

Data Analytics and Machine Learning Using Python

Machine Learning Using Scikit-Learn

- Introduction to Machine Learning
 - Supervised Learning
 - Unsupervised Learning
 - Reinforcement Learning
- ML using Scikit-Learn

https://github.com/indarsugiarto/ML_training

Intro to Machine Learning

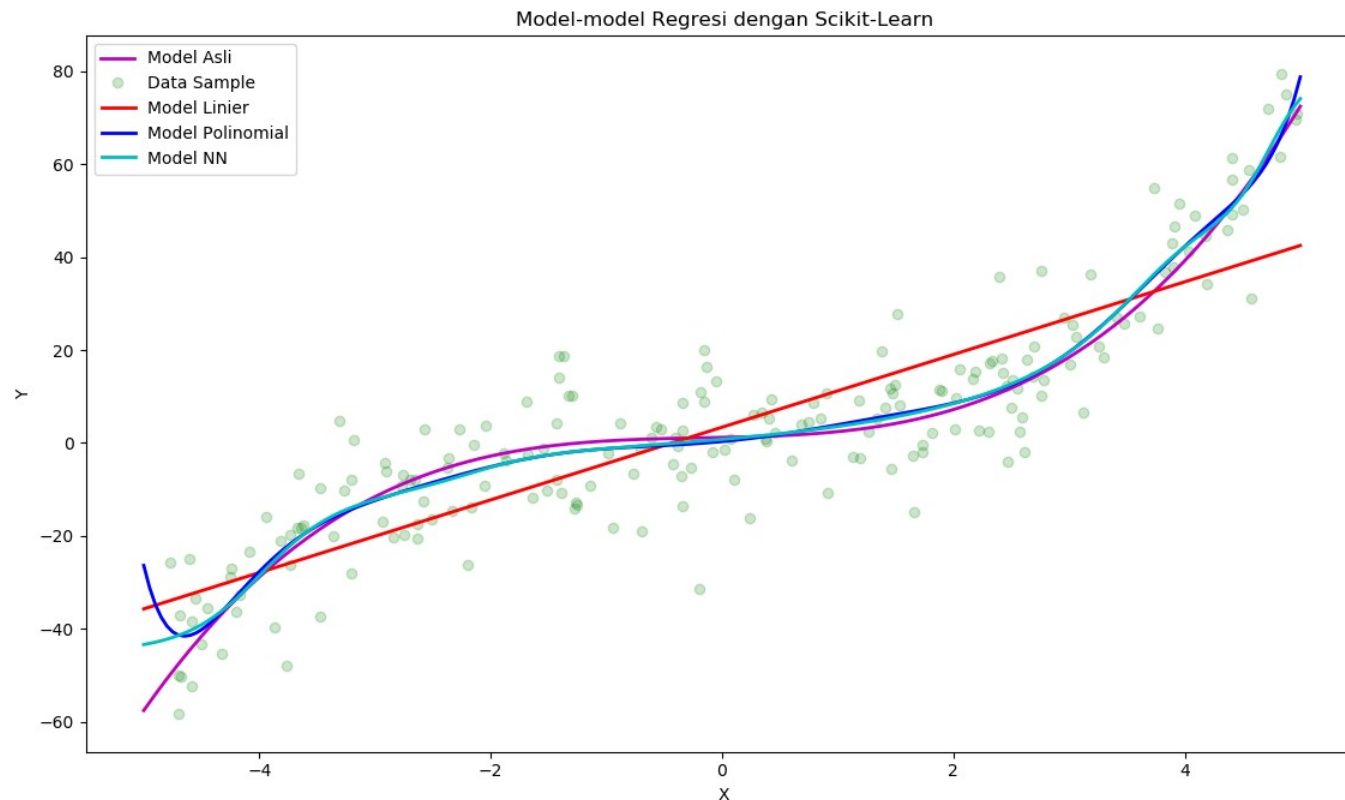
- Machine learning (ML) is a scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task.
- Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.
- It is closely related to the fields of optimization, statistics and data mining.

Wikipedia

- In general, it can be categorized as:
 1. Supervised: regression, classification
 2. Un-supervised: clustering, (pdf) estimation
 3. Semi-supervised / hybrid
 4. Reinforcement

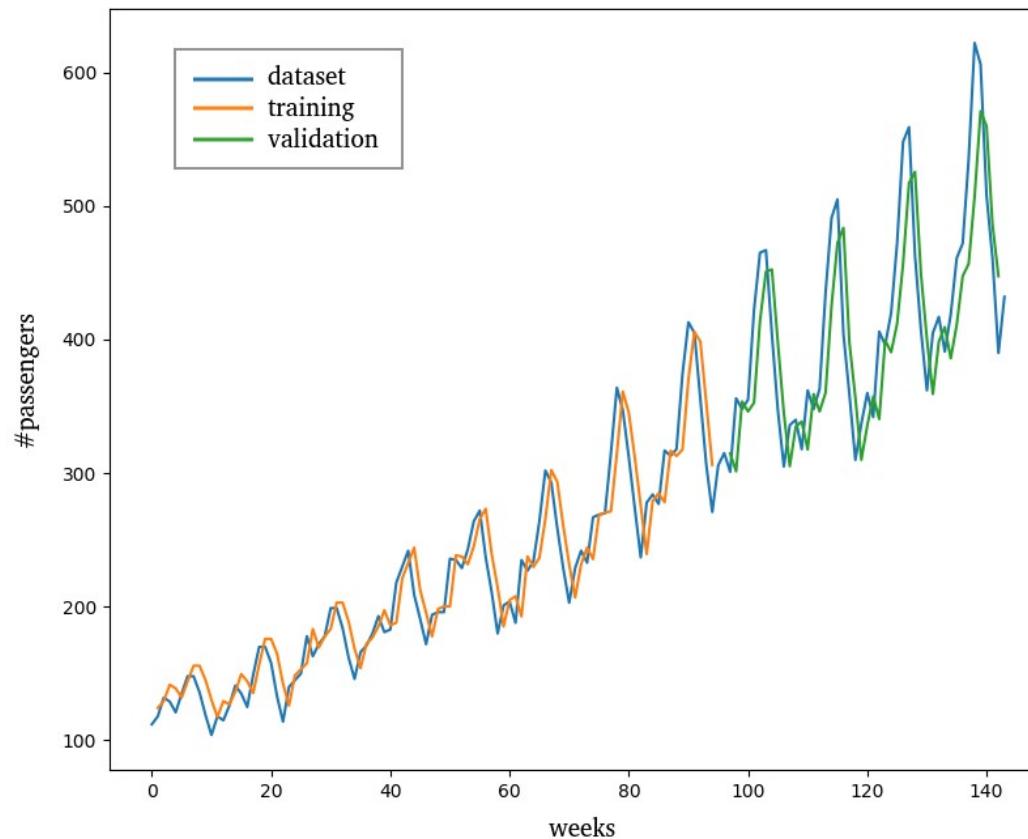
ML-Intro::Regression

- Given a set of data points $\{x^{(1)} \dots x^{(m)}\}$ associated with outcomes $\{y^{(1)} \dots y^{(m)}\}$, we want to build a model that predicts y from x .



ML-Intro::Forecasting

- Basically, it is similar to regression, but running on time series data



ML-Intro::Terminology

- Basic metrics – Given a regression model f , the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$SS_{\text{tot}} = \sum_{i=1}^m (y_i - \bar{y})^2$	$SS_{\text{reg}} = \sum_{i=1}^m (f(x_i) - \bar{y})^2$	$SS_{\text{res}} = \sum_{i=1}^m (y_i - f(x_i))^2$

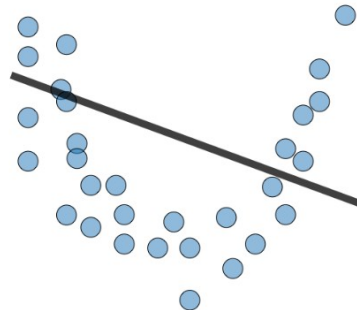
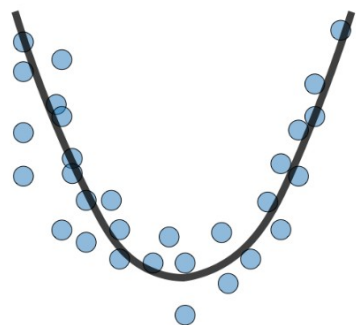
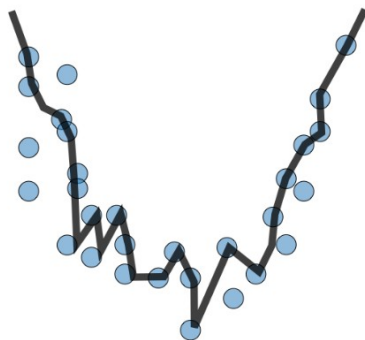
- Model selection – When selecting a model, we distinguish 3

Training set	Validation set	Testing set
<ul style="list-style-type: none"> - Model is trained - Usually 80% of the dataset 	<ul style="list-style-type: none"> - Model is assessed - Usually 20% of the dataset - Also called hold-out or development set 	<ul style="list-style-type: none"> - Model gives predictions - Unseen data

- Model selection differs in forecasting!

ML-Intro::Terminology

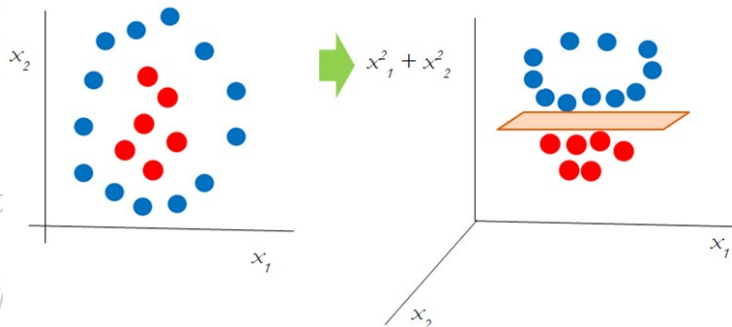
- Regularization – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues.

Symptoms	<ul style="list-style-type: none"> - High training error - Training error close to test error - High bias 	<ul style="list-style-type: none"> - Training error slightly lower than test error 	<ul style="list-style-type: none"> - Low training error - Training error much lower than test error - High variance
Regression			
Remedies	<ul style="list-style-type: none"> - Complexify model - Add more features - Train longer 		<ul style="list-style-type: none"> - Regularize - Get more data

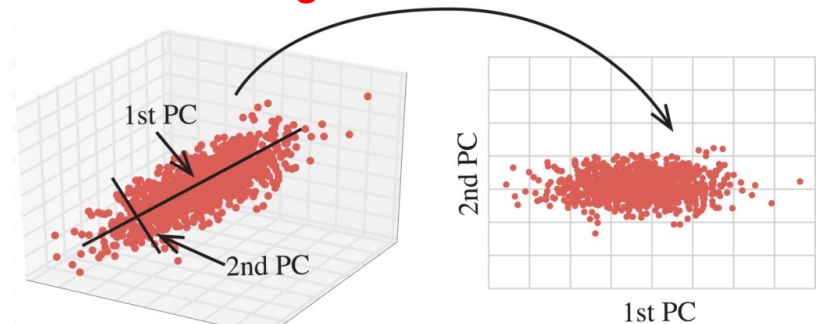
ML-Intro::Terminology

- Normalization – the process of scaling individual samples to have unit norm.
 - It might be needed to avoid out-of-range calculation, especially when involving certain activation / threshold function.
- Feature – an individual measurable property or characteristic of a phenomenon being observed.
 - Dimension is the number of features used in the data

Increasing dimension:

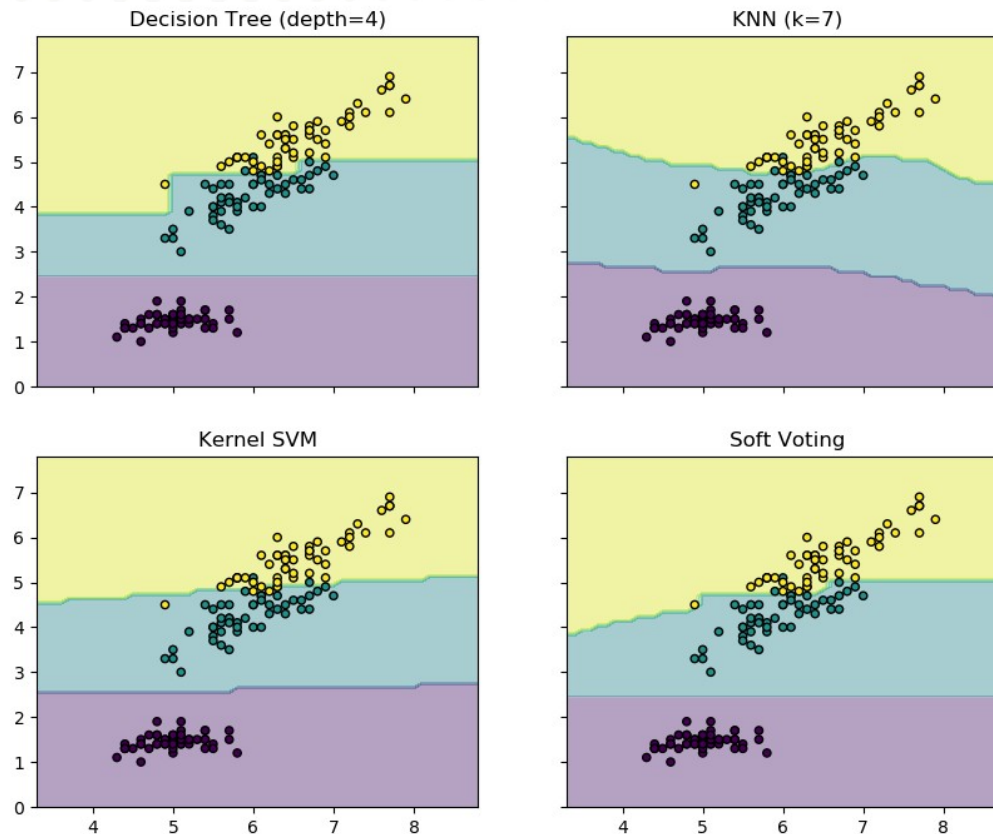


Reducing dimension:



ML-Intro::Classification

- Given a set of data points $\{x^{(1)} \dots x^{(m)}\}$ associated with classes $\{y^{(1)} \dots y^{(m)}\}$, we want to build a model that classify x into y .



ML-Intro::Terminology

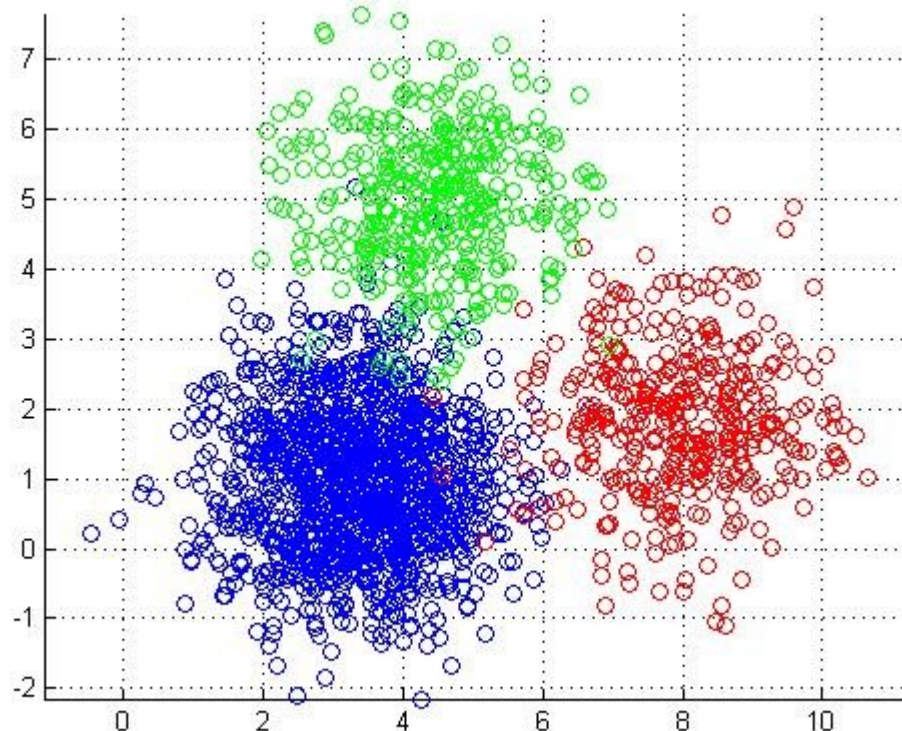
- Confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:

		Predicted class	
		+	-
Actual class	+	TP True Positives	FN False Negatives Type II error
	-	FP False Positives Type I error	TN True Negatives

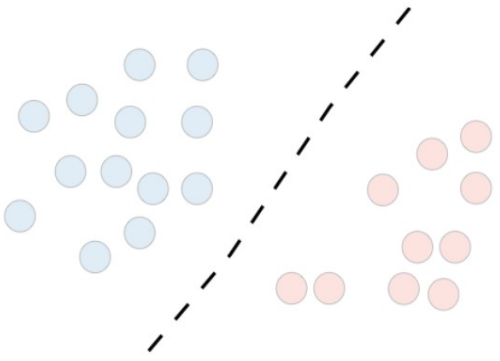
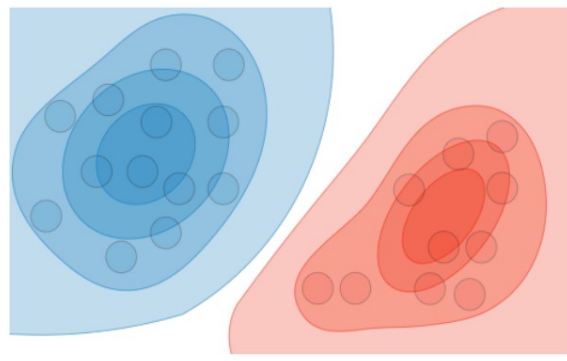
Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

ML-Intro::Clustering

- It is similar to classification, but without known labels (hence, “unsupervised”)



ML-Intro::PDF Estimation

	Discriminative model	Generative model
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes



Classification /
Clustering

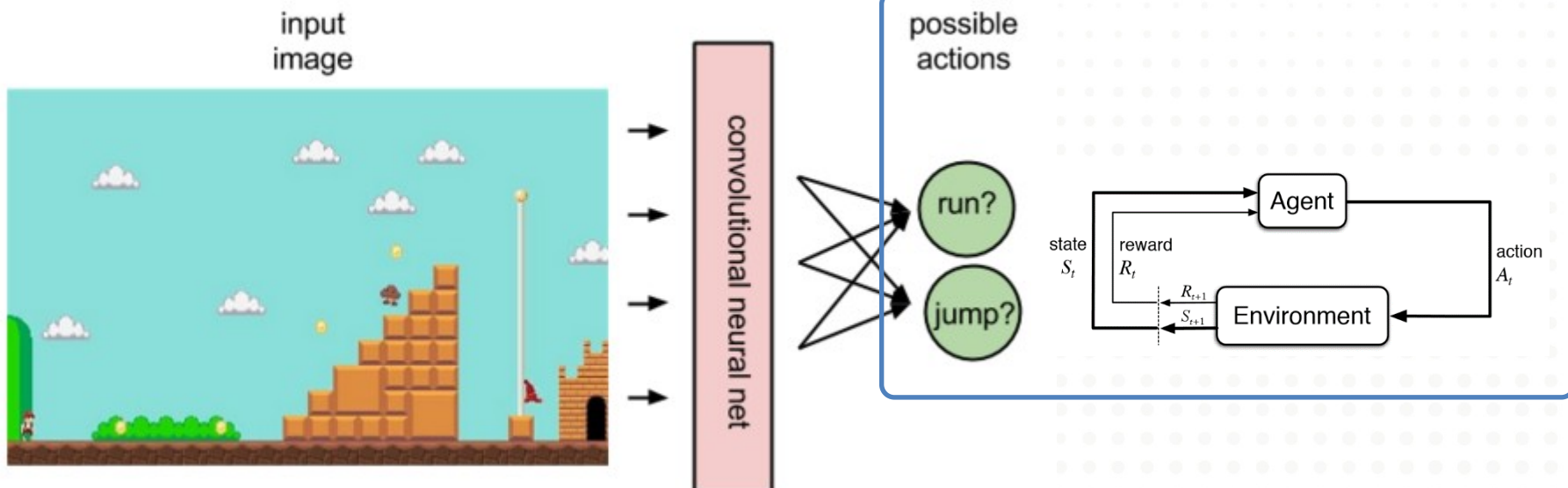


Estimation

ML-Intro::Reinforcement Learning

- Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward.

Convolutional Agent



Machine Learning Using Scikit-Learn

SKLearn-Intro

- Scikit-learn (SKLearn) is a library in Python that provides many unsupervised and supervised learning algorithms.
- It's built upon some libraries such as NumPy/Pandas, Sympy, and Matplotlib.
- Together with Scikit-image, SKLearn is one of the most actively developed machine learning tools.
- Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance.
- Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007.
 - Later Matthieu Brucher joined the project and started to use it as apart of his thesis work.
 - The project now has paid sponsorship from INRIA, Google, Tinyclues and the Python Software Foundation.

SKLearn-Intro::General Scheme

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
```

```
ds = pd.read_csv('myData.csv')
```

```
X = ds['X'].values
```

```
Y = ds['y'].values
```

```
mlp = MLPRegressor(hidden_layer_sizes=(20,20,20), activation='tanh', max_iter=1000)
```

```
mlp.fit(X, y.ravel())
```

```
X_mlp = [[i] for i in np.linspace(-3,3,100)]
```

```
y_mlp = mlp.predict(X_mlp)
```

SKLearn::Regression

- There are many regression techniques available in the Scikit-Learn library.
- Three basic but most commonly used are: linear, polynomial, logistic, and Neural Network
 - Those methods use different learning strategy, such as:
 - Gradient descent
 - Likelihood estimation
 - Newton-Raphson
 - etc

SKLearn::Linear Regression

```
import numpy as np
#import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
X_tests = np.array([[0], [2]])

lin_reg = LinearRegression()
lin_reg.fit(X, y)
print("Estimasi scikit: ", lin_reg.intercept_, lin_reg.coef_)
y_test = lin_reg.predict(X_tests)
```

See: [reg_lin.py](#)

SKLearn::Polynomial Regression

```
import numpy as np
X = 6 * np.random.rand(100, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(100, 1)

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
polynomial_regression = Pipeline((
    ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
    ("sgd_reg", LinearRegression())))

polynomial_regression.fit(X,y)
X_test = [[i] for i in np.linspace(-3,3,100)]
y_test = polynomial_regression.predict(X_test)
```

See: [reg_poly.py](#)

SKLearn::Linear and Polynomial Regression

```
class sklearn.linear_model.LinearRegression(fit_intercept=True,  
normalize=False, copy_X=True, n_jobs=None)
```

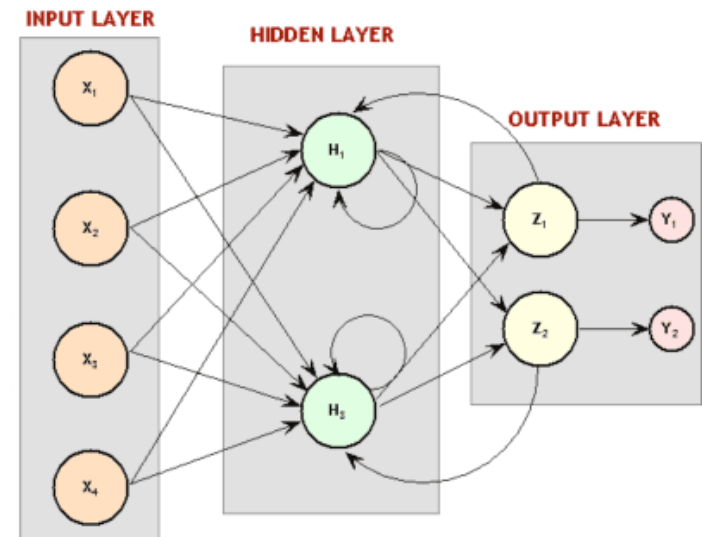
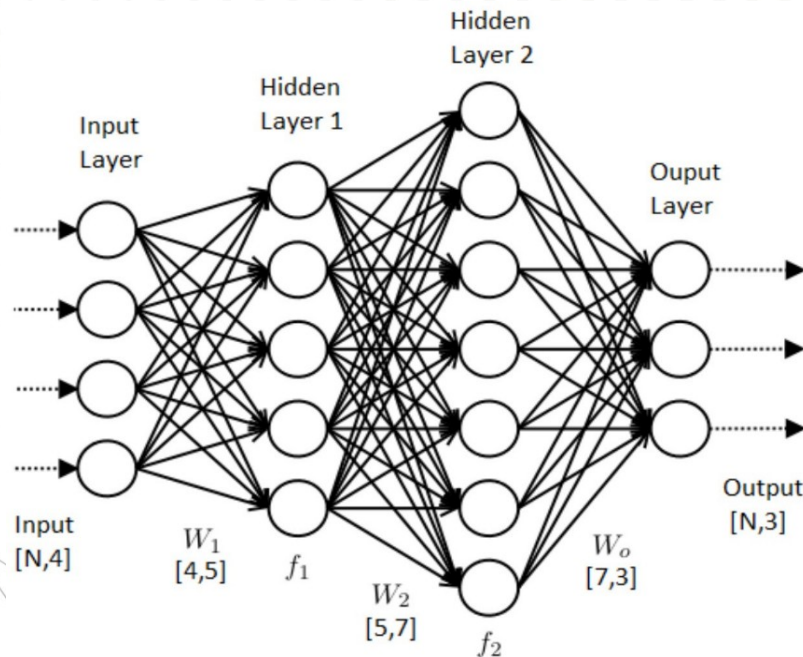
```
class sklearn.preprocessing.PolynomialFeatures(degree=2,  
interaction_only=False, include_bias=True)
```

```
class sklearn.pipeline.Pipeline(steps, memory=None)
```


SKLearn::MLP Regression

Neural network (NN) is the most commonly/widely used technique for artificial intelligence.

- There are many forms of NN: feed-forward, recurrent, CNN, spiking, etc.
- The most commonly used NN for regression is MLP (Multi Layer Perceptron)



SKLearn::MLP Regression

```
class sklearn.neural_network.MLPRegressor(hidden_layer_sizes=(100, ),  
activation='relu', solver='adam', alpha=0.0001, batch_size='auto',  
learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200,  
shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False,  
momentum=0.9, nesterovs_momentum=True, early_stopping=False,  
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,  
n_iter_no_change=10)
```

activation : {'identity', 'logistic', 'tanh', 'relu'}, default 'relu'

'identity', no-op activation, useful to implement linear bottleneck, returns $f(x) = x$

'logistic', the logistic sigmoid function, returns $f(x) = 1 / (1 + \exp(-x))$.

'tanh', the hyperbolic tan function, returns $f(x) = \tanh(x)$.

'relu', the rectified linear unit function, returns $f(x) = \max(0, x)$

solver : {'lbfgs', 'sgd', 'adam'}, default 'adam'

'lbfgs' is an optimizer in the family of quasi-Newton methods.

'sgd' refers to stochastic gradient descent.

'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, et.al

SKLearn::MLP Regression

```
import numpy as np  
from sklearn.neural_network import MLPRegressor
```

```
X = 6 * np.random.rand(100, 1) - 3  
y = 0.5 * X**2 + X + 2 + np.random.randn(100, 1)
```

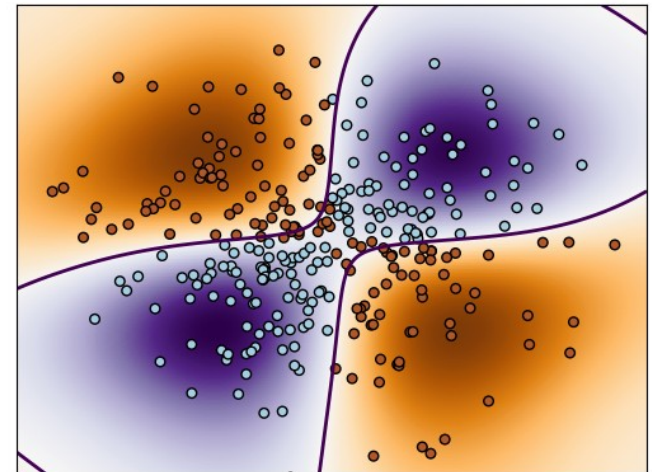
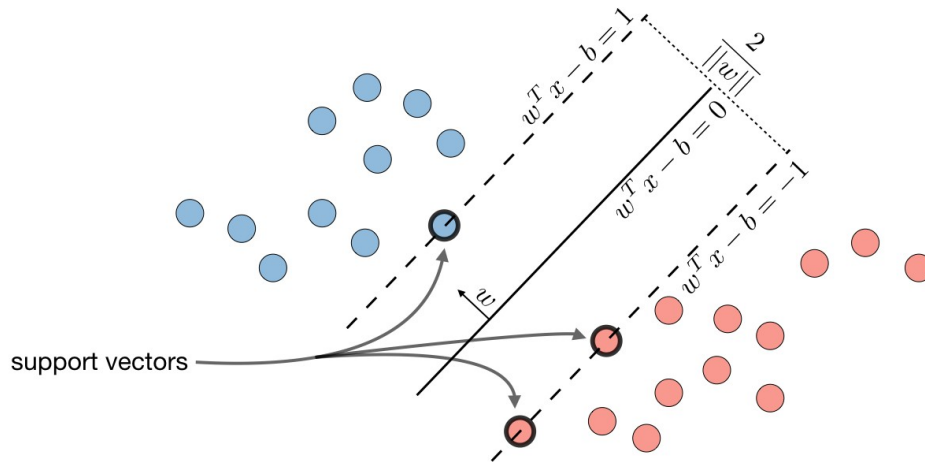
```
mlp = MLPRegressor(hidden_layer_sizes=(20,20,20), activation='tanh',  
solver='sgd', max_iter=10000)  
mlp.fit(X, y.ravel())
```

```
X_test = [[i] for i in np.linspace(-3,3,m)]  
y_test = mlp.predict(X_test)
```

See: reg_nn.py

SKLearn::Classification

- There are many classification techniques available in the Scikit-Learn library as well.
- One of the most commonly used algorithm is SVM (Support Vector Machine)
 - The goal of support vector machines is to find the line that maximizes the minimum distance to the line.



- *Remark:* the line is defined as $w^T x - b = 0$

SKLearn::SVM classifier

```
from sklearn import svm, datasets
```

```
iris = datasets.load_iris()
```

```
X = iris.data[:, :2]
```

```
y = iris.target
```

```
models = svm.SVC(kernel='rbf', gamma=0.7)
```

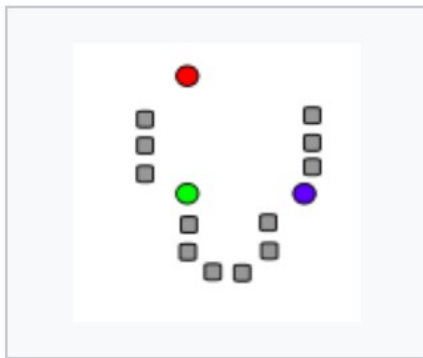
```
svm_predictor = models.fit(X, y)
```

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

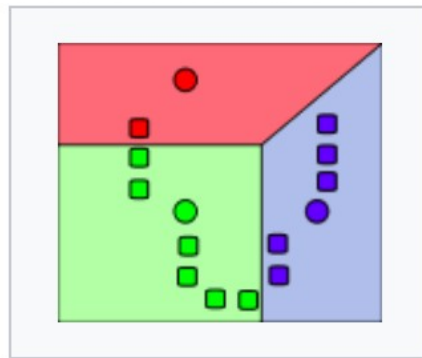
See: [plot_iris.py](#), [plot_digits_classification.py](#), [plot_face_recognition.py](#)

SKLearn::K-Means Clustering

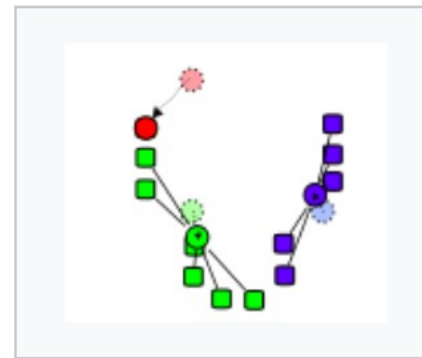
- k-means clustering is a method of vector quantization that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.
 - This results in a partitioning of the data space into Voronoi cells.



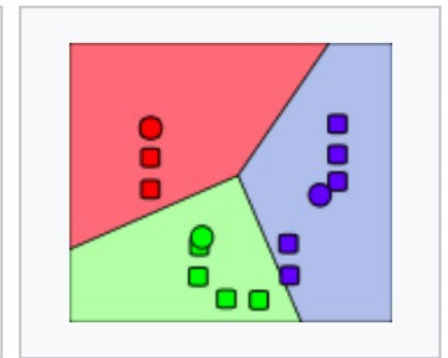
1. k initial "means" (in this case $k=3$) are randomly generated within the data domain (shown in color).



2. k clusters are created by associating every observation with the nearest mean. The partitions here represent the [Voronoi diagram](#) generated by the means.



3. The [centroid](#) of each of the k clusters becomes the new mean.



4. Steps 2 and 3 are repeated until convergence has been reached.

SKLearn::Clustering

- Clustering is similar to classification, but without labeled classes.
- In SKLearn, there are several clustering algorithm with their own pros- and cons-
 - Example algorithms: K-means, DBSCAN, Gaussian mixture, Birch, etc.
 - KMeans can be seen as a special case of Gaussian mixture model with equal covariance per component.

```
class sklearn.cluster.KMeans(n_clusters=8, init='k-means++', n_init=10,  
max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0,  
random_state=None, copy_x=True, n_jobs=None, algorithm='auto')
```


SKLearn::K-Means Clustering

```
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import KMeans
```

```
from sklearn.datasets import make_blobs
```

```
n_samples = 1500
```

```
random_state = 170
```

```
X, y = make_blobs(n_samples=n_samples, random_state=random_state)
```

```
y_pred = KMeans(n_clusters=2, random_state=random_state).fit_predict(X)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_pred); plt.show()
```

See: [plot_kmeans_assumptions.py](#)

SKLearn::Example Dataset



Attribute Information (All in centimeters)

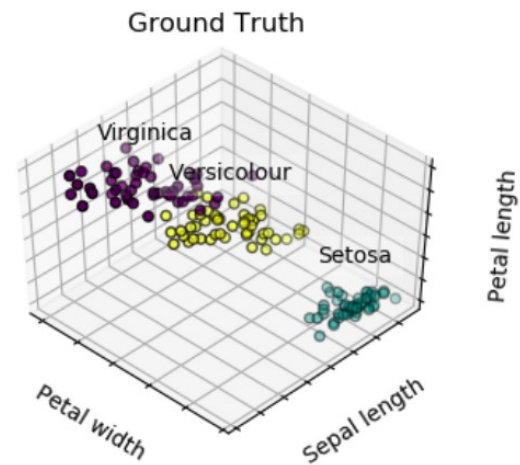
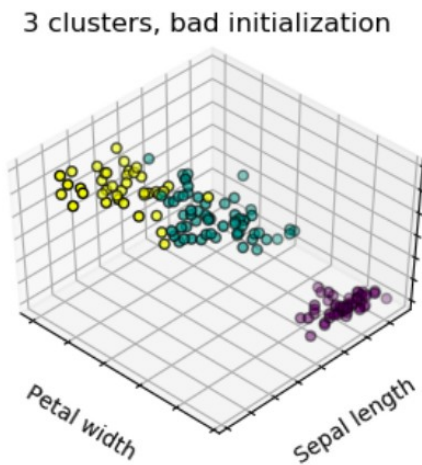
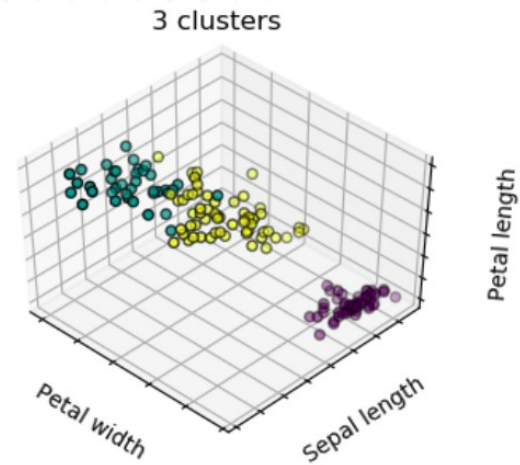
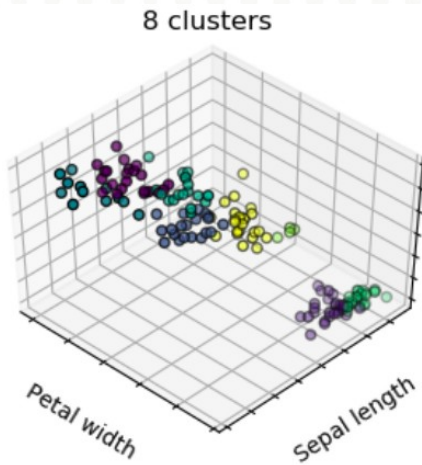
- > Sepal length
- > Sepal width
- > Petal length
- > Petal width
- > Flower class

Ex: 5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica



See: [plot_cluster_iris.py](#)

SKLearn::K-Means Clustering



See: [plot_cluster_iris.py](#)

References

- <https://scikit-learn.org/stable/tutorial/index.html>
- <https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/>
- <https://www.datacamp.com/community/tutorials/machine-learning-python>
- <https://www.kdnuggets.com/2016/04/top-10-ipython-nb-tutorials.html>
- <https://github.com/rhiever/Data-Analysis-and-Machine-Learning-Projects/>