pred_test, average='micro')  
11    print(f'{type(i).__name__:13s}: precision={p:0.2f}')
```

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

ML Model Selection and Searching

What ML model to choose?

Model selection

► manual:

model characteristics, \mathcal{O} complexity, etc.
browsing through Scikit-learn documentation,
...and also based on data assumptions.

► semi-automatic:

model search, and fun with python!

```
1 models = {  
2     SVC(gamma="scale"),  
3     SGDClassifier(tol=1e-3, eta0=0.1),  
4     GaussianNB()  
5 }  
6  
7 for i in models:  
8     i.fit(X_train, y_train)  
9     y_pred_test = i.predict(X_test)  
10    p = precision_score(y_test, y_pred_test, average='micro')  
11    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 dictionary= {a:x, b:y}

Model Hyperparameters Grid Search

The hyperparameter-set for SGD linear regressor

```
1 class sklearn.linear_model.SGDRegressor(  
2     loss      ='squared_loss', penalty      ='l2',  
3     alpha     =0.0001,      l1_ratio      =0.15,  
4     tol       =None,        shuffle       =True,  
5     verbose   =0,           epsilon       =0.1,  
6     eta0      =0.01,        power_t     =0.25,  
7     n_iter_no_change=5,     warm_start  =False,  
8     fit_intercept =True,    max_iter    =None,  
9     average     =False,    n_iter      =None  
10    random_state  =None,    learning_rate='invscaling',  
11    early_stopping=False,   validation_fraction=0.1  
12    )
```

Model Hyperparameters Grid Search

The hyperparameter-set for SGD linear regressor

```
1  class sklearn.linear_model.SGDRegressor(  
2      loss      ='squared_loss', penalty      ='l2',  
3      alpha     =0.0001,      l1_ratio      =0.15,  
4      tol       =None,        shuffle       =True,  
5      verbose   =0,           epsilon      =0.1,  
6      eta0      =0.01,        power_t     =0.25,  
7      n_iter_no_change=5,      warm_start  =False,  
8      fit_intercept =True,     max_iter    =None,  
9      average     =False,     n_iter      =None  
10     random_state =None,     learning_rate='invscaling',  
11     early_stopping =False,   validation_fraction=0.1  
12 )
```



Model Hyperparameters Grid Search

The hyperparameter-set for SGD linear regressor

```
1  class sklearn.linear_model.SGDRegressor(  
2      loss      ='squared_loss', penalty      ='l2',  
3      alpha     =0.0001,      l1_ratio      =0.15,  
4      tol       =None,        shuffle       =True,  
5      verbose   =0,           epsilon      =0.1,  
6      eta0      =0.01,        power_t     =0.25,  
7      n_iter_no_change=5,      warm_start  =False,  
8      fit_intercept =True,    max_iter    =None,  
9      average     =False,    n_iter      =None  
10     random_state  =None,    learning_rate='invscaling',  
11     early_stopping=False,   validation_fraction=0.1  
12 )
```



Search best hyperparameters in a (smaller) set, say

Model Hyperparameters Grid Search

The hyperparameter-set for SGD linear regressor

```
1 class sklearn.linear_model.SGDRegressor(  
2     loss      = 'squared_loss', penalty      = 'l2',  
3     alpha     = 0.0001,               l1_ratio    = 0.15,  
4     tol       = None,                 shuffle     = True,  
5     verbose   = 0,                   epsilon     = 0.1,  
6     eta0      = 0.01,                power_t     = 0.25,  
7     n_iter_no_change=5,               warm_start  = False,  
8     fit_intercept = True,             max_iter    = None,  
9     average     = False,             n_iter      = None  
10    random_state  = None,             learning_rate='invscaling',  
11    early_stopping=False,             validation_fraction=0.1  
12 )
```



Search best hyperparameters in a (smaller) set, say

```
1 model = SGDClassifier()  
2 tuning_parameters = {  
3     'alpha': [ 0.001, 0.01, 0.1],  
4     'max_iter': [1, 10, 100, 100],  
5     'learning_rate': ('constant', 'optimal', 'invscaling', 'adaptive')  
6 }  
7 ..  
8 grid_tuned = GridSearchCV(model, tuning_parameters, ..
```

Model Hyperparameters Grid Searching

How to set hyperparameters optimally?

Gridsearch seen in 3D for the two hyperspace dimensions:

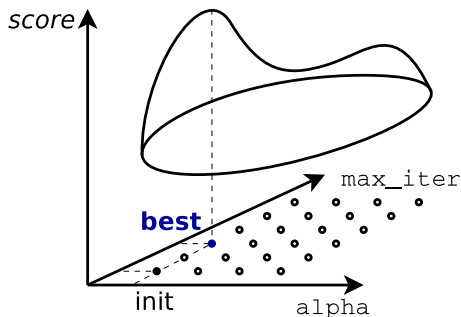
- ▶ $\alpha \in [1, 2, 3..]$ (NOTE: linear range for this plot only!),
- ▶ $\text{learnig_rate} \in [1, 2, 3..]$

Model Hyperparameters Grid Searching

How to set hyperparameters optimally?

Gridsearch seen in 3D for the two hyperspace dimensions:

- ▶ $\alpha \in [1, 2, 3..]$ (NOTE: linear range for this plot only!),
- ▶ $\text{learnig_rate} \in [1, 2, 3..]$

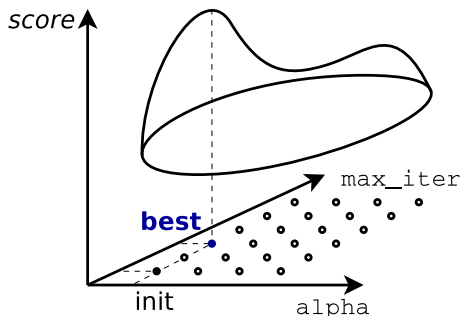


Model Hyperparameters Grid Searching

How to set hyperparameters optimally?

Gridsearch seen in 3D for the two hyperspace dimensions:

- ▶ $\alpha \in [1, 2, 3..]$ (NOTE: linear range for this plot only!),
- ▶ $\text{learnig_rate} \in [1, 2, 3..]$



But, what if there are many hyperparameters and many combinations? → Zzzzzzz!

Model Hyperparameters Random Searching

How to set hyperparameters faster but less optimally?

Replace `GridSearchCV()` with

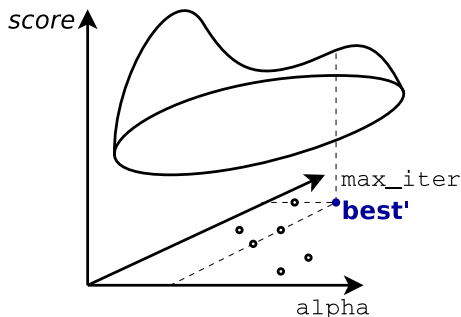
`RandomizedSearchCV(n_iter=100,...)`

Model Hyperparameters Random Searching

How to set hyperparameters faster but less optimally?

Replace `GridSearchCV()` with

`RandomizedSearchCV(n_iter=100,...)`

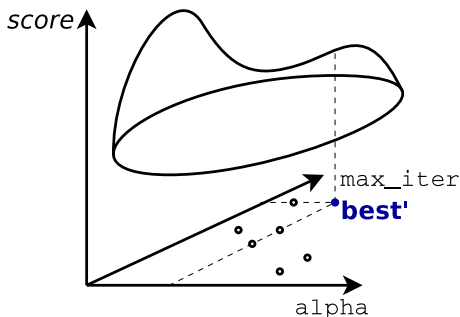


Model Hyperparameters Random Searching

How to set hyperparameters faster but less optimally?

Replace `GridSearchCV()` with

`RandomizedSearchCV(n_iter=100,...)`



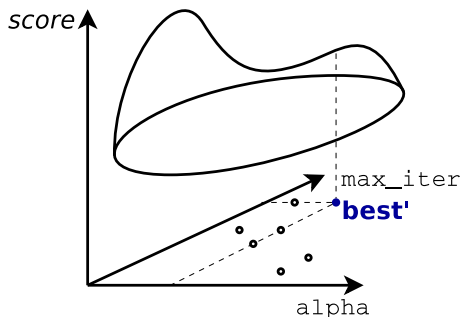
Faster, but will not yield the optimal score maximum,

Model Hyperparameters Random Searching

How to set hyperparameters faster but less optimally?

Replace `GridSearchCV()` with

`RandomizedSearchCV(n_iter=100,...)`



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