

Lecture 10: Preprocessing, more non-linear methods and AI

Preprocessing, support vector machines, decision trees, ensembles, machine learning and AI

Nils Olovsson

nils.olofsson@igp.uu.se

Introduction to data science and Python programming for medical research
Uppsala University

2025



UPPSALA
UNIVERSITET

Content

Introduction

Preprocessing

Support vector machines

Decision trees

Ensembles

Random forests

Conclusions

Computers and AI

Introduction

Introduction

Introduction: Problem

Real world data can be **nonlinear**.

Features can also have very **different ranges!**

Values can be **missing**.

- ▶ Preprocessing
- ▶ Nonlinear methods

Preprocessing

Preprocessing

Preprocessing: Encoding of categorical features

Features can have values according to discrete **categories**.

E.g. patient **age** categorized into **age groups**.

Categorical values need to be **encoded into numerical** values before we can include them in data analyses.

Ordinal encoding

Convert each category to a value.

The order of the encoded values have to make sense!

One-hot encoding

Convert each category to a new feature that can take the value 0 or 1. (Commonly used *e.g.* for multi class classification!)

Variables Table

Variable Name	Role	Type	Demographic
age	Feature	Integer	Age
sex	Feature	Categorical	Sex
cp	Feature	Categorical	
trestbps	Feature	Integer	
chol	Feature	Integer	
fbs	Feature	Categorical	
restecg	Feature	Categorical	
thalach	Feature	Integer	
exang	Feature	Categorical	
oldpeak	Feature	Integer	

Preprocessing: Missing values

We have previously seen how pandas can be used to **impute missing values** in our data with the `df.fillna(...)` method.

sklearn also contains methods for **data imputation** with more **advanced features** such as e.g. multi-variate estimation of missing values.

Using a more advanced method to impute missing values should be done with caution!
You should be able to understand and explain what you do to your data.

sklearn.impute

Transformers for missing value imputation.

User guide. See the [Imputation of missing values](#) section for further details.

IterativeImputer	Multivariate imputer that estimates each feature from all the others.
KNNImputer	Imputation for completing missing values using k-Nearest Neighbors.
MissingIndicator	Binary indicators for missing values.
SimpleImputer	Univariate imputer for completing missing values with simple strategies.

```

1 >> ## Original dataframe
2 >>      Group 1   Group 2
3 >> 0         NaN   2.428080
4 >> 1   0.368762   0.148749
5 >> 2   7.319432   8.214800
6 >> 3   5.459722   6.565469
7 >> ## Dataframe with imputed values
8 >>      Group 1   Group 2
9 >> 0   0.000000   2.428080
10 >> 1   0.368762   0.148749
11 >> 2   7.319432   8.214800
12 >> 3   5.459722   6.565469

```

Simple replacement of NaN with zero.

Preprocessing: Scaling and normalization

Features can represent **different things** so values can be of **different ranges**.

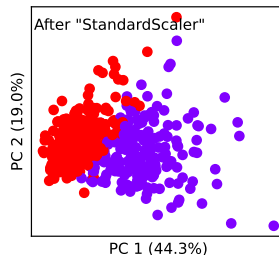
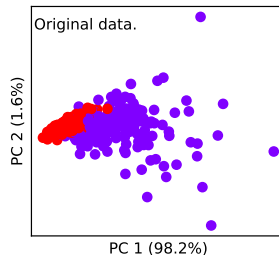
E.g. body temperature may vary between 35 and 42 degrees while blood concentration of a substance may range from 0 to $\frac{1}{1000}$.

Many methods are **not scale invariant**.

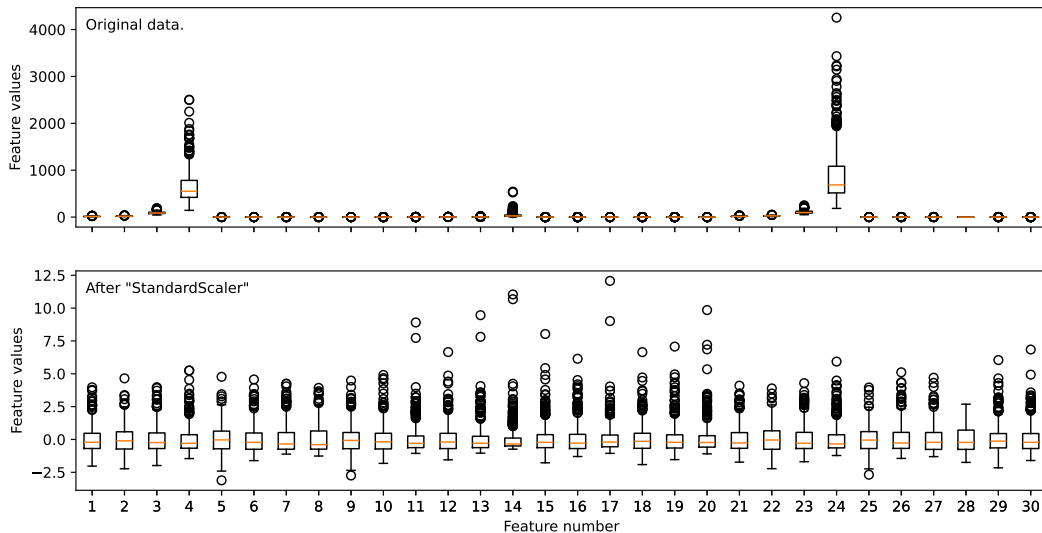
Data can be normalized by e.g.

1. Center the mean on 0.
2. Scale such that variance is 1.

Can **improve performance** of many estimators.



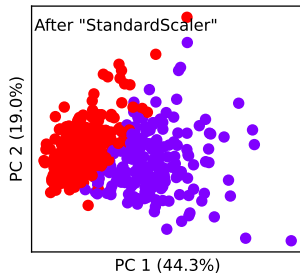
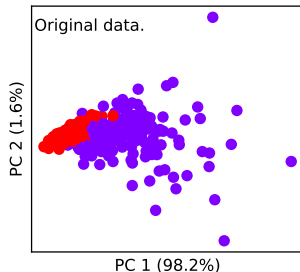
Preprocessing: Scaling breast cancer dataset



Preprocessing: Scaling effect on PCA of breast cancer dataset

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.decomposition import PCA
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.datasets import load_breast_cancer
6 # Load and scale data
7 X, y = load_breast_cancer(return_X_y=True)
8 X_tr = StandardScaler().fit_transform(X)
9 X_list = [X, X_tr]
10 labels = []
11 labels.append('Original data.')
12 labels.append('After "StandardScaler"')
13 # Plot PCA of original and scaled data
14 fig, axs = plt.subplots(2, 1, figsize=(3, 6))
15 cmap = plt.get_cmap('rainbow', 2)
16 for i in range(2):
17     pca = PCA(n_components=2)
18     X_pca = pca.fit_transform(X_list[i])
19     ax = axs[i]
20     ax.scatter(X_pca[:,0], X_pca[:,1],
21               c=y, cmap=cmap)
22     ax.text(0.01, 0.95, labels[i],
23            va='top', ha='left',
24            transform=ax.transAxes)
25     v = 100.0 * pca.explained_variance_ratio_
26     ax.set_xlabel(f'PC 1 ({v[0]:.1f}%)')
27     ax.set_ylabel(f'PC 2 ({v[1]:.1f}%)')
    
```



Preprocessing: Logarithmic transform

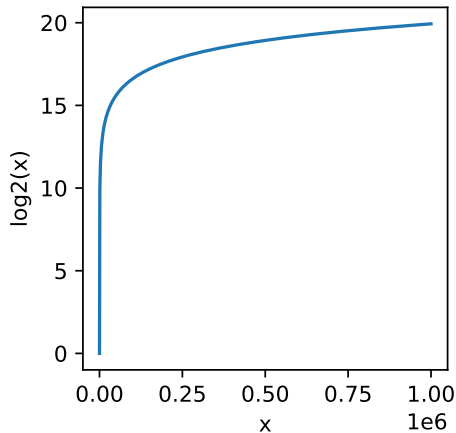
Single cell data typically **transformed** by taking a **logarithm**

$$X = \log_2 (X + 1)$$

This transform **is reasonable** because:

1. All values in this type of data is ≥ 0 .
2. The +1 makes sure that 0 in original data is 0 in transformed data.

The **logarithm compresses** the **range** of the data just as the StandardScaler.



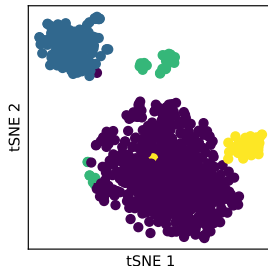
Preprocessing: Custom transforms in sklearn

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.base import BaseEstimator
4 from sklearn.base import TransformerMixin
5 from sklearn.manifold import TSNE
6
7 class LogTransform(BaseEstimator, TransformerMixin):
8     def __init__(self):
9         return None
10    def fit(self, x, y=None):
11        return self
12    def transform(self, X):
13        return self
14    def fit_transform(self, X):
15        X_tr = np.log(X + 1)
16        return X_tr
17
18 # Load data, transform and apply TSNE
19 X, y = common.load_CAF()
20 X = LogTransform().fit_transform(X)
21 tsne = TSNE(n_components=2, random_state=0)
22 X_tsne = tsne.fit_transform(X)
23
24 # Plot result
25 ax = plt.figure(figsize=(3,3)).gca()
26 ax.scatter(X_tsne[:,0], X_tsne[:,1],
27            marker='o', c=y.astype(int))
28 ax.set_xlabel('tSNE 1')
29 ax.set_ylabel('tSNE 2')
30 plt.show()
  
```

Can create our own sklearn classes using inheritance.

By doing this we can e.g. include our custom transform in a sklearn pipelines.



Preprocessing: Feature engineering

Derive new features from the old ones.

E.g., dataset with two features

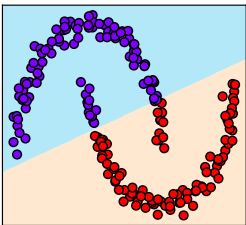
$$x_1, x_2$$

Expanding these using **polynomial features** of degree 2 gives six features

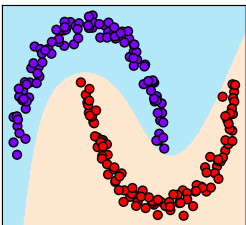
$$1, x_1, x_2, x_1^2, x_2^2, x_1x_2$$

This type of feature transform can enable the fitting of **linear estimators** to **non-linear data**.

(Beware of overfitting and combinatorial explosion!)



Original data.

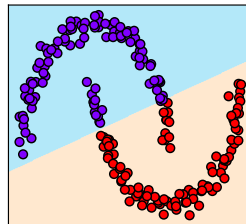


With "PolynomialFeatures"
(Degree 5)

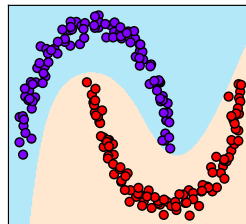
Preprocessing: Polynomial features in sklearn

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.preprocessing import PolynomialFeatures
5 from sklearn.datasets import make_moons
6 X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
7 degree = 5
8 poly = PolynomialFeatures(degree=degree)
9 X_poly = poly.fit_transform(X)
10 X_list = [X, X_poly]
11 p_list = [None, poly]
12 labels = []
13 labels.append('Original data.')
14 labels.append(f'With "PolynomialFeatures"\n(Degree {degree})')
15 # Visualize decision boundary of the two classifiers
16 fig, axs = plt.subplots(2, 1, figsize=(3, 6))
17 cmap = plt.get_cmap('rainbow', 2)
18 for i in range(2):
19     lg = LogisticRegression(random_state=0)
20     lg.fit(X_list[i], y)
21     y_pred = lg.predict(X_list[i])
22     ax = axs[i]
23     ax.scatter(X[:,0], X[:,1],
24               edgecolor='k',
25               c=y_pred, cmap=cmap,
26               zorder=3)
27     common.plot_classifier(lg, X_list[i], y,
28                           ax, preprocess=p_list[i])
29     ax.set_xlabel(labels[i])
    
```



Original data.



With "PolynomialFeatures"
(Degree 5)

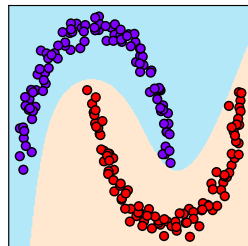
Preprocessing: Pipelines

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.preprocessing import PolynomialFeatures
5 from sklearn.pipeline import make_pipeline
6 from sklearn.datasets import make_moons
7 X, y = make_moons(n_samples=200,
8                   noise=0.05,
9                   random_state=0)
10 # Create a pipeline that first creates polynomial features,
11 # then fits a logistic regression to the transformed data
12 degree = 5
13 poly_log_res = make_pipeline(PolynomialFeatures(degree=degree),
14                             LogisticRegression(random_state=0))
15 label = f'Pipelined "PolynomialFeatures"\n(Degree {degree}) and\n"
16         LogisticRegression"'
17 # Plot classification result and decision boundary
18 ax = plt.figure(figsize=(3,3)).gca()
19 cmap = plt.get_cmap('rainbow', 2)
20 poly_log_res.fit(X, y)
21 y_pred = poly_log_res.predict(X)
22 ax.scatter(X[:,0], X[:,1],
23           edgecolor='k',
24           c=y_pred, cmap=cmap,
25           zorder=3)
26 common.plot_classifier(poly_log_res, X, y, ax)
27 ax.set_xlabel(label)
    
```

<https://scikit-learn.org/stable/api/sklearn.pipeline.html>

Preprocessing steps, transforms and estimators can be pipelined into a single object.



Pipelined "PolynomialFeatures"
(Degree 5) and
"LogisticRegression"

Support vector machines

Support vector machines

Support vector machines: Flexible regression and classification

Supervised machine learning methods
for **regression** and **classification**.

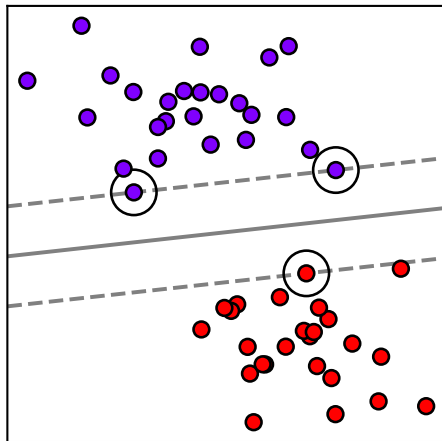
Support vector regression (SVR)

Used for **regression**.

Support vector classifier (SVC)

Used for **classification**.

Finds **line or curve** that **divides classes**.



<https://scikit-learn.org/stable/modules/svm.html>

<https://scikit-learn.org/stable/api/sklearn.svm.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>

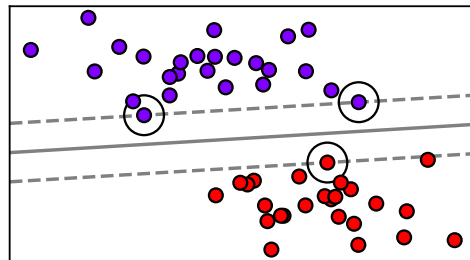
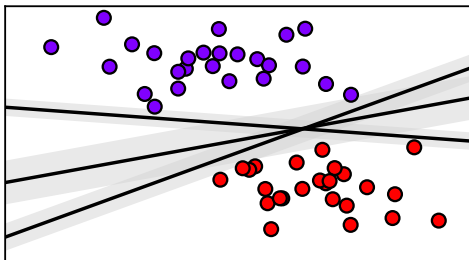
<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

Support vector machines: Linear SVC maximizes margin

Many lines can be draw to **separate two classes** of data.

SVC finds line that **separates classes** and has the **widest margin** to the closest data points on each side.

These data points, encircled below on the right, are called the **support vectors**.

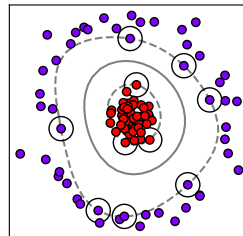
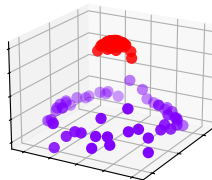
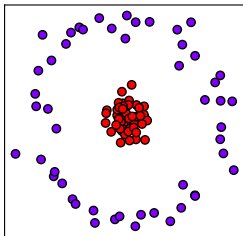


Support vector machines: Kernel SVC for non-linear data

Kernel SVC is motivated by same idea as **feature engineering**.
However, **inherent** to the method itself and therefore **more efficient**.

Kernels

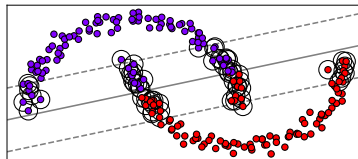
- Polynomial
- Radial basis function (RBF)



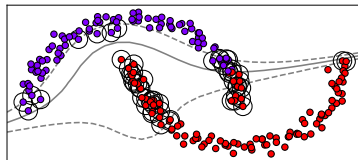
Support vector machines: Kernels (Example)

```

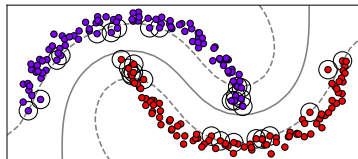
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.svm import SVC
4 from sklearn.datasets import make_moons
5
6 X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
7 kernels = ['linear', 'poly', 'rbf']
8
9 # Visualize decision boundary of the three classifiers
10 fig, axs = plt.subplots(3, 1, figsize=(6, 9))
11 cmap = plt.get_cmap('rainbow', 2)
12 for i in range(3):
13     ax = axs[i]
14     svc = SVC(kernel=kernels[i], C=1)
15     svc.fit(X, y)
16     y_pred = svc.predict(X)
17     ax.scatter(X[:,0], X[:,1],
18               edgecolor='k',
19               c=y_pred, cmap=cmap,
20               zorder=3)
21     common.plot_svc_decision_function(svc, ax)
22     ax.set_xlabel(f'kernel: {kernels[i]}')
    
```



kernel: linear



kernel: poly



kernel: rbf

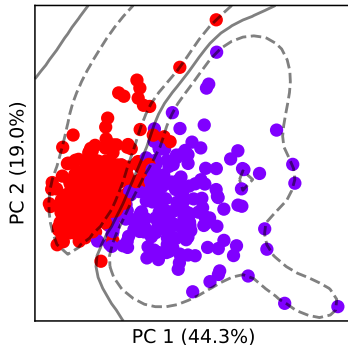
Radial basis function SVC is a very powerful non-linear classifier! Beware of overfitting!

Support vector machines: PCA transformed breast cancer data

```

1 import matplotlib.pyplot as plt
2 from sklearn.decomposition import PCA
3 from sklearn.svm import SVC
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.datasets import load_breast_cancer
6 # Load and scale data
7 X, y = load_breast_cancer(return_X_y=True)
8 X = StandardScaler().fit_transform(X)
9
10 # Apply PCA for dimensionality reduction
11 pca = PCA(n_components=2)
12 X_pca = pca.fit_transform(X)
13 # Create a SVC classifier
14 svc = SVC(kernel='rbf', C=10)
15 svc.fit(X_pca, y)
16
17 # Plot PCA of original and scaled data
18 ax = plt.figure(figsize=(3, 3)).gca()
19 cmap = plt.get_cmap('rainbow', 2)
20 ax.scatter(X_pca[:,0], X_pca[:,1],
21           c=y, cmap=cmap)
22 common.plot_svc_decision_function(svc, ax, False)
23 # Indicate PCA variance on axes
24 v = 100.0 * pca.explained_variance_ratio_
25 ax.set_xlabel(f'PC 1 ({v[0]:.1f}%)')
26 ax.set_ylabel(f'PC 2 ({v[1]:.1f}%)')
27 plt.show()

```



(Normally you would not fit your classifier to dimensionality reduced data used for visualization!)

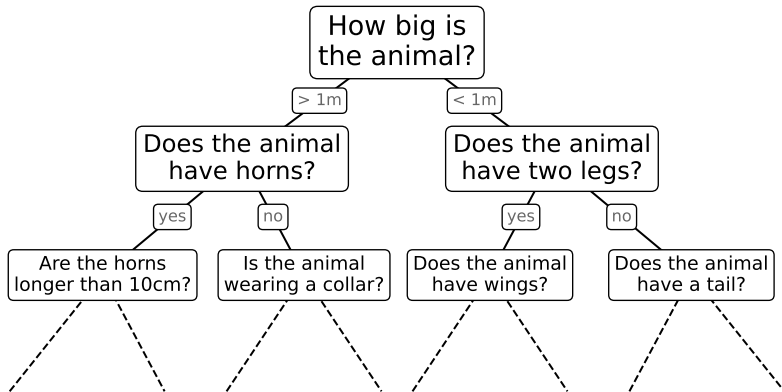
Decision trees

Decision trees

Decision trees: Successive binary splits

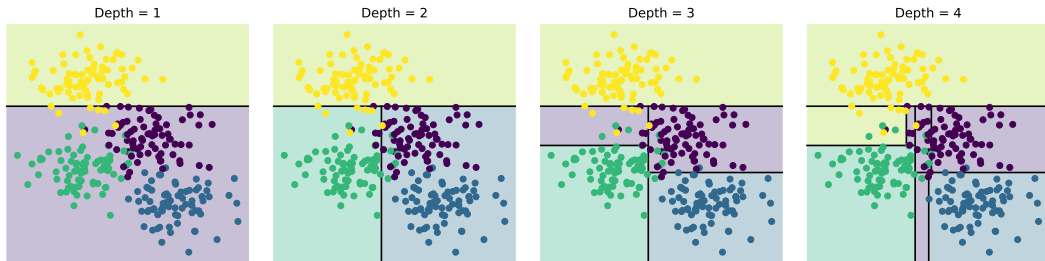
Classification or **regression** of data is performed by **splitting** data according to **cutoff values**.

Decision Tree: Animal Classification



Decision trees: Classification with different depths

Dimensionality reduction with PCA is achieved by **projecting** data points on the first PC vectors.



<https://scikit-learn.org/stable/modules/tree.html>

<https://scikit-learn.org/stable/api/sklearn.tree.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>

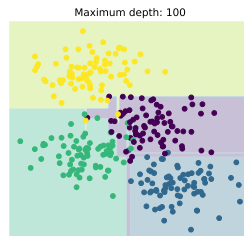
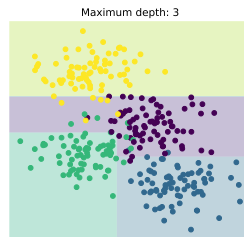
Decision trees: Overfitting

```

1 import matplotlib.pyplot as plt
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.datasets import make_blobs
4
5 fig, axs = plt.subplots(2, 1, figsize=(5, 10))
6
7 X, y = make_blobs(n_samples=300, centers=4,
8                   random_state=0, cluster_std=1.0)
9
10 tree0 = DecisionTreeClassifier(max_depth=3)
11 tree1 = DecisionTreeClassifier(max_depth=100)
12
13 common.visualize_tree(axs[0], tree0, X, y, boundaries=False)
14 common.visualize_tree(axs[1], tree1, X, y, boundaries=False)
15
16 axs[0].set_title('Maximum depth: 3')
17 axs[1].set_title('Maximum depth: 100')
18 plt.show()
  
```

Decision tree classifiers are very powerful non-linear classifiers! Beware of overfitting!

Overfitting is seen in this classifier as the thin separating regions, basically being adapted to a single data point.



Ensembles

Ensembles

Ensembles: Multiple instances

By using **several instances** of an **estimator results** can be **improved**.

Especially if the estimator is prone to overfitting.

- ▶ Bagging (Bootstrap aggregating)
- ▶ Adaptive boosting
- ▶ Gradient boosting

<https://scikit-learn.org/stable/modules/ensemble.html>
<https://scikit-learn.org/stable/api/sklearn.ensemble.html>
https://en.wikipedia.org/wiki/Bootstrap_aggregating
<https://en.wikipedia.org/wiki/AdaBoost>
https://en.wikipedia.org/wiki/Gradient_boosting

sklearn.ensemble

Ensemble-based methods for classification, regression and anomaly detection.

User guide. See the [Ensembles: Gradient boosting, random forests, bagging, voting, stacking](#) section for further details.

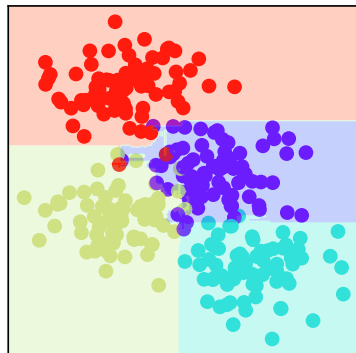
AdaBoostClassifier	An AdaBoost classifier.
AdaBoostRegressor	An AdaBoost regressor.
BaggingClassifier	A Bagging classifier.
BaggingRegressor	A Bagging regressor.
ExtraTreesClassifier	An extra-trees classifier.
ExtraTreesRegressor	An extra-trees regressor.
GradientBoostingClassifier	Gradient Boosting for classification.
GradientBoostingRegressor	Gradient Boosting for regression.
HistGradientBoostingClassifier	Histogram-based Gradient Boosting Classification Tree.
HistGradientBoostingRegressor	Histogram-based Gradient Boosting Regression Tree.
IsolationForest	Isolation Forest Algorithm.
RandomForestClassifier	A random forest classifier.
RandomForestRegressor	A random forest regressor.
RandomTreesEmbedding	An ensemble of totally random trees.
StackingClassifier	Stack of estimators with a final classifier.
StackingRegressor	Stack of estimators with a final regressor.
VotingClassifier	Soft Voting/Majority Rule classifier for unfitted estimators.
VotingRegressor	Prediction voting regressor for unfitted estimators.

Ensembles: Ensemble of trees

```

1 import matplotlib.pyplot as plt
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.ensemble import BaggingClassifier
4 from sklearn.datasets import make_blobs
5
6 # Generate blob data
7 X, y = make_blobs(n_samples=300, centers=4,
8                   random_state=0, cluster_std=1.0)
9
10 # Create a bagging using trees and fit to data
11 tree = DecisionTreeClassifier()
12 bag = BaggingClassifier(tree,
13                         n_estimators=100,
14                         max_samples=0.8,
15                         random_state=0)
16 bag.fit(X, y)
17
18 # Plot the data and the classifier regions
19 ax = plt.figure(figsize=(3, 3)).gca()
20 cmap = plt.get_cmap('rainbow', 4)
21 ax.scatter(X[:,0], X[:,1],
22           c=y, cmap=cmap)
23 common.plot_classifier(bag, X, y, ax)
24 plt.show()

```



Ensembles of decision tree classifiers
are very powerful non-linear classifiers!
Still need to beware of overfitting!

Random forests

Random forests

Random forests: Optimized ensembles of trees

Random forests combines **decision trees** into an **inherent ensemble** method.

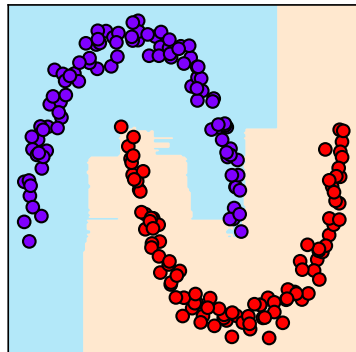
Less prone to overfitting than a single decision tree.

(In practice these should often be used instead of decision trees.)



Random forests: Toy data (Example)

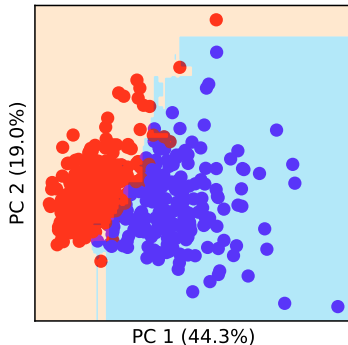
```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.datasets import make_moons
5
6 # Generate data
7 X, y = make_moons(n_samples=200, noise=0.05,
8                   random_state=0)
9
10 # Create and fit a random forest classifier
11 forest = RandomForestClassifier()
12 forest.fit(X, y)
13 y_pred = forest.predict(X)
14
15 # Visualize decision boundary of the random forest
16 ax = plt.figure(figsize=(3, 3)).gca()
17 cmap = plt.get_cmap('rainbow', 2)
18 ax.scatter(X[:,0], X[:,1],
19           edgecolor='k',
20           c=y_pred, cmap=cmap,
21           zorder=3)
22 common.plot_classifier(forest, X, y, ax)
```



Random forests: PCA transformed breast cancer data

```

1 import matplotlib.pyplot as plt
2 from sklearn.decomposition import PCA
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.datasets import load_breast_cancer
6 # Load and scale data
7 X, y = load_breast_cancer(return_X_y=True)
8 X = StandardScaler().fit_transform(X)
9
10 # Apply PCA for dimensionality reduction
11 pca = PCA(n_components=2)
12 X_pca = pca.fit_transform(X)
13 # Create a SVC classifier
14 forest = RandomForestClassifier()
15 forest.fit(X_pca, y)
16
17 # Plot PCA of original and scaled data
18 ax = plt.figure(figsize=(3, 3)).gca()
19 cmap = plt.get_cmap('rainbow', 2)
20 ax.scatter(X_pca[:,0], X_pca[:,1],
21           c=y, cmap=cmap)
22 common.plot_classifier(forest, X, y, ax)
23 # Indicate PCA variance on axes
24 v = 100.0 * pca.explained_variance_ratio_
25 ax.set_xlabel(f'PC 1 ({v[0]:.1f}%)')
26 ax.set_ylabel(f'PC 2 ({v[1]:.1f}%)')
27 plt.show()
  
```



(Normally you would not fit your classifier to dimensionality reduced data used for visualization!)

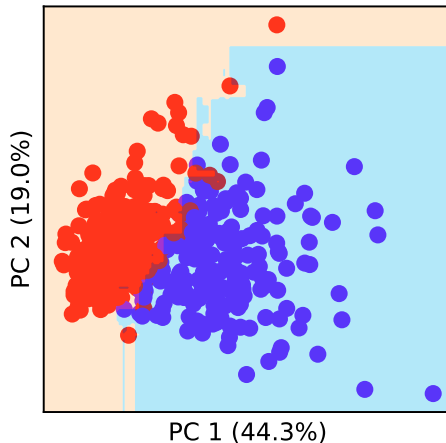
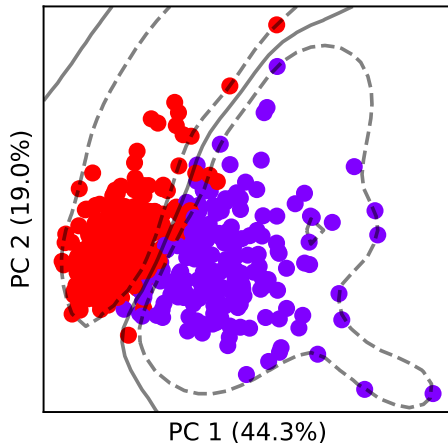
Conclusions

Conclusions

Conclusions: Support vector machines and random forests

Support vector machines and **random forests** are very different methods but both provide very **powerful techniques** for **non-linear regression** and **classification**.

(Note: All examples here have been on classification.)



Conclusions: Preprocessing and non-linear models

Preprocessing

Categorical features need to be encoded and missing values handled.

Scale, normalize or transform data.

E.g. log-transform single cell data.

Sklearn preprocessing and estimation can be chained into a single object using a pipeline.

Support vector machines

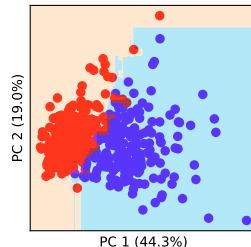
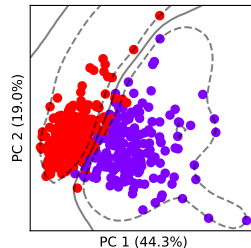
Powerful method for non-linear regression and classification.

Beware overfitting.

Random forests

Powerful method for non-linear regression and classification.

Beware overfitting.



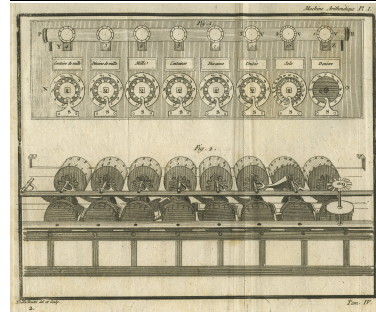
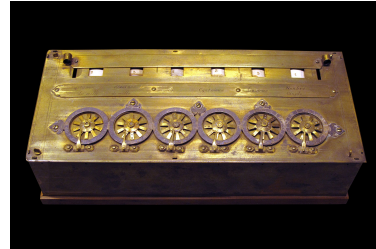
Computers and AI

Computers and AI

Computers and AI: Historical computation



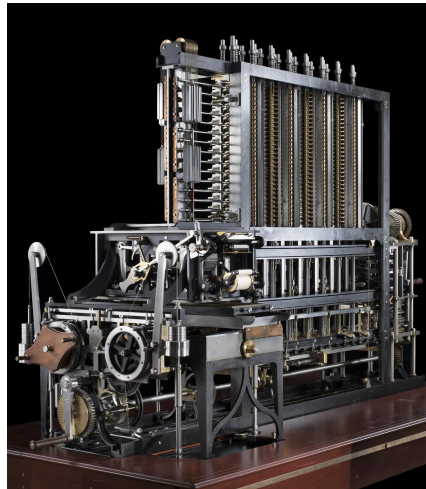
https://en.wikipedia.org/wiki/Antikythera_mechanism
<https://en.wikipedia.org/wiki/Pascaline>



Computers and AI: Charles Babbage's difference engine (1800s)



https://en.wikipedia.org/wiki/Charles_Babbage
<https://collection.sciencemuseumgroup.org.uk/objects/co62748>



Ada's mother called it a "thinking machine", and reported that it "raised several Nos. to the 2nd & 3rd powers, and extracted the root of a Quadratic Equation". It would change the course of Ada's life.

Computers and AI: Ada Lovelace and the analytical engine (1800s)



352

MACHINE ANALYTIQUE

matière que les ondes à la surface des eaux tranquilles et que, dans la formation de ces ondulations, les eaux sont ébranlées sur profondeurs les plus considérables.

Les observations des marées offrent un vif intérêt scientifique; la géologie tirera un grand parti de la connaissance exacte des oscillations de l'Océan; car, avec leur mouvement de transmission plus ou moins rapide, on pourra déterminer avec assez d'exactitude, la profondeur de la mer bien loin du littoral.

NOTIONS SUR LA MACHINE ANALYTIQUE DE M. CHARLES BABBAGE, par Mr. L.-F. MARRASSE, capitaine du génie militaire.

Les travaux qui appartiennent à plusieurs branches des sciences mathématiques, quoique paraissant, au premier abord, être uniquement du ressort de l'esprit, peuvent néanmoins se diviser en deux parties distinctes: l'une qu'on peut appeler mécanique, parce qu'elle est soumise à des lois précises et invariables, susceptibles d'être ordonnées physiquement, tandis que l'autre qui exige l'intervention du raisonnement, est plus spécialement du domaine de la pensée. Dès lors on pourra se proposer de faire exécuter par le moyen de machines la partie mécanique du travail, et réserver à la seule intelligence celle qui dépend de la liberté de raisonnement. Ainsi la rigueur à laquelle sont soumises les règles de calcul numérique a dû, depuis longtemps, faire songer à employer des instruments matériels, soit pour exécuter entièrement ces calculs, soit pour les abréger. De là sont nés plusieurs inventions dirigées vers ce but, mais qui ne l'atteignent, en général, qu'imparfaitement. Ainsi la machine de Pascal, tant vantée, n'est maintenant qu'un simple objet de curiosité qui, tout en prouvant une

353

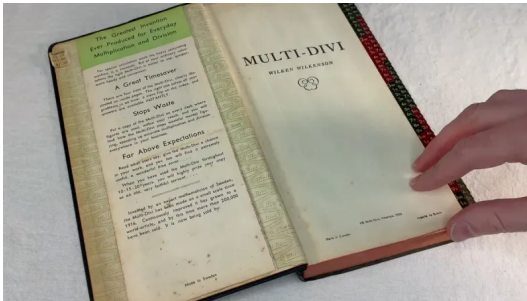
DE M. CHARLES BABBAGE.

NOMBRE des OPERATIONS.	CAUTIONS des MANÈGES ou Signes indiquant le mode des opérations.	CARPENS DES VARIABLE.		MARCHE des OPERATIONS.
		à l'usage des opérations.	Colonne marquée la moitié des opérations.	
1	×	$P_1 \times P_1 = P_1$	$P_1 \dots\dots\dots da'$	
2	×	$P_1 \times P_1 = P_1$	$P_1 \dots\dots\dots d'a'$	
3	×	$P_1 \times P_1 = P_1$	$P_1 \dots\dots\dots d'a'$	
4	×	$P_1 \times P_1 = P_1$	$P_1 \dots\dots\dots d'a'$	
5	×	$P_1 \times P_1 = P_1$	$P_1 \dots\dots\dots d'a'$	
6	×	$P_1 \times P_1 = P_1$	$P_1 \dots\dots\dots d'a'$	
7	×	$P_1 \times P_1 = P_1$	$P_1 \dots\dots\dots d'a'$	

Comme les carions ne font qu'indiquer comment et sur quelles colonnes la machine doit agir, il est clair qu'il faudra encore, dans chaque cas particulier, introduire les données numériques du calcul. Ainsi, dans l'exemple que nous venons choisi, on devra préalablement écrire les valeurs numériques de m, n, d, m', n', d' dans l'ordre et sur les colonnes indiquées, après quoi l'on fera agir la machine qui donnera la valeur de l'incconnue x pour ce cas particulier. Pour avoir la valeur de y , il faudra faire une autre série d'opérations analogues aux précédentes. Mûs on voit qu'elles se réduiront à quatre seulement, car le dénominateur de l'expression de y , seul le s'ige, est le même que celui de x , et égal à $m'n - m'n'$. Dans le tableau précédent l'on remarquera que la colonne des opérations indique de suite quatre multiplications, deux soustractions et une division. Ainsi l'en pourra, au besoin, n'employer que trois carions des opérations; pour cela il suffira d'introduire dans la machine un appareil qui, par exemple, après la première multiplication, retirene le carion relatif à cette opération, et ne lui permette d'avancer, pour être remplacé par

Somerville's son): "It does not appear to me that cerebral matter need be more unmanageable to mathematicians than *sidereal & planetary* matter & movements; if they would but inspect it from the *right point of view*. I hope to bequeath to the generations a *Calculus of the Nervous System*." An impressive vision—10 years before, for example, [George Boole](#) would talk about similar things.

Computers and AI: Human computation (1800s and early 1900s)

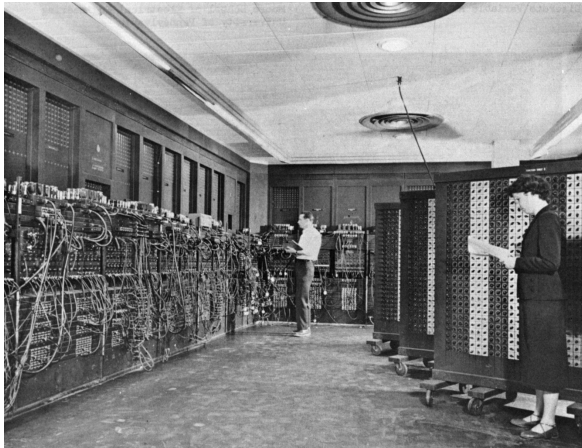


A visionary application to meteorology can be found in the scientific work of [Lewis Fry Richardson](#) who, in 1922, estimated that 64,000 humans could forecast the weather for the whole globe by solving the attending differential [primitive equations](#) numerically.^[33] Around 1910 he had already used human computers to calculate the stresses inside a masonry dam.^[34]

[https://en.wikipedia.org/wiki/Computer_\(occupation\)](https://en.wikipedia.org/wiki/Computer_(occupation))
<https://hackaday.com/2025/03/28/math-optimized-swedens-maximal-multi-divi/>
<https://www.youtube.com/watch?v=56AA1yxFoQs>

Computers and AI: Electrical (1940s)

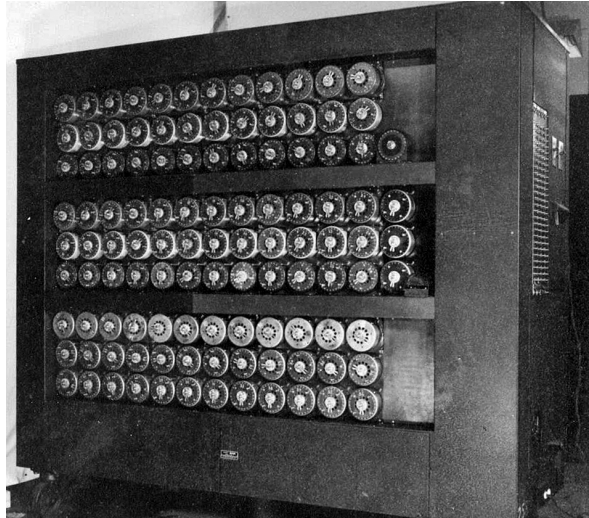
Electronic Numerical Integrator and Computer (ENIAC)



Computers and AI: Alan Turing (1940s-1950s)



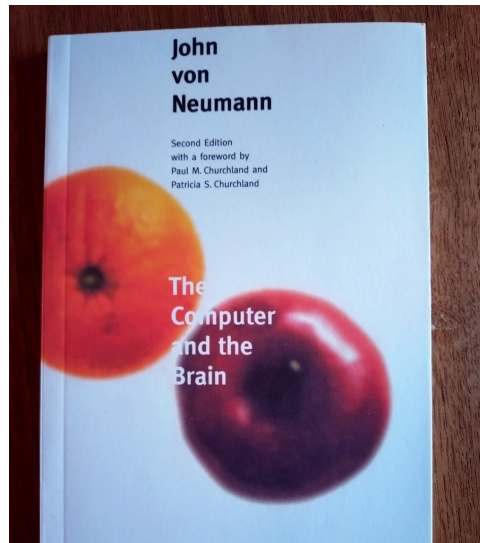
https://en.wikipedia.org/wiki/Alan_Turing
<https://en.wikipedia.org/wiki/Bombe>



Computers and AI: John von Neumann (1940s-1950s)



https://en.wikipedia.org/wiki/John_von_Neumann



Computers and AI: Perceptron neural network (1950s)

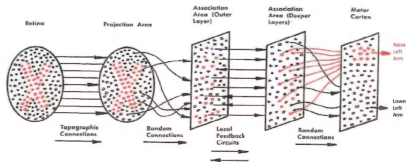


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

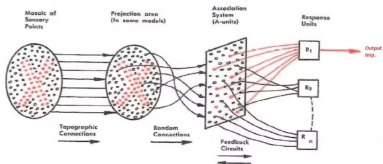
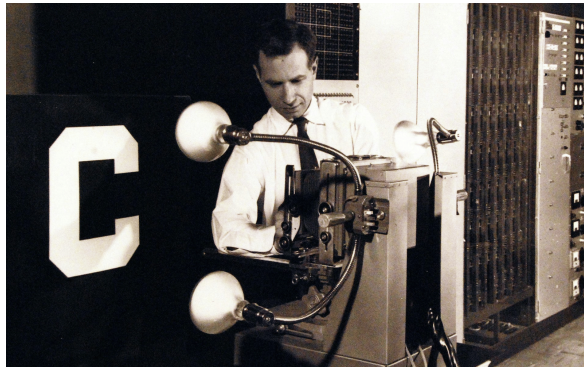


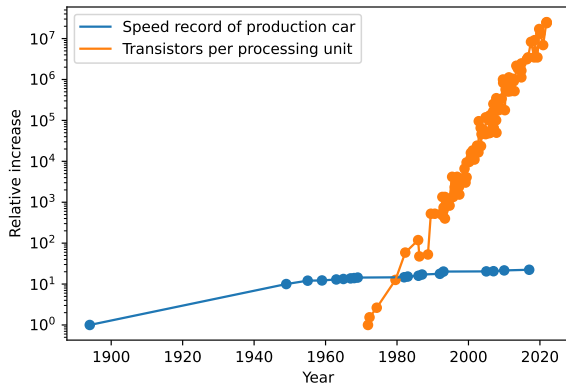
FIG. 2 — Organization of a perceptron.



Computers and AI: Vacuum tube computers and the transistor (1940s-1950s)



Computers and AI: Microprocessor development (1970s-



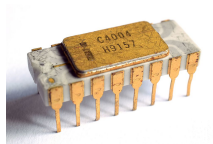
https://en.wikipedia.org/wiki/Production_car_speed_record
<https://github.com/karlrupp/microprocessor-trend-data>



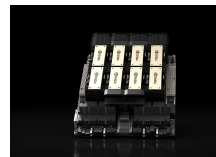
Benz Velocipede (1894)
20 km/h



Koenigsegg Agera RS (2017)
450 km/h



Intel 4004 (1971)
 2.3×10^3 transistors



Nvidia H200 (2024/2025)
 8×10^{10} transistors

Computers and AI: Deep blue (Heuristic search) (1990s)

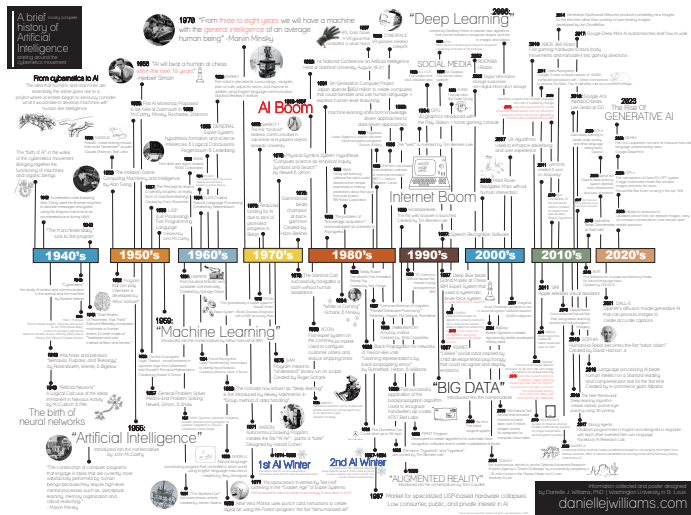


[https://en.wikipedia.org/wiki/Deep_Blue_\(chess_computer\)](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer))

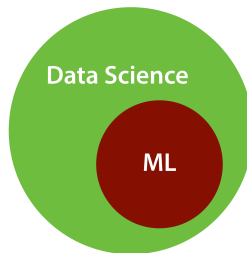
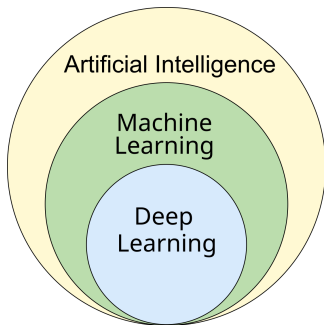
Computers and AI: Deep learning (2010s)



Computers and AI: History of AI



Computers and AI: Machine learning, AI and datascience



Data Science, but not ML: Gathering data, harmonising data, interpreting data.

Adapted from Peter Bloem.