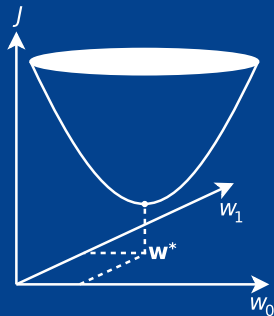


MACHINE LEARNING

LESSON 4: Training I

CARSTEN EIE FRIGAARD
SPRING 2019



L04: Training I

Agenda

- ▶ Discuss your WIKI contribution...(5 min)
- ▶ Exercise from L03:
`Exercise: L03/metrics.ipynb`
- ▶ L04 Training I: *Training a linear regression model*
`Exercise: L04/linear_regression_1.ipynb`
`Exercise: L04/linear_regression_2.ipynb`
- ▶ After the class:
yet another WIKI page,
deadline *L06*, 05-03-2019
(in two weeks).
- ▶ Next lesson: L05 Training II:
Training and model concepts
- ▶ NOTE: on page # in [HOML]
...up to 7 editions.

Editor: Nicole Tache
Production Editor: Nicholas Adams
Copyeditor: Rachel Monaghan
Proofreader: Charles Roumeliotis

March 2017: First Edition

Revision History for the First Edition

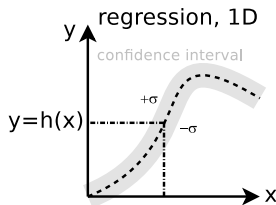
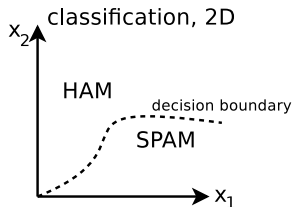
2017-03-10: First Release

RESUMÉ: Classification vs. Regression

Given the following

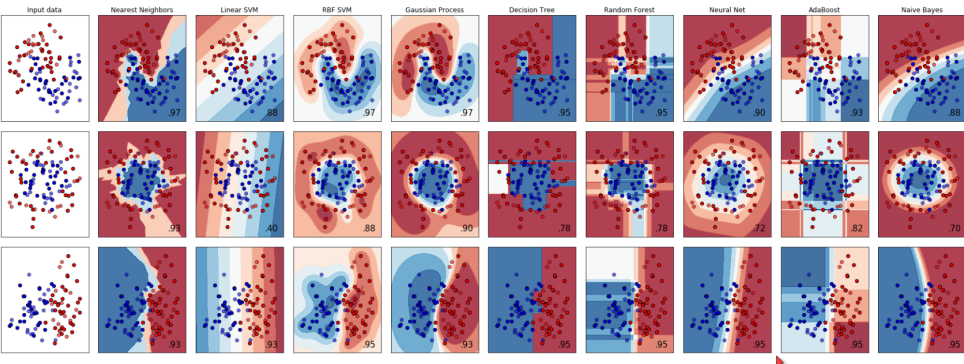
$$h : \mathbf{x} \rightarrow y$$

- ▶ if y is discrete/categorical variable, then this is a **classification** probl
- ▶ if y is real number/continuous, then this is a **regression** problem.



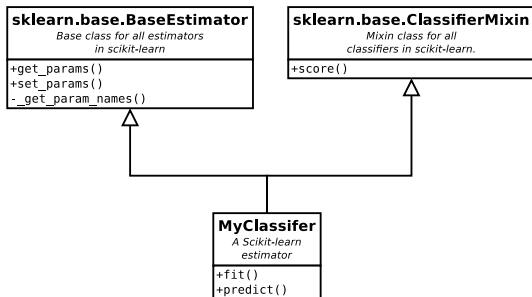
RESUMÉ: Classification

Decision Boundaries for different Models and Datasets



Source code: `L03/Extra/plot_classifier_comparison.ipynb` in [GITMAL].

RESUMÉ: The Scikit-learn Fit-Predict Interface



Python module and class function and member encapsulation:

- ▶ module private: one underscore
- ▶ class-private: two underscores

via mangled names.

...NOTE: no `virtual void fit() = 0`; declaration in python!

...for modules, private funcs can still be accessed via a hack?!

...src file: `/opt/anaconda3/pkgsrc/.../sklearn/base.py`

RESUMÉ: Exercise:

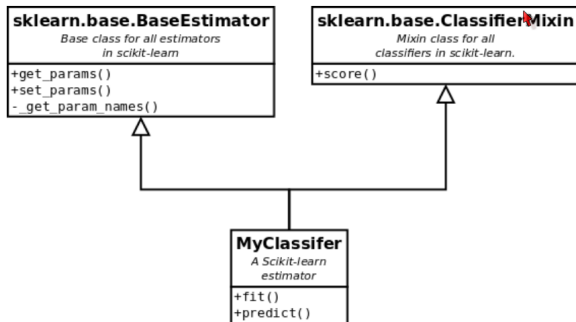
L03/dummy_classifier.ipynb

A dummy classifier for the fit-predict interface,
plus intro to a Stochastic Gradient Decent method (SGD)

Qb Implement a dummy binary classifier

Follow the code found in [HOML], p84, but name you estimator `DummyClassifier` instead of `Never5Classifier`.

Here our Python class knowledge comes into play. The estimator class hierarchy looks like



All Scikit-learn classifiers inherit from `BaseEstimator` (and possibly also `ClassifierMixin`), and they must have a `fit-predict` function pair (strangely not in the

Exercise: L03/metrics.ipynb

Nomenclature

For a binary classifier

NAME	SYMBOL	ALIAS
true positives	TP	
true negatives	TN	
false positives	FP	type I error
false negatives	FN	type II error

Exercise: L03/metrics.ipynb

Nomenclature

For a binary classifier

NAME	SYMBOL	ALIAS
true positives	TP	
true negatives	TN	
false positives	FP	type I error
false negatives	FN	type II error

and $N = N_P + N_N$ being the total number of samples and the number of positive and negative samples respectively.

Exercise: L03/metrics.ipynb

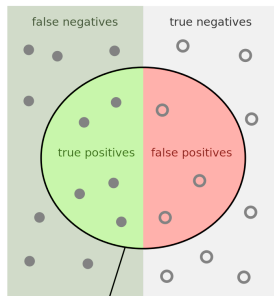
Nomenclature

For a binary classifier

NAME	SYMBOL	ALIAS
true positives	TP	
true negatives	TN	
false positives	FP	type I error
false negatives	FN	type II error

and $N = N_P + N_N$ being the total number of samples and the number of positive and negative samples respectively.

[https://en.wikipedia.org/wiki/Precision_and_recall]



Exercise: L03/metrics.ipynb

Precision, recall and accuracy, F_1 -score,
and confusion matrix

$$\text{precision,} \quad p = \frac{TP}{TP+FP}$$

$$\text{recall (or sensitivity),} \quad r = \frac{TP}{TP+FN}$$

$$\text{accuracy,} \quad a = \frac{TP+TN}{TP+TN+FP+FN}$$

$$F_1\text{-score,} \quad F_1 = \frac{2pr}{p+r}$$

Exercise: L03/metrics.ipynb

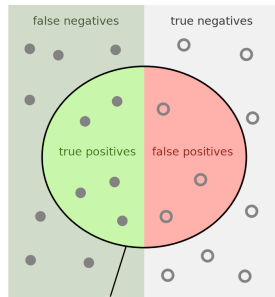
Precision, recall and accuracy, F_1 -score,
and confusion matrix

precision, $p = \frac{TP}{TP+FP}$

recall (or sensitivity), $r = \frac{TP}{TP+FN}$

accuracy, $a = \frac{TP+TN}{TP+TN+FP+FN}$

F_1 -score, $F_1 = \frac{2pr}{p+r}$



Precision = $\frac{\text{green semi-circle}}{\text{green and red semi-circle}}$

Recall = $\frac{\text{green semi-circle}}{\text{green rectangle}}$

Exercise: L03/metrics.ipynb

Precision, recall and accuracy, F_1 -score, and confusion matrix

precision, $p = \frac{TP}{TP+FP}$

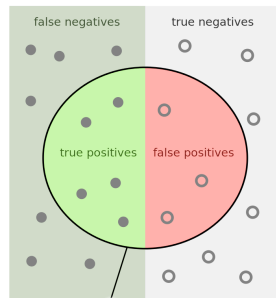
recall (or sensitivity), $r = \frac{TP}{TP+FN}$

accuracy, $a = \frac{TP+TN}{TP+TN+FP+FN}$

F_1 -score, $F_1 = \frac{2pr}{p+r}$

Confusion Matrix, $\mathbf{M}_{\text{confusion}} =$

	actual true	actual false
predicted true	TP	FP
predicted false	FN	TN



Precision = $\frac{\text{green semi-circle}}{\text{green semi-circle} + \text{red semi-circle}}$

Recall = $\frac{\text{green semi-circle}}{\text{green semi-circle} + \text{green rectangle}}$

NOTE₀: you can compare precision... F_1 -score, but not necessarily the cost, J .

NOTE₁: beware of matrix transpose and interpretation of 'TP/TN'!

Exercise: L03/metrics.ipynb

Nomenclature for the Confusion Matrix

		True condition				
		Total population	Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR-}}$	F1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$		

Exercise: L03/metrics.ipynb

Nomenclature for the Confusion Matrix

		True condition			
		Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR-}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$	

Mr. Itmal



: prevalence, positive predictive value, etc.
not important to know at all!

Exercise: L03/metrics.ipynb

Accuracy Paradox...

```
1 class ParadoxClassifier(BaseEstimator):
2     def fit(self, X, y=None):
3         pass
4     def predict(self, X):
5         assert X.ndim==2
6         return np.ones(X.shape[0],dtype=bool)
```

Exercise: L03/metrics.ipynb

Accuracy Paradox...

```
1 class ParadoxClassifier(BaseEstimator):
2     def fit(self, X, y=None):
3         pass
4     def predict(self, X):
5         assert X.ndim==2
6         return np.ones(X.shape[0],dtype=bool)
```

Test via the breast cancer Wisconsin dataset...

Exercise: L03/metrics.ipynb

Accuracy Paradox...

```
1 class ParadoxClassifier(BaseEstimator):
2     def fit(self, X, y=None):
3         pass
4     def predict(self, X):
5         assert X.ndim==2
6         return np.ones(X.shape[0],dtype=bool)
```

Test via the breast cancer Wisconsin dataset...

```
1 print('The Accuracy Paradox: a naive classifier')
2 X, y_true = load_breast_cancer(return_X_y=True)
3
4 X_train, X_test, y_train, y_test
5     = train_test_split(X, y_true, test_size=0.2, shuffle=True)
6
7 clf = ParadoxClassifier()
8 clf.fit(X_train, y_test)
9 y_pred = clf.predict(X_test)
10
11 a = accuracy_score(y_pred, y_test)
12 print(' acc=', a, ', N=', y_pred.shape[0])
```

Exercise: L03/metrics.ipynb

Accuracy Paradox...

```
1 class ParadoxClassifier(BaseEstimator):
2     def fit(self, X, y=None):
3         pass
4     def predict(self, X):
5         assert X.ndim==2
6         return np.ones(X.shape[0], dtype=bool)
```

Test via the breast cancer Wisconsin dataset...

```
1 print('The Accuracy Paradox: a naive classifier')
2 X, y_true = load_breast_cancer(return_X_y=True)
3
4 X_train, X_test, y_train, y_test
5     = train_test_split(X, y_true, test_size=0.2, shuffle=True)
6
7 clf = ParadoxClassifier()
8 clf.fit(X_train, y_test)
9 y_pred = clf.predict(X_test)
10
11 a = accuracy_score(y_pred, y_test)
12 print(' acc=', a, ', N=', y_pred.shape[0])
```

prints: acc=0.6228070175438597,
N=114

NOTE₀: for MNIST, a dum classify as '5' $\sim a = 10\%$

NOTE₁: for MNIST, a dum classify not-as '5' $\sim a = 90\%$

Training a Linear Regressor

Linear Regression: In one dimension

The well know linear equation

$$y(x) = \alpha x + \beta$$

or changing some of the symbol names, so that $h(\mathbf{x}; \mathbf{w})$ means the **predicted** value from x for a parameter set \mathbf{w} , via the hypothesis function

Training a Linear Regressor

Linear Regression: In one dimension

The well know linear equation

$$y(x) = \alpha x + \beta$$

or changing some of the symbol names, so that $h(\mathbf{x}; \mathbf{w})$ means the **predicted** value from x for a parameter set \mathbf{w} , via the hypothesis function

$$h(x; \mathbf{w}) = w_0 + w_1 x$$

Training a Linear Regressor

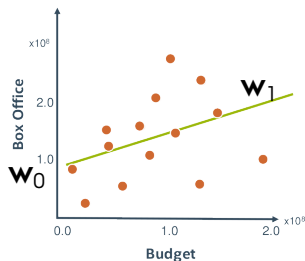
Linear Regression: In one dimension

The well know linear equation

$$y(x) = \alpha x + \beta$$

or changing some of the symbol names, so that $h(\mathbf{x}; \mathbf{w})$ means the **predicted** value from x for a parameter set \mathbf{w} , via the hypothesis function

$$h(x; \mathbf{w}) = w_0 + w_1 x$$



Training a Linear Regressor

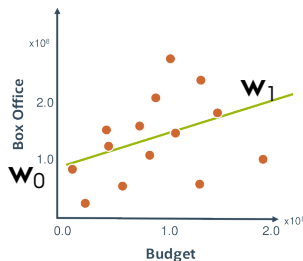
Linear Regression: In one dimension

The well know linear equation

$$y(x) = \alpha x + \beta$$

or changing some of the symbol names, so that $h(\mathbf{x}; \mathbf{w})$ means the **predicted** value from x for a parameter set \mathbf{w} , via the hypothesis function

$$h(x; \mathbf{w}) = w_0 + w_1 x$$



Question: how do we find the \mathbf{w}_n 's?

Training a Linear Regressor

Linear Regression: Hypothesis Function in N -dimensions

For 1-D:

$$h(x^{(i)}; w) = w_0 + w_1 x^{(i)}$$

Training a Linear Regressor

Linear Regression: Hypothesis Function in N -dimensions

For 1-D:

$$h(x^{(i)}; w) = w_0 + w_1 x^{(i)}$$

The same for N -D:

$$\begin{aligned} h(\mathbf{x}^{(i)}; \mathbf{w}) &= \mathbf{w}^\top \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \\ &= w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_d x_d^{(i)} \end{aligned}$$

Training a Linear Regressor

Linear Regression: Hypothesis Function in N -dimensions

For 1-D:

$$h(x^{(i)}; w) = w_0 + w_1 x^{(i)}$$

The same for N -D:

$$\begin{aligned} h(\mathbf{x}^{(i)}; \mathbf{w}) &= \mathbf{w}^\top \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \\ &= w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_d x_d^{(i)} \end{aligned}$$

and to ease notation we always prepend \mathbf{x} with a 1 as

$$\begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \mapsto \mathbf{x}^{(i)}, \quad \text{by convention in the following...}$$

Training a Linear Regressor

Linear Regression: Hypothesis Function in N -dimensions

For 1-D:

$$h(x^{(i)}; w) = w_0 + w_1 x^{(i)}$$

The same for N -D:

$$\begin{aligned} h(\mathbf{x}^{(i)}; \mathbf{w}) &= \mathbf{w}^\top \begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \\ &= w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_d x_d^{(i)} \end{aligned}$$

and to ease notation we always prepend \mathbf{x} with a 1 as

$$\begin{bmatrix} 1 \\ \mathbf{x}^{(i)} \end{bmatrix} \mapsto \mathbf{x}^{(i)}, \quad \text{by convention in the following...}$$

yielding the vector form of the hypothesis function

$$h(\mathbf{x}^{(i)}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}^{(i)}$$

Training a Linear Regressor

Linear Regression: Loss or Objective Function

Individual loss, via a square difference

$$\begin{aligned} L^{(i)} &= (h(\mathbf{x}^{(i)}; \mathbf{w}) - y^{(i)})^2 \\ &= (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \end{aligned}$$

Training a Linear Regressor

Linear Regression: Loss or Objective Function

Individual loss, via a square difference

$$\begin{aligned} L^{(i)} &= (h(\mathbf{x}^{(i)}; \mathbf{w}) - y^{(i)})^2 \\ &= (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \end{aligned}$$

and to minimize all the $L^{(i)}$ losses (or indirectly also the MSE or RMSE) is to minimize the sum of all the individual costs, via the total cost function J

$$\begin{aligned} \text{MSE}(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \frac{1}{n} \sum_{i=1}^n L^{(i)} \\ &= \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \end{aligned}$$

Training a Linear Regressor

Linear Regression: Loss or Objective Function

Individual loss, via a square difference

$$\begin{aligned}L^{(i)} &= (h(\mathbf{x}^{(i)}; \mathbf{w}) - y^{(i)})^2 \\ &= (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2\end{aligned}$$

and to minimize all the $L^{(i)}$ losses (or indirectly also the MSE or RMSE) is to minimize the sum of all the individual costs, via the total cost function J

$$\begin{aligned}\text{MSE}(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \frac{1}{n} \sum_{i=1}^n L^{(i)} \\ &= \frac{1}{n} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\end{aligned}$$

Ignoring constant factors, this yields our linear regression cost function

$$\begin{aligned}J &= \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \\ &\propto \text{MSE}\end{aligned}$$

Training a Linear Regressor

Minimizing the Linear Regression: The `argmin` concept

Our linear regression cost function was

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Training a Linear Regressor

Minimizing the Linear Regression: The `argmin` concept

Our linear regression cost function was

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

and training amounts to finding a value of \mathbf{w} , that minimizes J . This is denoted as

$$\begin{aligned}\mathbf{w}^* &= \operatorname{argmin}_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) \\ &= \operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\end{aligned}$$

Training a Linear Regressor

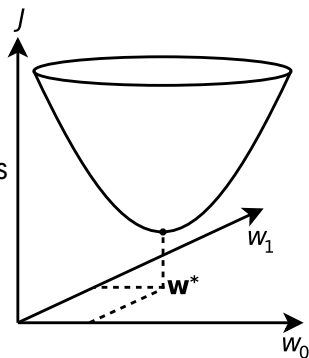
Minimizing the Linear Regression: The `argmin` concept

Our linear regression cost function was

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

and training amounts to finding a value of \mathbf{w} , that minimizes J . This is denoted as

$$\begin{aligned}\mathbf{w}^* &= \operatorname{argmin}_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) \\ &= \operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\end{aligned}$$



Training a Linear Regressor

Minimizing the Linear Regression: The `argmin` concept

Our linear regression cost function was

$$J(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

and training amounts to finding a value of \mathbf{w} , that minimizes J . This is denoted as

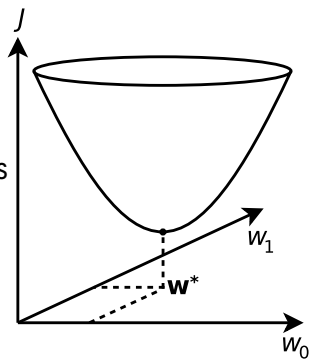
$$\begin{aligned}\mathbf{w}^* &= \operatorname{argmin}_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) \\ &= \operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2\end{aligned}$$

and by minima, we naturally hope for

- ▶ the global minimum

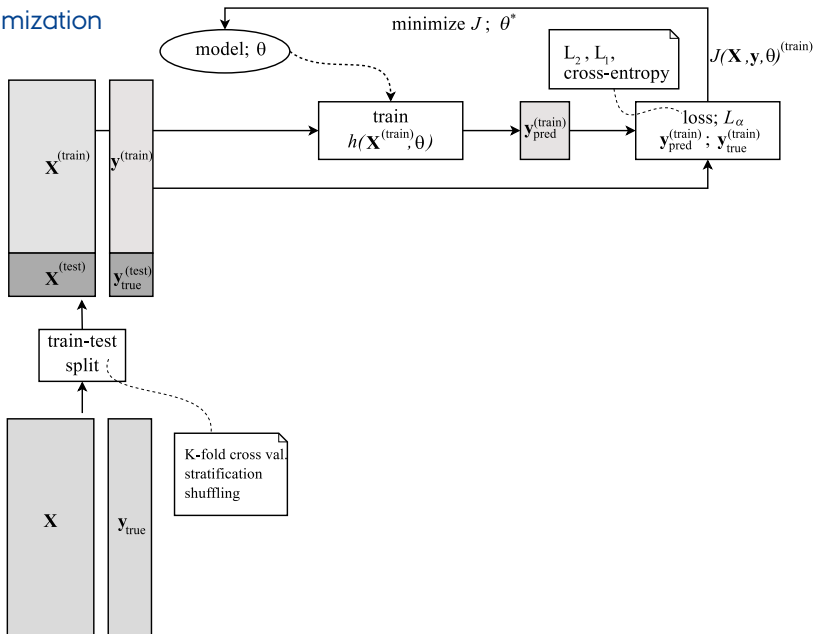
thought for non-linear models this cannot be guaranteed, hitting some

- ▶ local minimum



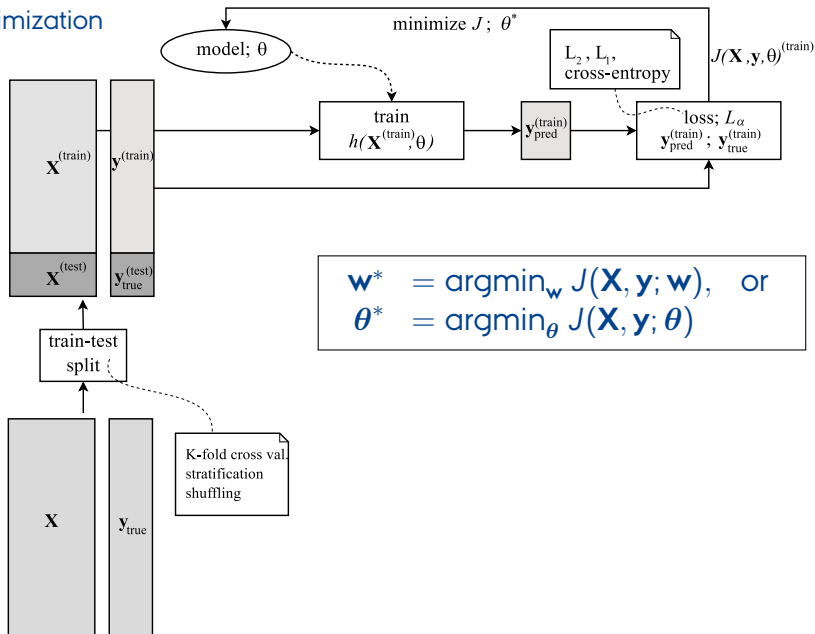
Training in General

Minimization



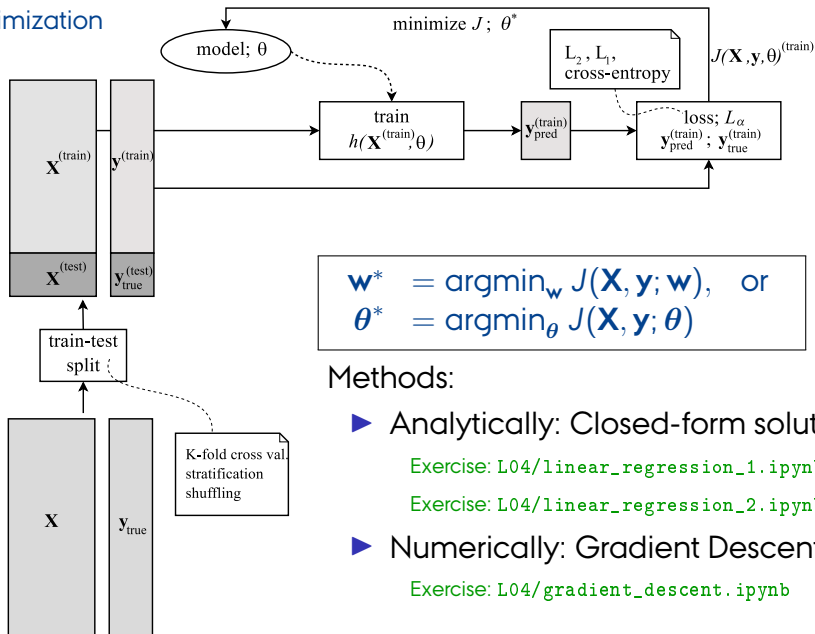
Training in General

Minimization



Training in General

Minimization



Methods:

- Analytically: Closed-form solution

Exercise: L04/linear_regression_1.ipynb

Exercise: L04/linear_regression_2.ipynb

- Numerically: Gradient Descent

Exercise: L04/gradient_descent.ipynb

Exercise: L04/linear_regression_1.ipynb

Training: The Closed-form Linear-Least-Squares Solution

To solve for \mathbf{w}^* in closed form, we find the gradient of J with respect to \mathbf{w}

$$\nabla_{\mathbf{w}} J = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \dots, \frac{\partial J}{\partial w_m} \right]^\top$$

Exercise: L04/linear_regression_1.ipynb

Training: The Closed-form Linear-Least-Squares Solution

To solve for \mathbf{w}^* in closed form, we find the gradient of J with respect to \mathbf{w}

$$\nabla_{\mathbf{w}} J = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \dots, \frac{\partial J}{\partial w_m} \right]^\top$$

Taking the partial derivery $\partial/\partial_{\mathbf{w}}$ of the J via the gradient (nabla) operator

$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \\ 0 &= \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y} \end{aligned}$$

Exercise: L04/linear_regression_1.ipynb

Training: The Closed-form Linear-Least-Squares Solution

To solve for \mathbf{w}^* in closed form, we find the gradient of J with respect to \mathbf{w}

$$\nabla_{\mathbf{w}} J = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \dots, \frac{\partial J}{\partial w_m} \right]^\top$$

Taking the partial derivery $\partial/\partial_{\mathbf{w}}$ of the J via the gradient (nabla) operator

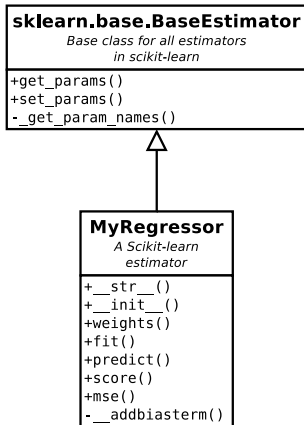
$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{X}, \mathbf{y}; \mathbf{w}) &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \\ 0 &= \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y} \end{aligned}$$

with a small amount of matrix algebra, this gives the *normal equation*

$$\begin{aligned} \mathbf{w}^* &= \operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \\ &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \end{aligned}$$

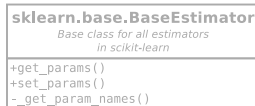
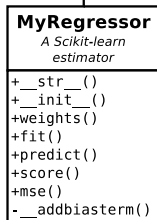
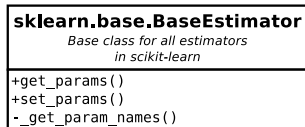
Exercise: L04/linear_regression_2.ipynb

Python class: `MyRegressor`



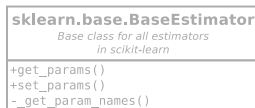
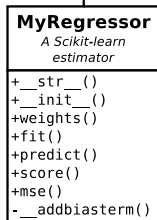
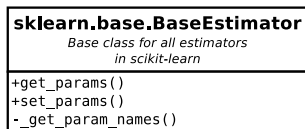
Exercise: L04/linear_regression_2.ipynb

Python class: `MyRegressor`



Exercise: L04/linear_regression_2.ipynb

Python class: `MyRegressor`



Exercise: create a linear regressor, inheriting from `BaseEstimator` and implement `score()` and `mse()`.

NOTE: no inhering from `ClassifierMixin`.