Jakub Gałka
jgalka@agh.edu.pl, C2-419
office hours 2021: Monday 11:30-14:00

Data Analysis and Pattern Recognition

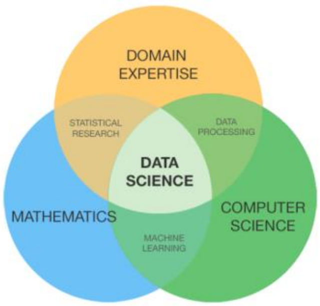
Analiza Danych i Rozpoznawanie Wzorców

» Grades
» Laboratory classes - 50%
» Brief lecture test - 50%

» Data Science, Data Engineering
» Machine Learning, Deep Learning
» Artificial Intelligence
» Big Data, IoT
» Industry 4.0

Jobs

Deep / Machine Learning Engineer
Data Analyst / Scientist / Engineer
Big Data Engineer



Data Science Job

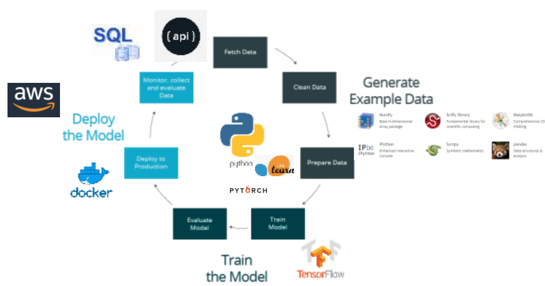
» Python, R, libraries
» Machine learning
» Statistical methods, inference, data visualisation
» Domain knowledge (eg. financial)
» SQL, noSQL
» Web, Cloud, API

Machine Learning Job

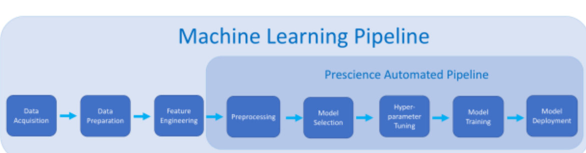
- Python, libraries, C++
- ML: regression, classification, cluster analysis, model evaluation
- Deep learning (DNN, GAN, RL ...)
- Feature engineering, DSP
- Linear algebra, applied statistics
- Serving, Docker, API, TFX, Git
- General algorithms, data structures
- Domain knowledge (NLP, computer vision, financial)

Machine Learning Pipeline

Machine Learning Pipeline



Tools



Types of ML predictions

Classification

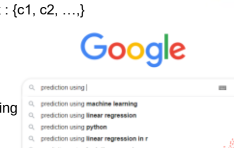
Predicts category (class, label)

Known labelled input x : {c1, c2, ….}

Regression

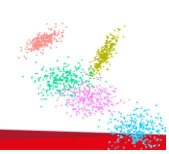
Predicts value

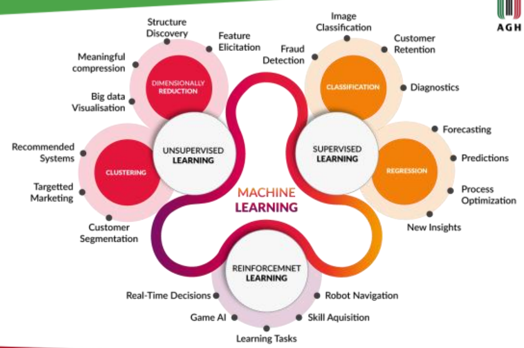
Known I/O value mapping y=f(x)

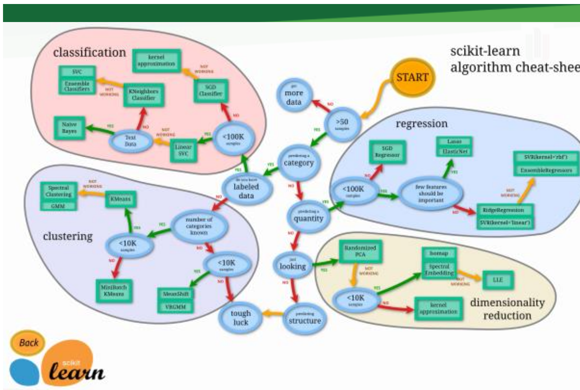


Clustering

Groups similar elements (no labels)







Classification

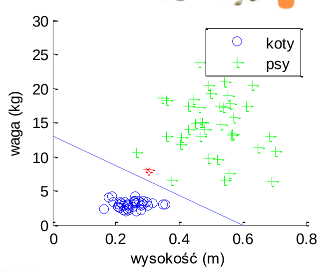
Predict known label for unseen data

x - *feature vector*

$$\boldsymbol{x} = [x_1, \ldots, x_L]^T$$

c_i *–class, label* from the set of *M* classes, *i*=1,…,*M*

$$\boldsymbol{x} \overset{?}{\rightarrow} c_i, \quad \bigcup_{i=1}^{M} c_i = \Omega$$



Bayes Classifier (MAP)

MAP (*Maximum a Posteriori prob.*): we are looking for a specific label c_i^*

$$c_i^* = \underset{c_i, i=1,\ldots,M}{\arg\max}\, P(c_i \mid \boldsymbol{x})$$

Thomas Bayes (1702-1761)

That would maximize the observed probability of the class, given the input ***x***.

Binary calssification: *M*=2

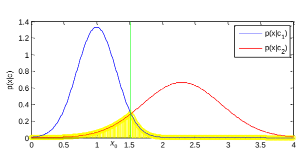
Class priors: $P(c_1)$, $P(c_2)$

If we knew likelihood $p(\boldsymbol{x}|c_i)$:

$$P(c_i \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid c_i) P(c_i)}{p(\boldsymbol{x})}$$

Binary Naive Bayes

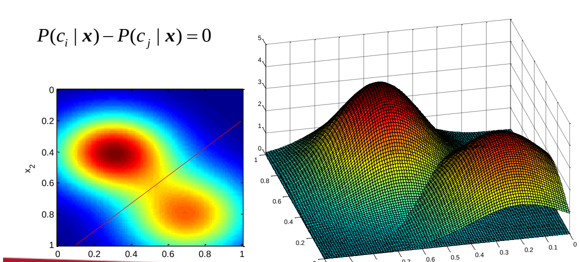
$$p(\boldsymbol{x} \mid c_1) P(c_1) \overset{?}{><} p(\boldsymbol{x} \mid c_2) P(c_2)$$



Naive Bayes – 2D

Dyscrimination Hyperplane

$$P(c_i \mid \boldsymbol{x}) - P(c_j \mid \boldsymbol{x}) = 0$$

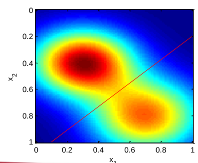


Decision function (Fischer discriminator)

Monotonic function of MAP

$$g_i(\boldsymbol{x}) \equiv f\left(P(c_i \mid \boldsymbol{x})\right) \qquad \forall j \neq i \quad g_i(\boldsymbol{x}) > g_j(\boldsymbol{x}) \Rightarrow \boldsymbol{x} \in c_i$$

Decision plane $g_{ij}(\boldsymbol{x}) \equiv g_i(\boldsymbol{x}) - g_j(\boldsymbol{x}) = 0, \quad i, j = 1, \ldots, M, \; i \neq j$



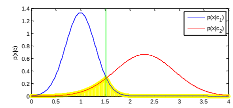
How to tackle an *N*-class problem using binary classifiers only ?

www.agh.edu.pl

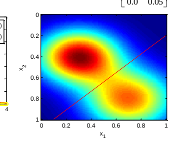
Naive Bayes – Normal PDF

Multivariate N-PDF $p(\boldsymbol{x} \mid c_i) = \frac{1}{(2\pi)^{l/2}\left|\Sigma_i\right|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \Sigma_i^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right)$

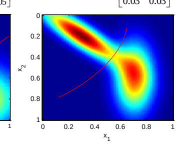
$$\boldsymbol{\mu}_i = E[\boldsymbol{x}] \qquad \Sigma_i = E\left[(\boldsymbol{x}-\boldsymbol{\mu}_i)(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\right] \qquad \boldsymbol{\Theta}_i = \{\boldsymbol{\mu}_i, \Sigma_i\}$$



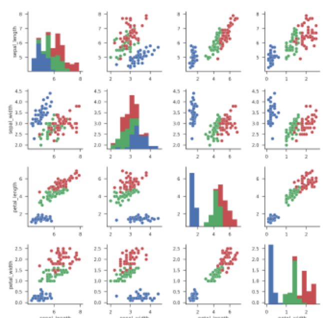
$\begin{bmatrix} 0.01 & 0.0 \\ 0.0 & 0.05 \end{bmatrix}$



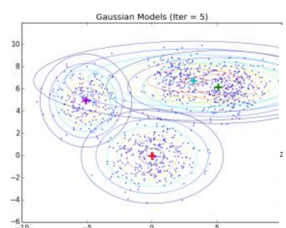
$\begin{bmatrix} 0.04 & 0.03 \\ 0.03 & 0.03 \end{bmatrix}$

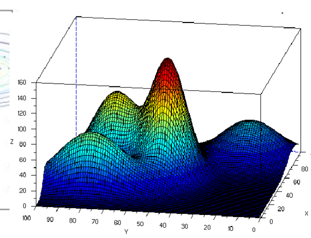


www.agh.edu.pl

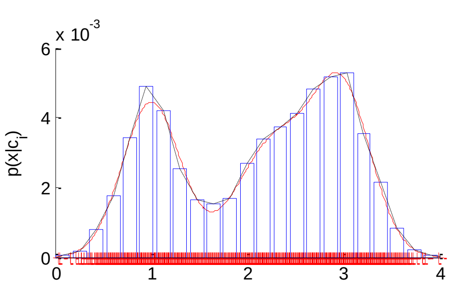


```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(iris.data, iris.target).predict(iris.data)
>>> print("Number of mislabeled points out of a total %d points : %d"
... % (iris.data.shape[0],(iris.target != y_pred).sum()))
Number of mislabeled points out of a total 150 points : 6
```





www.agh.edu.pl



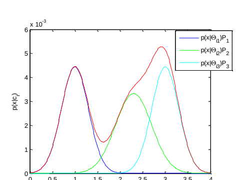
www.agh.edu.pl

Gaussian Mixture Model

GMM – statistical model of complex distributions

Carl Friedrich Gauss (1777-1855)

$$p(\boldsymbol{x} \mid c_i) = \sum_{j=1}^{J} p(\boldsymbol{x} \mid \boldsymbol{\Theta}_{ij}) P_j$$



$$\sum_{j=1}^{J} P_j = 1, \quad \int_{x \in \boldsymbol{X}} p(\boldsymbol{x} \mid \boldsymbol{\Theta}_{ij}) d\boldsymbol{x} = 1$$

$$\max\left\{ p^{ML} = \prod_k p(\boldsymbol{x}_k, \boldsymbol{\Theta}, P_1, \ldots, P_J) \right\}$$

mixture.GaussianMixture

www.agh.edu.pl

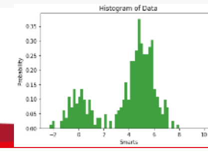
```python
import numpy as np
from sklearn import mixture
np.random.seed(1)
g = mixture.GaussianMixture(n_components=3)
# Generate random observations with two modes centered on 0
# and 10 to use for training.
obs = np.concatenate((np.random.randn(100, 1), 10 + np.random.randn(300, 1), 5 + np.random.randn(200, 1)))
g.fit(obs)
l=g.predict([[0], [2], [6], [9], [10]])
print(l)
p=g.predict_proba([[0], [2], [6], [9], [10]])
print(p)
```

```
[1 1 2 0 0]
[[1.74450697e-23 9.99985504e-01 1.44962481e-05]
 [2.78709519e-14 8.18856673e-01 1.81143327e-01]
 [3.62624405e-04 3.43844650e-10 9.99637375e-01]
 [9.98599654e-01 3.64032460e-22 1.40034569e-03]
 [9.99984542e-01 1.86890101e-27 1.54577593e-05]]
```

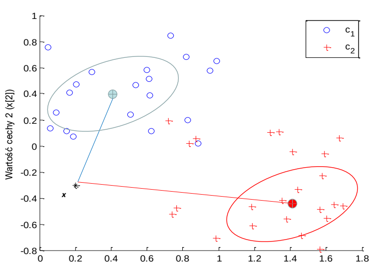


```python
import numpy as np
from sklearn import mixture
np.random.seed(1)
g1 = mixture.GaussianMixture(n_components=3)
g2 = mixture.GaussianMixture(n_components=3)
# Generate random observations with two modes centered on 0
# and 10 to use for training.
obs1 = np.concatenate((np.random.randn(100, 1), 5 + np.random.randn(300, 1))) # 0 & 5
obs2 = np.concatenate((2.5 + np.random.randn(100, 1), 6 + np.random.randn(200, 1))) # 2.5 & 6
g1.fit(obs1)
g2.fit(obs2)
p1=g1.score([[4]])
print("log p(x=4|g1)=",p1)
p2=g2.score([[4]])
print("log p(x=4|g2)=",p2)
p1=g1.score([[2]])
print("log p(x=2|g1)=",p1)
p2=g2.score([[2]])
print("log p(x=2|g2)=",p2)
print("What is P(g|x)=? , Bayes")
print("P(g1|x=2)=", np.exp(p1)*0.5/(np.exp(p1)*0.5+np.exp(p2)*0.5))
print("P(g2|x=2)=", np.exp(p2)*0.5/(np.exp(p1)*0.5+np.exp(p2)*0.5))
```

```
log p(x=4|g1)= -1.7072020753278263
log p(x=4|g2)= -2.2636967356701074
log p(x=2|g1)= -4.153287400767302
log p(x=2|g2)= -2.212724778571532
What is P(g|x)=? , Bayes
P(g1|x=2)= 0.1255860595922101
P(g2|x=2)= 0.8744139404077899
```



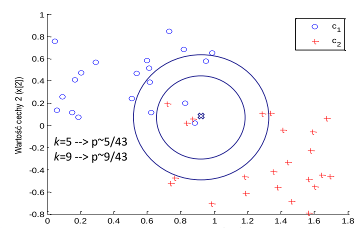
Nearest neighbor



*k*NN p(x) estimator (*k-Nearest Neighbor*)

- *k*NN: what whoul be the radius (volume) of a hypersphere with k samples inside ?
- We measure the radius or volume using some metric (e.g. Euclidean)

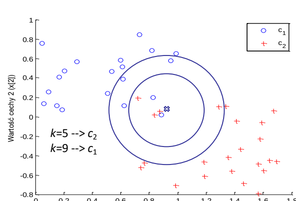
$$\tilde{p}(\boldsymbol{x})=\frac{k}{KV(\boldsymbol{x})}$$



kNN Classifier

$$c^*(\boldsymbol{x})=\underset{c_i,\, i=1,\dots,M}{\arg\max}\left(\left|\left\{x_{c_i}\in V_k(\boldsymbol{x})\right\}\right|\right)$$

$$k\%M \neq 0$$

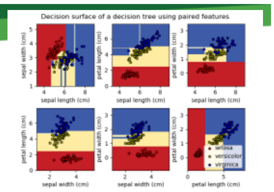


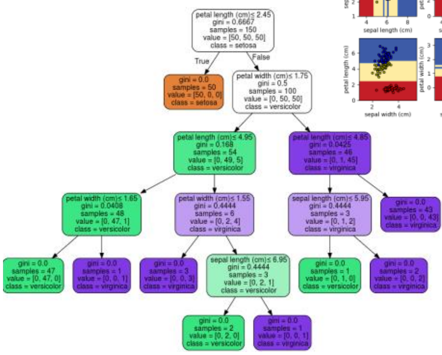
kNN – ScikitLearn

```python
X = [[0], [1], [2], [3]]
y = [0, 0, 1, 1]
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)
print(neigh.predict([[1.1]]))
print(neigh.predict_proba([[1.1]]))
print(neigh.predict([[1.9]]))
print(neigh.predict_proba([[1.9]]))
print(neigh.predict([[2.1]]))
print(neigh.predict_proba([[2.1]]))
```

```
[0]
[[0.66666667 0.33333333]]
[1]
[[0.33333333 0.66666667]]
[1]
[[0.33333333 0.66666667]]
```

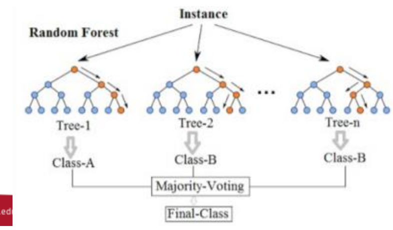
Decision tree



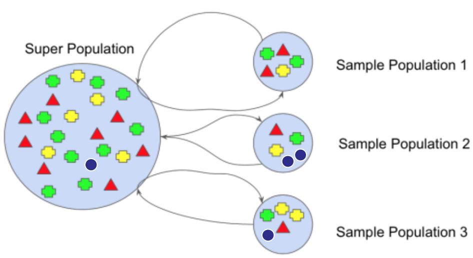


Random Forrest

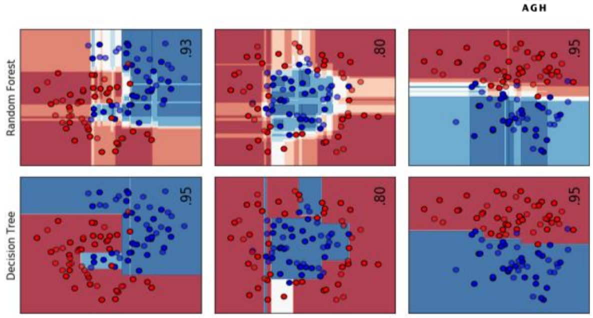
» Forrest of decision trees
» Ensemble of classifiers
» Bagging of bootstrapped classifiers
» Training using bootstrapping



Bootstrapping



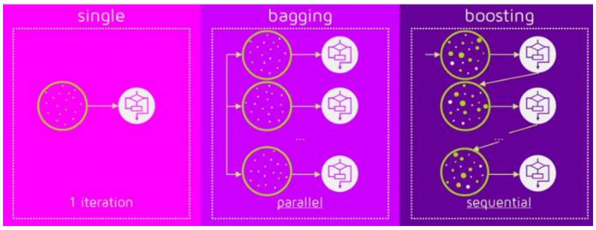

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification

>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(n_estimators=100, max_depth=2,
...                              random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=2, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
>>> print(clf.feature_importances_)
[0.14205973 0.76664038 0.0282433  0.06305659]
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

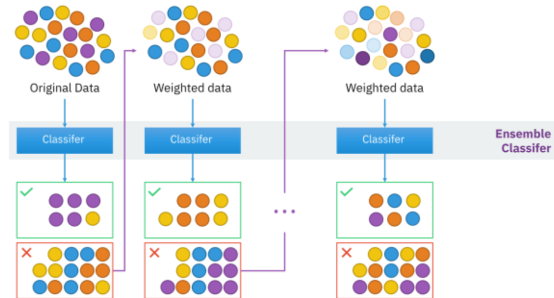
Random Forrest

» Robust, do not overfit as easily
» Can handle non-normalized data or missing data
» Can be run in parallel
» Models can be huge (RAM, cores)
» Provides feature significance information

Boosting



Boosting algorithm



XG Boost

eXtreme Gradient Boosting

» XGBoost library: Implementation for efficient boosting
  huge parallelization, efficiency
» Usually used with Trees or other weak (simple) classifiers
» Sparse aware (can deal with missing data)
» Block structure (for parallelization)
» Continued training (for boosting already trained models)

$ pip install xgboost

Artificial Neural Networks

Rosenblatt perceptron

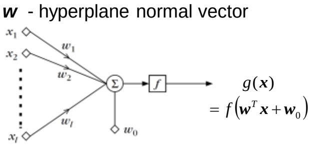
Frank Rosenblatt (1928-1971)

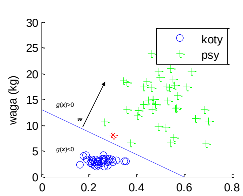
Algebraic function which defines a discriminating hyperplane

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0, \quad \mathbf{w} = [w(1), \ldots, w(L)]$$

$$g_{ij} = \mathbf{w}^T (\mathbf{x}_i - \mathbf{x}_j) = 0$$

w - hyperplane normal vector





Perceptron inference example

x=[2, 3], y* = 1 w=[-1, -3, 2],

Linear:

Y(x) =

Relu:

Loss linear:

L1 = | y – y*| =

MSE, (L2) =

Loss Relu:

L1: L2:

Perceptron inference example

x=[2, 3], y* = 1   w=[-1, -3, 2],

Linear:

Y(x) = w0 + w1*x1 + w2*x2 = -1 + (-3*2) + (2*3) = -1 + -6 +6 = -1

Relu:

Relu(-1) = 0

Loss linear:

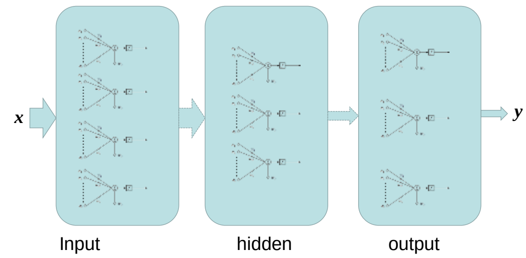
L1 = | y – y*| = |-1 – 1| = 2

MSE, (L2) = $(y – y^*)^2$ = 4

Loss Relu:

L1: |0 – 1| = 1,   L2: $(0-1)^2$ = 1

Multilayer Perceptron



MLP inference example

x=[2, 3], y* = [1, 0]   Linear: w11=[-1, -3, 2], w12=[4, 0, -1],   ReLU: w21=[0, -1, 1], w22=[0, 2, -2]

y11 = -1 + -3*2+2*3 = -1 -> w2

y12 = 4 + 0 + -1*3 = 1 -> w2

y21 = ReLU(0 -1*(-1) + 1*(1)) = 2

Y22 = ReLU(0 + 2*(-1) + -2*(1)) = ReLU(-4) = 0

L1 = sum|y*-y| = |2-1| + |0-0| = 1

L2 = (1 – 2)**2 + (0-0)**2 = 1

Loss:

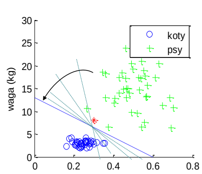
L1

MSE, (L2)

Neural weights estimation

» Iterative perceptron algorithm (classic method)

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \rho_t \sum_{x \in \boldsymbol{X}_{err}} \delta_x \boldsymbol{x}, \quad \rho_t \in \Re^+, \ \delta_x = \begin{cases} -1, & x \in c_1 \\ 1, & x \in c_2 \end{cases}$$

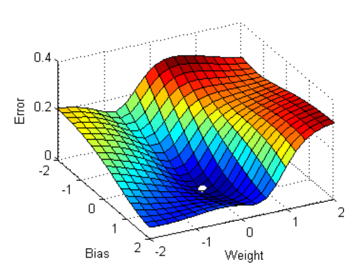


» Gradient descent algorithm (practical)

$$w' = w - L * \mathrm{d}F(w, X)/\mathrm{d}w$$

- *L*: learning rate, e.g. 10e-3
- F: Loss function
- *w*: neural weights
- *X*: training examples

Loss hyper-plane



Gradient descent example

x=[2, 3], y* = 1   w=[-1, -3, 2], L = 0.1 F_{MSE} = (y-y*)²   y(x) = ?

Y(x) = -1 + (-3)*2 + 2*3 = -1   -> F(w,x) = (-1-1)²=4

*w' = w – L ** dF(*w, X*)/dw

dF/dw = d(w0 + w1*2 + w2*3 – 1)²/dw = …

/w0:

FFN/MLP, one-hot classifier

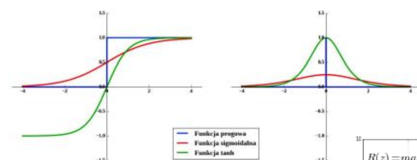
```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=False,
              epsilon=1e-08, hidden_layer_sizes=(5, 2),
              learning_rate='constant', learning_rate_init=0.001,
              max_iter=200, momentum=0.9, n_iter_no_change=10,
              nesterovs_momentum=True, power_t=0.5, random_state=1,
              shuffle=True, solver='lbfgs', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
>>> clf.predict([[2., 2.], [-1., -2.]])
array([1, 0])
```

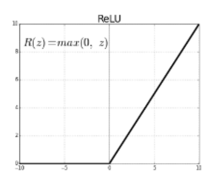


```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD

# Generate dummy data
import numpy as np
x_train = np.random.random((1000, 20))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1)), num_classes=10)
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)

model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128, verbose=0)
score = model.evaluate(x_test, y_test, batch_size=128)
print(score)
```

```
100/100 [==============================] - 0s 704us/step
[2.2926833629608154, 0.12999999523162842]
```

Activation functions and derivatives





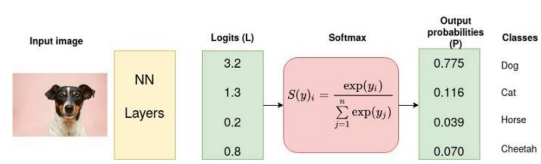
$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

Softmax network outputs

» Usually used as the network output activations
» 1 example – one class only (no multilabel problems, one-hot approach)
» Trained to produce probability estimation
» Often trained under a log-loss / cross-entropy loss function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

Softmax example



Softmax example

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

- » Y1 = 0.3   y2 = 0.1
- » P(y1) = e^0.3 / (e^0.3 + e^0.1) = a
- » P(y2) = e^0.1 / (……) = 1 – a
- » ==1

Common Loss Functions

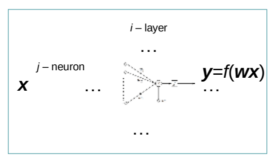
- » L1: |y – y^|
- » L2, MSE: $(y – y^\wedge)^2$
- » Binary Cross Entropy:
  –(y^ * log(p) + (1-y^)*log(1-p))

  Multi-Class Cross Entropy:
  $L(x_i, y_j)$ = - sum_j { y_{ij} * log(p_{ij}) }
  One-hot encoding: y^ = [0, 0, 1, 0, …, 0]
- » Divergence (for ratio-measures):
  log (y^/y + y/y^)   1:1 -> log(2)

Cross-Entropy example

- » x: y = [0.1, 0.6, 0.3]   y^ = [0, 0, 1]
- » L = – (y^ * log(p) + (1-y^)*log(1-p))
- » L = - (0*log(0.1) + 1 *log(0.9) +
  0* log(0.6) + 1*log(0.4) +
  1*log(0.3) + 0*log(0.7))

Backpropagation - assumptions



$$\boldsymbol{w}_{ij}^{t+1} = \boldsymbol{w}_{ij}^{t} + \Delta \boldsymbol{w}_{ij}$$

w – weights
i – layer (*i*=1,…,*I*)
j,m – neuron, synapse (*j*,*m*=1,…,N_i)
k – sample (*k*=1,…,*K*)
μ – learning rate
J – loss function
y – expected outputs
δ – gradient

$$\Delta \boldsymbol{w}_{ij} = -\mu \frac{\delta J}{\delta \boldsymbol{w}_{ij}}$$

$$J = \frac{1}{2}\sum_{k=1}^{K}\sum_{m=1}^{M}\left(y_k(m) - \hat{y}_k(m)\right)^2$$

$$\Delta \boldsymbol{w}_{ij} = -\mu \sum_{k=1}^{K} \delta_{ij,k}\, \boldsymbol{y}_{i-1,k}$$

$$\delta_{ij,k} \equiv \frac{\delta\left(\sum_{m=1}^{M}\left(y_k(m) - \hat{y}_k(m)\right)^2\right)}{\delta\left(\boldsymbol{w}_{ij,k} \cdot \boldsymbol{y}_{i-1,j,k}\right)}$$

Rys. [5]

Backpropagation - solution

$$\boldsymbol{w}_{ij}^{t+1} = \boldsymbol{w}_{ij}^{t} - \mu \sum_{k=1}^{K} \delta_{ij,k}\, \boldsymbol{y}_{i-1,k}$$

$$f(x) = \frac{1}{1+\exp(-\alpha x)}$$

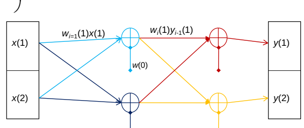
$$f'(x) = \alpha f(x)(1 - f(x))$$

Solution for *J*=*L**MSE(***y***-***y***^)

$$i = I: \quad \delta_{i=I,j,k} = \left[f\left(\boldsymbol{w}_{ij}\boldsymbol{y}_{i-1,k}\right) - \hat{y}_k(j)\right] f'\left(\boldsymbol{w}_{ij}\boldsymbol{y}_{i-1,k}\right)$$

$$i < I: \quad \delta_{i-1,j,k} = \left(\sum_{m=1}^{N_i} \delta_{im} w_{im}(j)\right) f'\left(\boldsymbol{w}_{i-1,j}\boldsymbol{y}_{i-2,k}\right)$$

w – weights
i – layer (*i*=1,…,*I*)
j,m – neuron, synapse (*j*,*m*=1,…,N_i)
k – sample (*k*=1,…,*K*)
μ – learning rate
J – loss function
y – expected outputs
δ – gradient



Backpropagation - implementation

1. Inicjalizacja – wylosuj wszystkie wagi $\boldsymbol{w}^{t=0}$
2. Wyznacz odpowiedzi $\boldsymbol{y}_{i,k}=f(\boldsymbol{w}\boldsymbol{y}_{i-1,k})$ wszystkich neuronów sieci dla każdego wektora treningowego (*forward step*)
3. Oblicz funkcję kosztu *J* (krok pośredni)
4. *i*=*I*, Dla wszystkich wzorców (*k*) wyznacz wartość $\boldsymbol{\delta}_{Ij,k}$ każdego neuronu wyjściowego (*backward step*)
5. Dla wszystkich warstw (kolejno wstecz) *i*=*I*,*I*-1,…,2 wyznacz $\boldsymbol{\delta}_{i-1,j,k}$ wszystkich neuronów dla każdego wzorca (*k*)
6. Ustaw wszystkie wagi $$\boldsymbol{w}_{ij}^{t+1} = \boldsymbol{w}_{ij}^{t} - \mu \sum_{k=1}^{K} \delta_{ij,k}\, \boldsymbol{y}_{i-1,k}$$
7. t=t+1, *goto* 2, *until* $J<J_{tr}$ OR *dJ*/*d****w***<*e* OR *t*>*T*

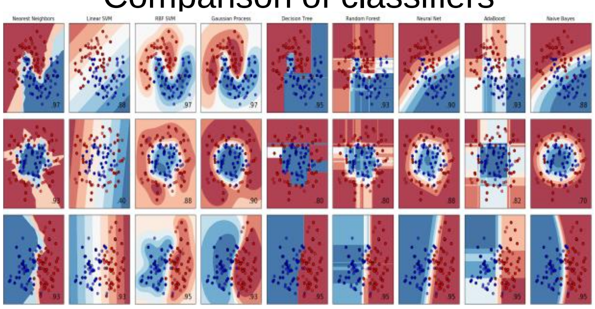
Stochastic gradient descent

» GD Problems
- RAM
- Computational power

» SGD Solution
- Batch
- Batch-size
- Epoch

» https://losslandscape.com/gallery/

Comparison of classifiers

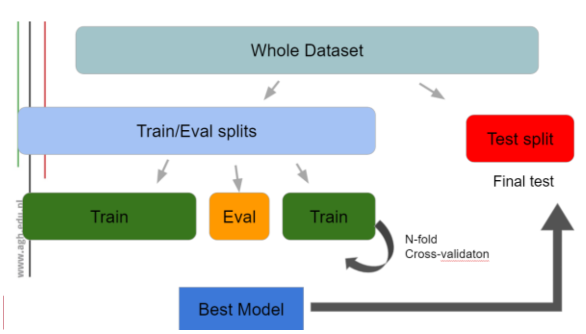


Model evaluation

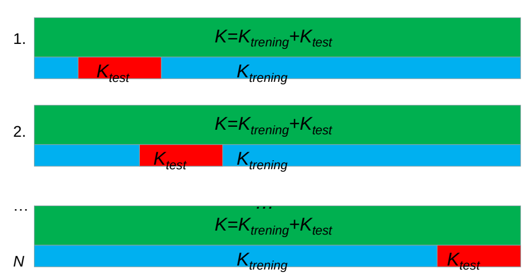
» How good is our model?
» Generalization, Robustness

» Loss Function
» Performance: Accuracy, Error rate (%)
» Domain-specific metric

Model evaluation schema



N-fold Cross-validation $ERR = \frac{1}{N}\sum_{n=1}^{N} ERR_n$



Leave-one-out

1. $K=K_{trening}+K_{test}$

2. $K_{test}=1$ $K_{trening}$

   $K_{test}=1$ $K_{trening}$

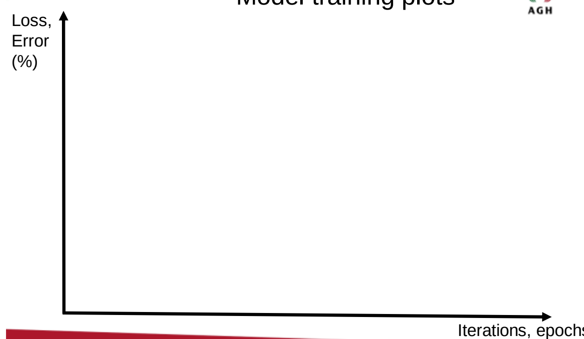
…

…

K $K_{trening}$ $K_{test}=1$

Maximizing the count of both training and validation sets.

Model training plots



Overfitting

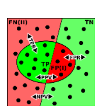
» Train/Eval mismatch
» Data problem
» Model problem
» Optimizer problem

How to avoid overfitting

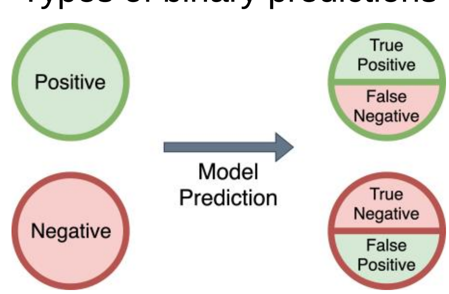
» Better Data (quantity, quality, representativeness)
» Data augumentation
» Feature engineering
» Hyper-parameter optimization (x-val)
» Model (architecture and size change)
» Regularization L1, L2
» Batch-Normalization
» Dropout
» Shorter training

Binary classification

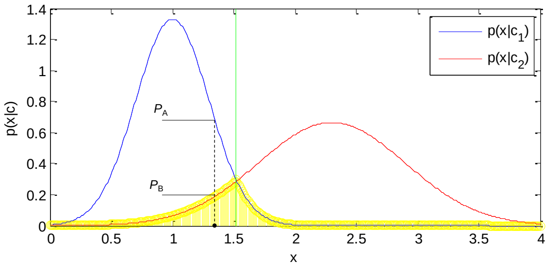
» Two-class problem
» Often stated as thresholding problem
» How to assess detection performance and confidence
» Is the decision true or false ?
» How good is it ?



Types of binary predictions



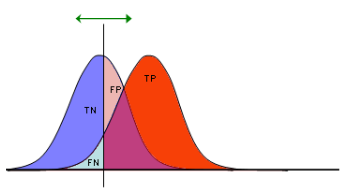
MAP probability

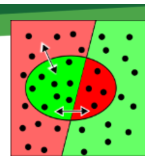


$P_A > P_B$

Performance of Binary Classifiers (detectors)

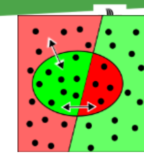
- Accuracy (%) not the best choice (class quantity bias)
- TP, TN, FP, FN
- Recall (TPR, Sensitivity), Precision
- AUC, F-score





TP	FP
FN	TN
1	1

Binary classification evaluation



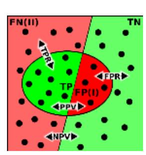
		Condition (as determined by "Gold standard")		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	True Positive	False Positive (Type I error)	Positive predictive value = Σ True Positive / Σ Test Outcome Positive
	Test Outcome Negative	False Negative (Type II error)	True Negative	Negative predictive value = Σ True Negative / Σ Test Outcome Negative
		Sensitivity = Σ True Positive / Σ Condition Positive	Specificity = Σ True Negative / Σ Condition Negative	

źr.: http://en.wikipedia.org/wiki/Precision_and_recall

Binary classification performance

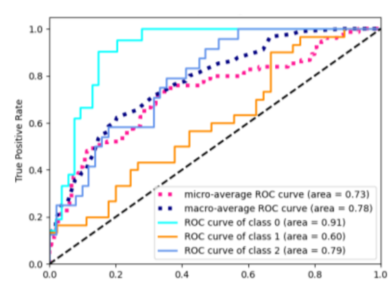
- Recall, Sensitivity, True Positive Rate

  Recall = TP / (TP + **FP**)



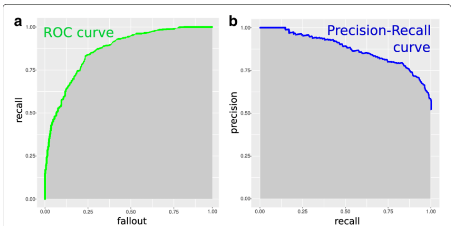
- Precision = TP / (TP + **FN**)
- Accuracy = (TP + TN) / All
- F-measure = 2*Precision*Recall/(Precision+Recall)

ROC - *Receiver Operating Characteristic* (TPR vs FPR)
AUC – Area Under Curve

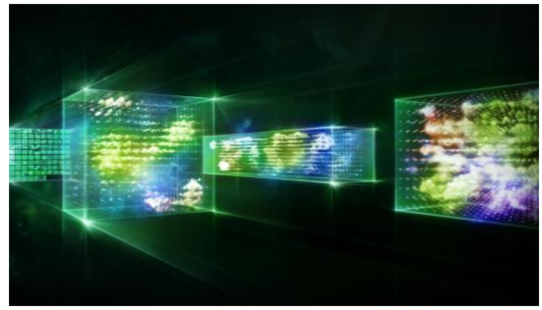


Precision-Recall Plot

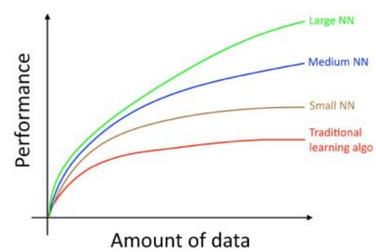
for inbalanced data



Why going deep ?



Why Deep Learning ?



Why Now ? (2010+)

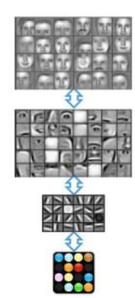
» More data
» More computational power
» More interest -> more people
» Better algorithms
» Better results
» More applications

and loop…

What is Deep Learning

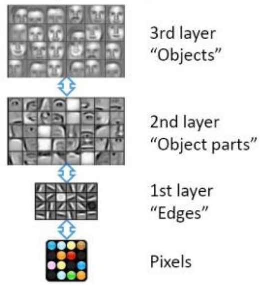
WiKi:

» Deep learning is a class of machine learning algorithms
» use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input.
» learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners.
» learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.



Layered abstraction

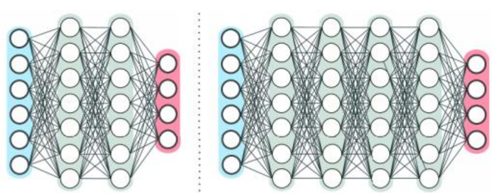
Feature representation



How To „Deep Learning" ?

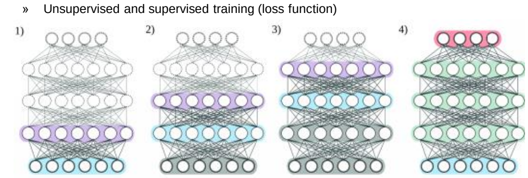
Caffe, Caffe2, Chainer, mxnet, PaddlePaddle, PYTORCH, TensorFlow, theano, torch, WOLFRAM

Normal vs Deep Feed-Forward Network



Deep network training

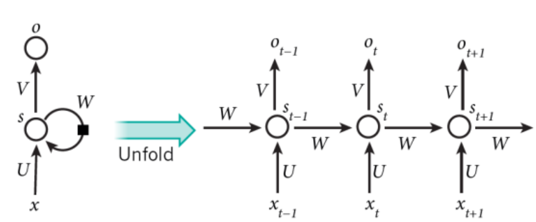
- Network pretraining (1-3) and Fine-tunning (4)
- Unsupervised and supervised training (loss function)



- Regularization (L2, L1, Dropout)
- Data augumentation
- Meta-parameters optimization (Batch size, net size)
- Evaluation (Metric)

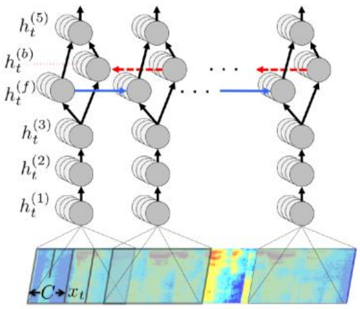

```python
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)

dataset = numpy.loadtxt("my_dataset.csv", delimiter=",")
# input (X) and output (Y)
X = dataset[:,0:39]
Y = dataset[:,39]

# create model
# 12, 8, 4, and 1 neuron in consecutive layers
model = Sequential()
# input layer
model.add(Dense(12, input_dim=39, activation='relu'))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(Dense(8, activation='relu'))
model.add(layers.Dropout(0.1, noise_shape=None, seed=None))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Output layer have one leuron - '1' for Speech, '0' for non-speech frame

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
model.fit(X, Y, epochs=100, batch_size=20)

# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# predict output
predictions = model.predict(X)
```

Recurrent Neural Networks



Recurrent Neural Networks

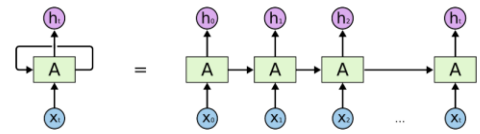
BAIDU (12.2014) Deep Speech: Scaling up end-to-end ASR



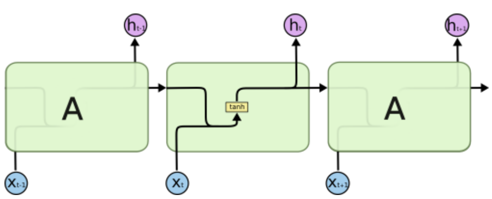
RNN vs LSTM

» Vanishing gradient problem

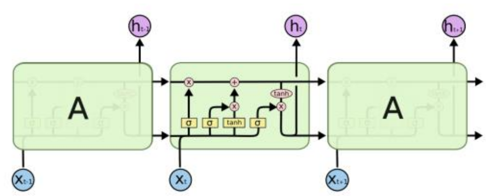
» RNN acts as a very deep FF



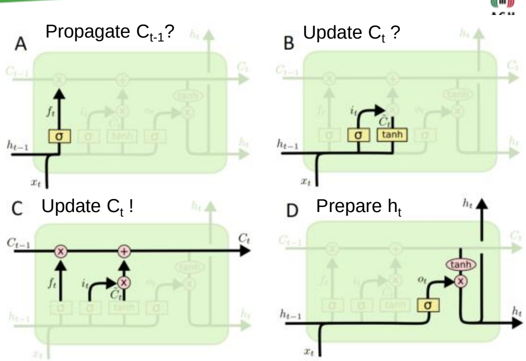
Standard RNN unwrapped



Long Short-Term Memory Networks (1997)

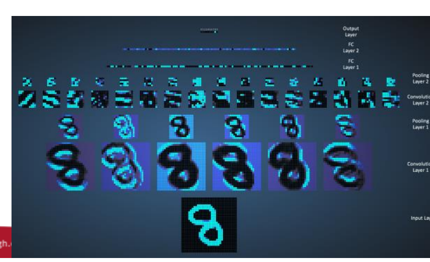


Steps of LSTM operation

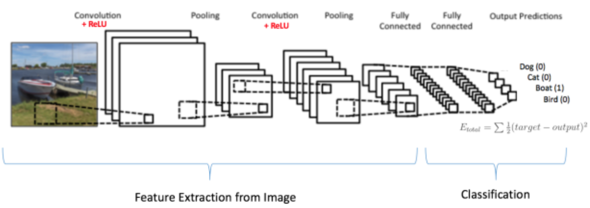


Convolutional Neural Networks

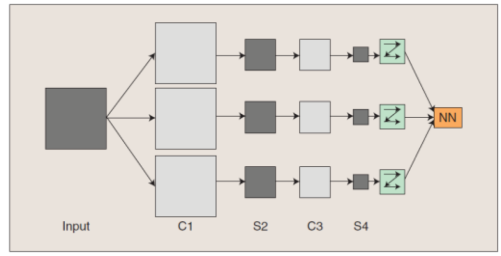
- In fully conected FF networks number of weights is to big for huge inputs.
- Same features can be observed in different places of the same signal (as in images)
- We can **group neurons** to analyse different parts of data –> lower number of weights



CNN workflow



Convolutional Neural Networks



Max-pooling, Average-pooling

Mapa aktywacji

15	8	0	3
8	5	40	1
7	6	33	8
7	8	10	9

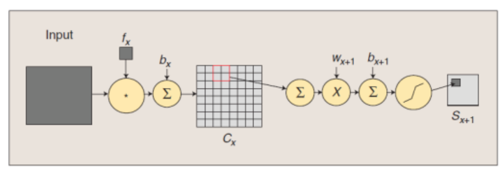
Maksimum aktywacji

15	40
9	35

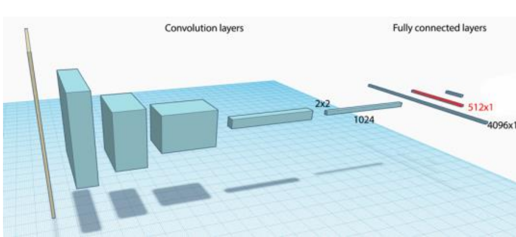
Średnia aktywacji

9	11
7	15

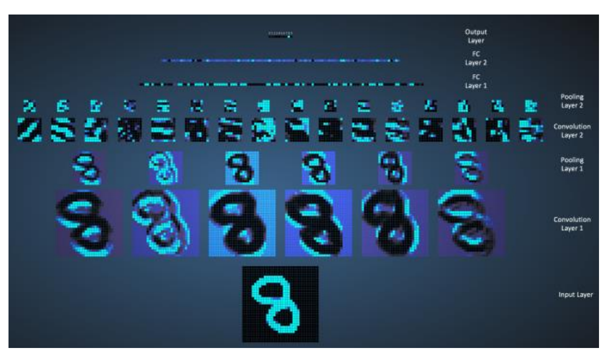
Elementary CNN Subsampling



Exemple of typical CNN-FC model



CNN classification



CNN concepts

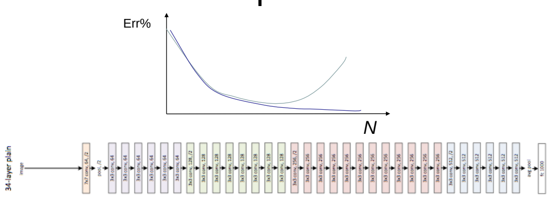
- Kernel / filter size,
- Number of kernels
- Stride – kernel shift (usually 1)
- Padding
- Pooling – reduces dimensionality
    - Max-pooling, Average-pooling
- Flattening

CNN in Keras

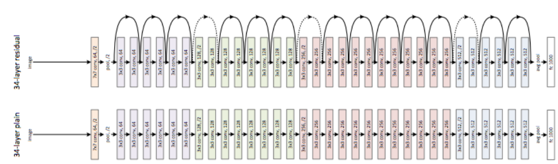
```python
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```python
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

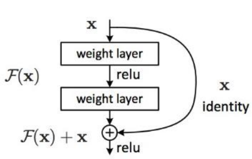

Deep CNN



ResNet - Residual Networks



ResNet - Residual Networks



$$y = x + F(x)$$

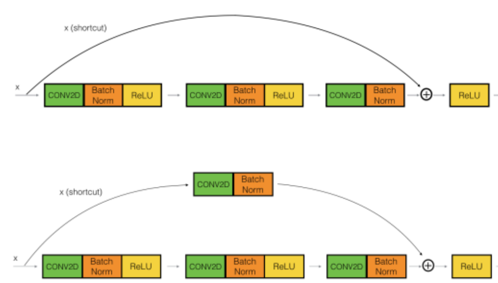
$$\frac{\delta E}{\delta x} = \frac{\delta E}{\delta y} * \frac{\delta y}{\delta x} = \frac{\delta E}{\delta y} * (1 + F'(x))$$

$$= \frac{\delta E}{\delta y} + \frac{\delta E}{\delta y} * F'(x)$$

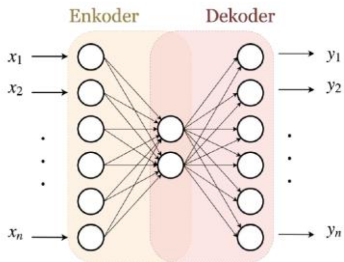
X_shortcut = X # Store the initial value of X in a variable

Here perform convolution + batch norm operations on X

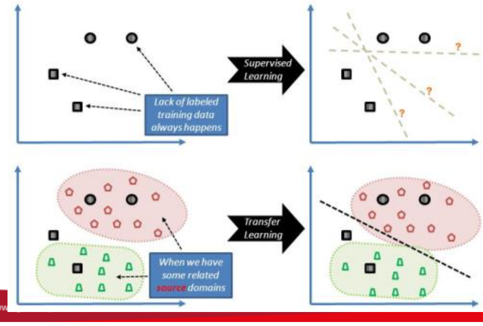
X = Add()([X, X_shortcut]) # SKIP Connection

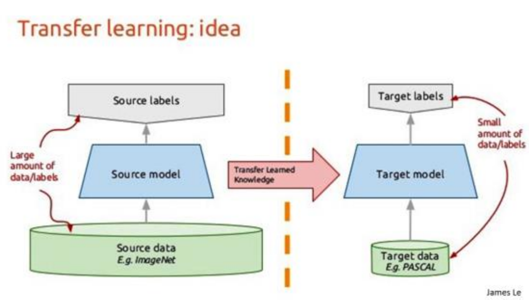


Autoencoders

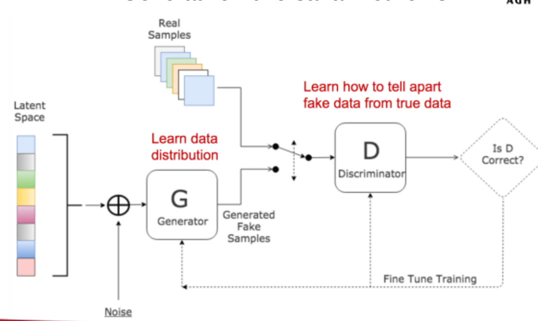


Transfer Learning

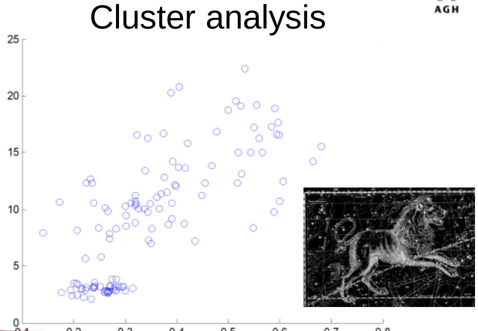




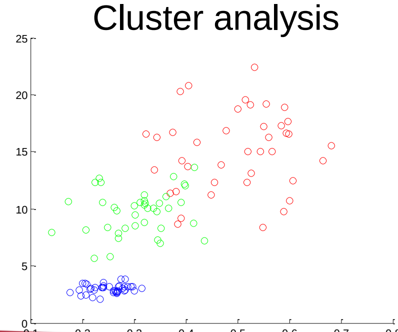
Generative Adversarial Networks



Cluster analysis



Cluster analysis

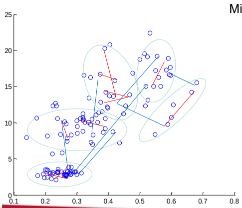


Analiza skupień

» Przyporządkowanie obiektów (wektorów cech) do K zbiorów $\boldsymbol{X}_{k=1..K}$, w których obiekty z danego zbioru $\boldsymbol{X}_k$ są do siebie podobne, a między zbiorami różne.

» Kryterium podobieństwa: miara odległości (np. metryka euklidesowa)

» Kryterium jakości przyporządkowania



Minimalizacja zmienności w zbiorze:

$$q=\min_{\{X_k\}}\sum_{k=1}^{K}\sum_{\substack{x_i,x_j\in X_k\\ x_i\neq x_j}}\delta(\boldsymbol{x}_i,\boldsymbol{x}_j)$$

Maksymalizacja zmienności między zbiorami:

$$r=\max_{\{X_k\}}\sum_{k=1}^{K}\sum_{\substack{x_i\in X_k\\ x_j\notin X_k}}\delta(\boldsymbol{x}_i,\boldsymbol{x}_j)$$

Analiza skupień - zastosowania

» *Data-mining*, wykrywanie prawidłowości w zbiorach danych

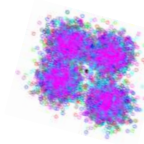
» Generowanie hipotez na podstawie danych

» Weryfikacja hipotez na podstawie danych

» Redukcja wymiarowości, kwantyzacja

» Kompresja, kodowanie

» Modelowanie



Metody hierarchiczne *Linkage*

» Metody iteracyjne

» Proste w implementacji

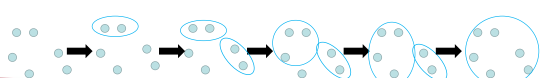
» Dość szybkie

» Algorytm *Linkage*:

– Łączenie (aglomeracja) skupień otrzymanych w poprzednim kroku działania algorytmu.

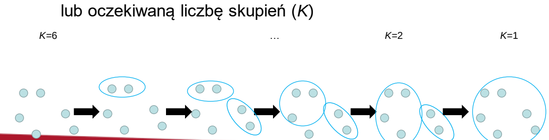
– Łączenie na podstawie podobieństwa minimalnego między obiektami oraz zbiorami. (*x*~***X***, *x*~*x*, ***X***~***X***)

– Iteracyjne powtarzanie łączenia

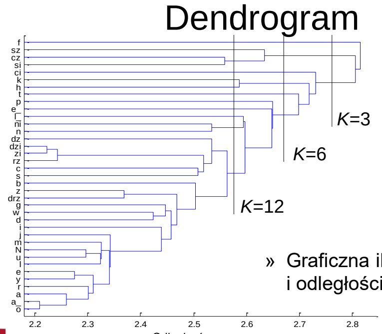


Algorytm *Linkage*

1. Inicjalizacja: każdy wektor $\boldsymbol{x}_i$ jest 1-elementowym zbiorem $\boldsymbol{X}_i$ (skupieniem)
2. Znajdź najmniejszą **odległość między** wszystkimi **zbiorami**
   $d_{\min}=\min(\ d(\boldsymbol{X}_i,\boldsymbol{X}_j)\ ),\ i!=j$
3. Połącz te zbiory, dla których znaleziono $d_{\min}$
4. Powtarzaj 2) i 3) aż otrzymasz 1 zbiór (skupienie) lub oczekiwaną liczbę skupień (K)



Dendrogram



» Graficzna ilustracja skupień i odległości między zbiorami

Definicje odległości dla zbiorów

» *Single Linkage*

$$d_{\text{single}}=\min_{\substack{x\in X_i\\ y\in X_j}}\delta(\boldsymbol{x},\boldsymbol{y})$$

» *Complete Linkage*

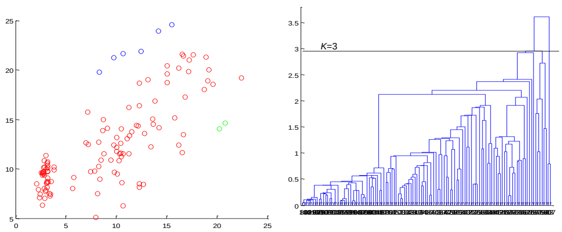
$$d_{\text{complete}}=\max_{\substack{x\in X_i\\ y\in X_j}}\delta(\boldsymbol{x},\boldsymbol{y})$$

» *Average Linkage*

$$d_{\text{average}}=\frac{1}{|\boldsymbol{X}_i||\boldsymbol{X}_j|}\sum_{x\in X_i}\sum_{y\in X_j}\delta(\boldsymbol{x},\boldsymbol{y})$$

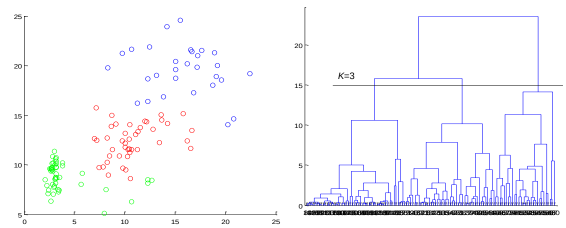
Single linkage przykład

$$d_{\text{single}} = \min_{\substack{x \in X_i \\ y \in X_j}} \delta(x, y)$$



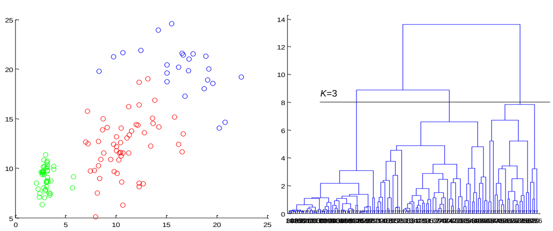
Complete linkage przykład

$$d_{\text{complete}} = \max_{\substack{x \in X_i \\ y \in X_j}} \delta(x, y)$$

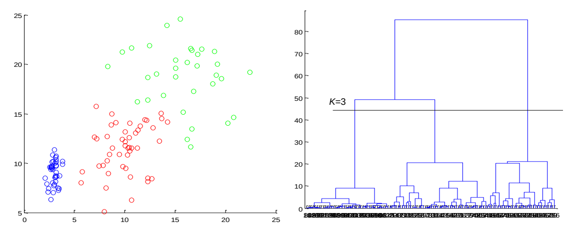


Average linkage przykład

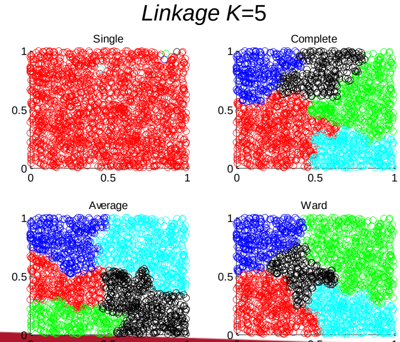
$$d_{\text{average}} = \frac{1}{|X_i||X_j|} \sum_{x \in X_i} \sum_{y \in X_j} \delta(x, y)$$



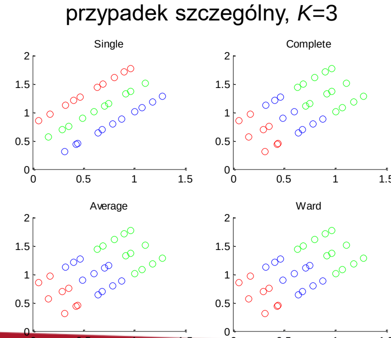
Ward's linkage przykład



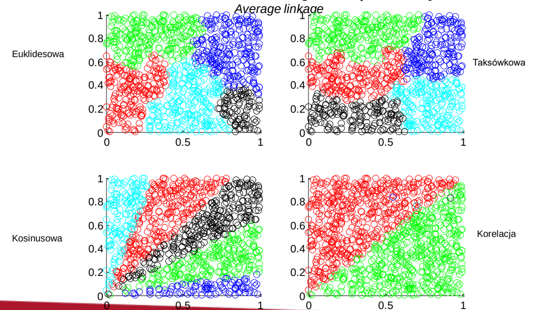
Analiza wektorów losowych *Linkage K*=5



Algorytm *linkage* przypadek szczególny, *K*=3



Różne metryki (K=5)



K-Means założenia

» Bardzo popularny
» Występuje w wielu odmianach
» Nie hierarchiczny
» Zakładamy z góry K - oczekiwaną liczbę skupisk
» Funkcja celu to minimalizacja rozmiarów skupień:

$$q = \min_{\{\boldsymbol{X}_k\}} \sum_{k=1}^{K} \sum_{\substack{\boldsymbol{x}_i, \boldsymbol{x}_j \in \boldsymbol{X}_k \\ \boldsymbol{x}_i \neq \boldsymbol{x}_j}} \delta(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

» Zawsze zbieżny,
» ale nie zawsze do globalnego optimum.

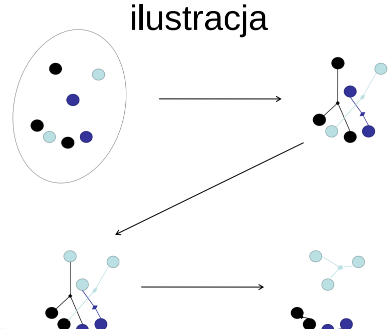
K-Means

1. Wybierz wartość K.
2. Przypisz (losowo) wszystkie wektory $\boldsymbol{x}_i$ do K skupień.
3. Wyznacz średnie $\boldsymbol{\mu}_k$ wszystkich K skupień (*K-Means*).
4. Dla wszystkich wektorów $\boldsymbol{x}_i$ znajdź najbliższe średnie $\boldsymbol{\mu}_k$.

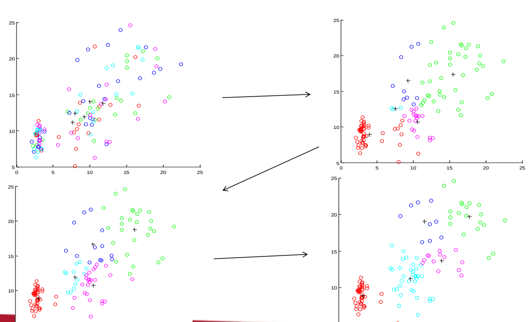
$$\boldsymbol{\mu}^*(\boldsymbol{x}_i) = \arg\min_{\boldsymbol{\mu}_k} \delta(\boldsymbol{x}_i, \boldsymbol{\mu}_k)$$

5. Przypisz wszystkie wektory $\boldsymbol{x}_i$ do skupień o najbliższej średniej.
6. Powtarzaj kroki 3–5 aż do ustabilizowania się pozycji średnich lub braku zmian w obsadzeniu skupień.

Algorytm *K-Means* ilustracja

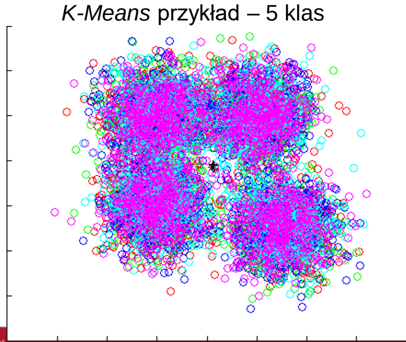


K-Means – przykład K=5

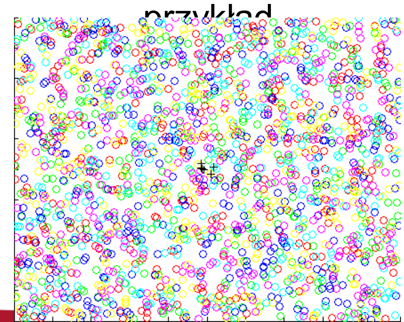


K-Means przykład

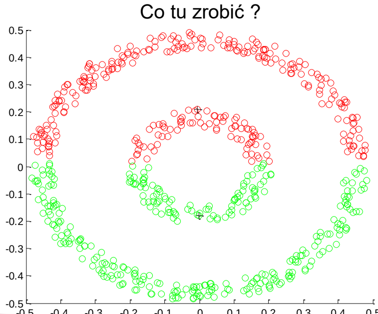
K-Means przykład – 5 klas



K-Means przykład

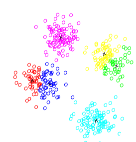


K-Means - problem
Co tu zrobić ?

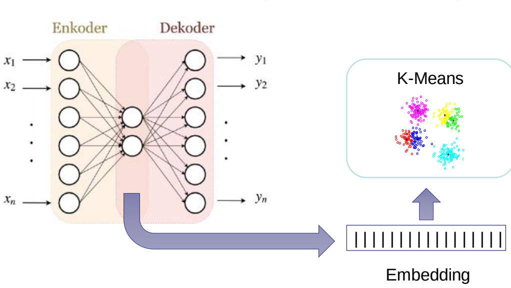


Podsumowanie

» Konieczność wyboru miary podobieństwa
» Ustalone a-prioti *K* - liczba skupień
» Zawsze zbieżne, ale
» Nie zawsze zbieżne do optimum globalnego
» Inicjowanie rozkładów początkowych



Neural clustering w/embeddings



Other methods

» K-Means, MiniBatch K-Means
» MeanShift, VBGMM
» Spectral: PCA, k-PCA
» Agglomerative, hierarchical
» Statistical, GMM
» Neural, embeddings

Clustering evaluation

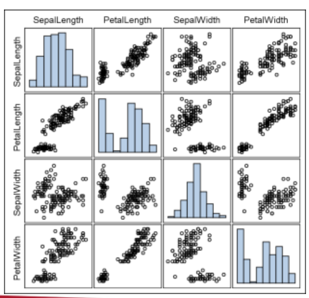
» Ground truth (%)
» Loss
» Adjusted Rand Index
» Mutual information (NMI, AMI)
» Completnes, Homogenity, v-measure
» Silhouette cefficients
» Contingency matrix
» Business metrics

Feature Engineering

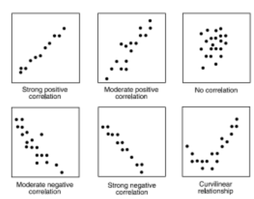
» Using domain knowledge of the data to create features that make machine learning algorithms work.
» Why ?
    Beter features = better results
    ML requires numerical inputs
» How ?

Data Exploration

» Scatter plots
» Histograms
» Distribution type
    Modality
» Ranges
» Correlation
» Quantity
» Understanding



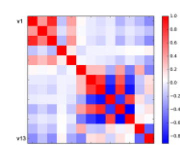
Feature correlation/covariance



» Pearson correlation coefficient

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

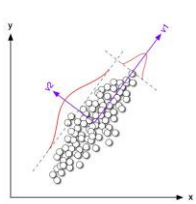
» Correlation/Covariance matrix

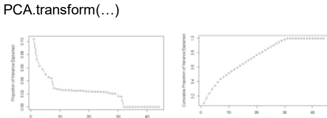


Correlation reduction

» PCA – Principal Component Analysis

```
>>> from sklearn.decomposition import PCA
PCA.fit(…)
PCA.transform(…)
```



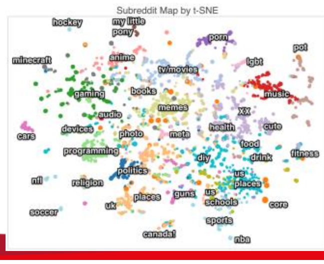


Dimensionality reduction

» Feature Selection (quality, variance)
» Missing Values
» Feature Removal
(statistical tests – eg.Chi2, low variance, high x-corr)
» Feature space transformations
    PCA dimensionality reduction

t-Distributed Stochastic Neighbor Embedding (t-SNE)

» Great 2D/3D visualisation tool
» Dimensionality reduction
» Local similarities
» No global relations.

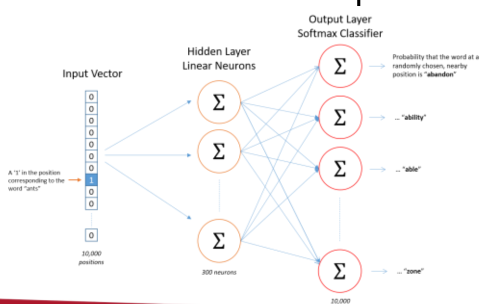


One-hot encoding

» Categorical data encoding
» Non-numerical data encoding
» No order of data categories
» High dimensionality
>>> sklearn.preprocessing.OneHotEncoder

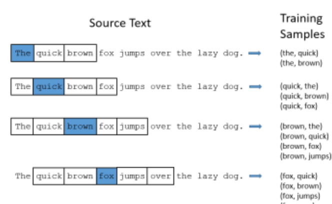
Cat	Dog	Cow	Frog	Fish	Bird	Bee
0	0	0	0	1	0	0

One-Hot as NN input



Word2vec algorithm

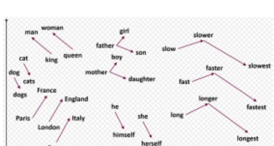
» Train your network using hot-word representation
» Use skip-gram method, or continuous bag of words



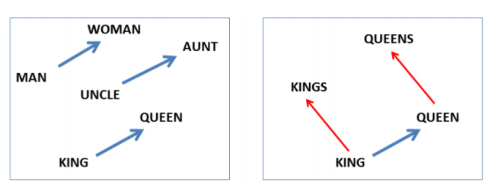
» Use bottleneck feature as embeddings

Word2Vec – word embeddings (Google, Mikolov)

» Converting text into high-dimensional vector of numbers (latent space embedding, 300-500 dim)
» Preserves linguistic and pragmatic information
» Embeddings are easy to manipulate and use in ML algorithms



Word embeddings - latent space

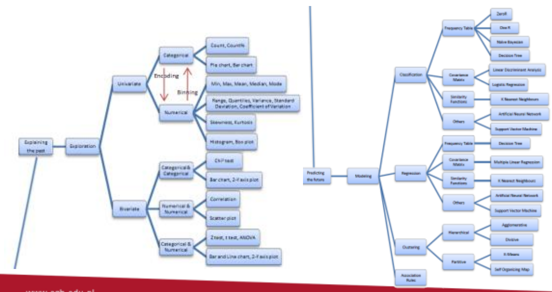


Typical data features

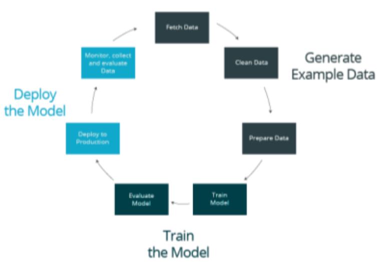
» Sound
» Video
» Image
» NLP
» General time series
» Frames, regresion models, histograms

Data science cheatsheet

http://chem-eng.utoronto.ca/~datamining/dmc/data_mining_map.htm



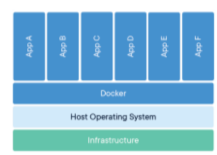
Deploying ML models



Model Deployment

» API for predictor (I/O)

» REST API (JSON)
» WebService (HTTP)
» Docker containers
» Cloud
AWS, Azure, Google
» Big Data



TF Serving

» Scalability,
» Maintanance
» CI/CD
» TF Serving
» TF Lite
Mobile, Embedded

TF Lite

» Embedded, mobile – runtime library with APIs and hardware support

1. Pick model
2. Optimize (quantization, etc.)
3. Convert to *Lite* format (FlatBuffer)
4. Deploy using TF library