MEME19403: Topic 4 Predictive Modelling

Dr Liew How Hui

May 2021

Revision

What have we learned?

- Week 7–9: Data loading, encoding, EDA, cleaning $\rightarrow$ ETL (or ELT)
- Week 9: encoding (one-hot? label?), pre-processing (need scaling?), PCA
- Week 10: Logistic regression (a statistical learning model based on linear boundary), training+testing data

Revision (cont)

What we haven't discussed?

- How do we measure the "trained" model?
  - Output y is categorical: Accuracy (how many we predict right out of total), F1-score, logistic loss, ROC (Receiver Operating Characteristics), ...
  - Output y is numeric: Mean-squared-error $\frac{1}{n}\|y_{exact} - y_{predict}\|^2$, correlation, coefficient of determination $R^2 = 1 - \sum(y_i - \bar{y})^2 / \sum(y_i - f_i)^2$, ...
- If the measured score is low, are there other statistical / machine learning models for classification problem?
- No time for regression problems, time-series model.

Outline

1. Supervised Learning (cont)
   - SVM
   - Naive Bayes
   - kNN
   - Decision Tree
   - Neural Network
2. Applications

Supervised Learning Models

Linear Models:

- Logistic Regression ✓
- Linear SVM
- Naive Bayes

Nonlinear Models:

- k-Nearest Neighbour (kNN)
- Decision Tree
- Neural Network

Classificier:

$$h_D(\mathrm{x}) = \underset{j=1,\dots,K}{\operatorname{argmax}} \mathbb{P}(Y = j | \mathrm{X} = \mathrm{x}). \tag{1}$$

Outline

Linear SVM Theory

$$h_D(\mathrm{x}) = \mathrm{sign}(\mathrm{w}^T\mathrm{x} + b) \in \{+1,\ -1\}$$

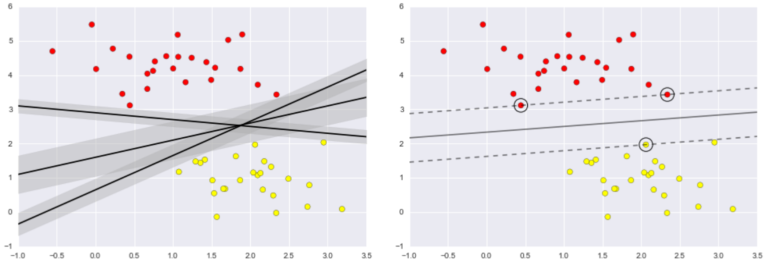
where

$$\min_{\mathrm{w},b} \mathrm{w}^\top \mathrm{w} \quad \text{s.t.} \quad \begin{array}{ll} \forall i,\ y_i(\mathrm{w}^T\mathrm{x}_i + b) & \geq 0 \\ \min_i \left|\mathrm{w}^T\mathrm{x}_i + b\right| & = 1 \end{array}$$

Equivalent formulation:

$$\min_{\mathrm{w},b} \underbrace{\mathrm{w}^T\mathrm{w}}_{l_2-regularizer} + C \sum_{i=1}^{n} \underbrace{\max\left[1 - y_i(\mathrm{w}^T\mathrm{x}_i + b), 0\right]}_{hinge-loss} \quad (2)$$

Support Vector Machine (SVM)



https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html

Support Vector Machine (SVM)

For linear classification (using liblinear as backend):

```
sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', *,
dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True,
intercept_scaling=1, class_weight=None, verbose=0,
random_state=None, max_iter=1000)
```

For linear regression:

```
sklearn.svm.LinearSVR(epsilon=0.0, tol=0.0001, C=1.0,
loss='epsilon_insensitive', fit_intercept=True,
intercept_scaling=1.0, dual=True, verbose=0,
random_state=None, max_iter=1000)
```

Others: Using libsvm as backend: sklearn.svm.SVC, sklearn.svm.SVR (can handle linear and some nonlinear boundaries)

Outline

Naive Bayes Theory

According to Bayes Theorem: For $j \in \{1, \cdots, K\}$,

$$\mathbb{P}(Y = j|\mathrm{X} = \mathrm{x}) = \frac{\mathbb{P}(\mathrm{X} = \mathrm{x}|Y = j)\mathbb{P}(Y = j)}{\mathbb{P}(\mathrm{X} = \mathrm{x})}, \tag{3}$$

the generative classifier is

$$\begin{aligned} h_D(\mathrm{x}) &= \operatorname*{argmax}_{j\in\{1,\dots,K\}} \mathbb{P}(Y = j|\mathrm{X} = \mathrm{x}) \\ &= \operatorname*{argmax}_{j\in\{1,\dots,K\}} \frac{\mathbb{P}(\mathrm{X} = \mathrm{x}|Y = j)\mathbb{P}(Y = j)}{\mathbb{P}(\mathrm{X} = \mathrm{x})} \\ &= \operatorname*{argmax}_{j\in\{1,\dots,K\}} \mathbb{P}(\mathrm{X} = \mathrm{x}|Y = j)\mathbb{P}(Y = j) \\ &= \operatorname*{argmax}_{j\in\{1,\dots,K\}} [\ln \mathbb{P}(\mathrm{X} = \mathrm{x}|Y = j) + \ln \mathbb{P}(Y = j)] \end{aligned} \tag{4}$$

Theory (cont)

where

- $\mathbb{P}(Y = j|\mathrm{X} = \mathrm{x})$, the *posterior probability*, is the probability that the new observation belongs to the jth class, given the predictor value for that observation;
- $\mathbb{P}(\mathrm{X} = \mathrm{x}) = \sum_{k=1}^{K} \mathbb{P}(\mathrm{X} = \mathrm{x}|Y = k)\mathbb{P}(Y = k)$ is regarded as a constant w.r.t the response class j;
- $\mathbb{P}(\mathrm{X} = \mathrm{x}|Y = j)$ is the *likelihood function*, a density function of X for an observation that comes from the jth class. It is relatively large if there is a high probability that an observation in the jth class has $\mathrm{X} \approx \mathrm{x}$;
- $\mathbb{P}(Y = j)$ is the *prior probability*, i.e. the probability that a randomly chosen observation comes from the jth class of response variable Y.

Naive Bayes and Laplace Correction

A naïve Bayes classifier (NB) (`https://en.wikipedia.org/wiki/Naive_Bayes_classifier`) is a simple probabilistic classifier (3) based on applying Bayes' theorem with **strong (naive) independence assumptions**:

$$
\begin{aligned}
&\mathbb{P}(\mathrm{X} = \mathrm{x}|Y = j) \\
&= \mathbb{P}(X_1 = x_1, X_2 = x_2, ..., X_p = x_p|Y = j) \\
&= \mathbb{P}(X_1 = x_1|Y = j) \times \cdots \times \mathbb{P}(X_p = x_p|Y = j) \\
&= \prod_{i=1}^{p} \mathbb{P}(X_i = x_i|Y = j).
\end{aligned}
$$

Naive Bayes Classificiers and Laplace Correction

Therefore, (4) becomes

$$\begin{aligned} h_D(\mathrm{x}) &= \operatorname*{argmax}_{j\in\{1,\dots,K\}} \mathbb{P}(Y=j)\prod_{i=1}^{p}\mathbb{P}(X_i=x_i|Y=j) \\ &= \operatorname*{argmax}_{j\in\{1,\dots,K\}} \ln\mathbb{P}(Y=j)+\left[\sum_{i=1}^{p}\ln\mathbb{P}(X_i=x_i|Y=j)\right]. \end{aligned} \tag{5}$$

Naïve Bayes Classifiers (cont)

The estimate of the prior distribution $\mathbb{P}(Y = j)$ using MLE is as follows:

$$\mathbb{P}\widehat{(Y = j)} = \frac{\#\{i \ : \ y_i = j\}}{n}. \tag{6}$$

However, it is possible to choose

$$\mathbb{P}(Y = j) = \frac{1}{K}$$

if we know the outcome should be uniformly distributed.

Naïve Bayes Classifiers (cont)

The features X_i can be categorical or numeric:

- One X_i is categorical — Categorical NB ✓
- All X_i are binary — Bernoulli NB
- All X_i are integral — Multinomial NB & Complement NB
- One X_i is numeric — Gaussian NB

Categorical NB

The feature X_i

- is categorical;
- takes on M_i possible values: $\{c_1^{(i)}, \cdots, c_{M_i}^{(i)}\} =: \mathscr{C}_i$

The conditional distribution **without Laplace smoothing** is

$$\mathbb{P}(X_i = c | Y = j) = \frac{n_{X_i=c \ \& \ Y=j}}{n_{Y=j}}, \quad c \in \mathscr{C}_i. \tag{7}$$

Categorical NB (cont)

The conditional distribution **with Laplace smoothing** is

$$\mathbb{P}(X_i = c | Y = j;\ \alpha) = \frac{n_{X_i=c\ \&\ Y=j} + \alpha}{n_{Y=j} + \alpha d_i} \tag{8}$$

where α is a **smoothing parameter** and d_i is the number of available categories of feature X_i defined above. It has the following form:

```
from sklearn.naive_bayes import CategoricalNB
CategoricalNB(alpha=1.0, fit_prior=True,
  class_prior=None)
```

Categorical NB (cont)

Example

Table Q3(d) shows a data set containing 7 observations with 3 categorical predictors, X_1, X_2 and X_3.

Observation	X_1	X_2	X_3	Y
1	C	No	0	Positive
2	A	Yes	1	Positive
3	B	Yes	0	Negative
4	B	Yes	0	Negative
5	A	No	1	Positive
6	C	No	1	Negative
7	B	Yes	1	Positive

Table Q3(d)

Without Laplace smoothing, predict the response, Y for an observation with $X_1 = \text{B}$, $X_2 = \text{Yes}$ and $X_3 = 1$ using Naïve Bayes approach.

Categorical NB (cont)

Solution: Let '+' denote 'Positive' and '-' denote 'Negative'.

prior, $\mathbb{P}(Y)$	$\mathbb{P}(X_1\|Y)$	$\mathbb{P}(X_2\|Y)$	$\mathbb{P}(X_3\|Y)$	prop, Π	$\hat{Y}$
$\mathbb{P}(+) = \frac{4}{7}$	$\mathbb{P}(\text{B}\|+) = \frac{1}{4}$	$\mathbb{P}(\text{Yes}\|+) = \frac{1}{2}$	$\mathbb{P}(1\|+) = \frac{3}{4}$	0.0536	
$\mathbb{P}(-) = \frac{3}{7}$	$\mathbb{P}(\text{B}\|-) = \frac{2}{3}$	$\mathbb{P}(\text{Yes}\|-) = \frac{2}{3}$	$\mathbb{P}(1\|-) = \frac{1}{3}$	0.0635	$\checkmark, -$

Since $\mathbb{P}(Y = -|X) > \mathbb{P}(Y = +|X)$, Y has higher probability to be "Negative".

Gaussian Naïve Bayes

For continuous inputs X_i in (5), it is assume that X_i is Gaussian:

$$\mathbb{P}(X_i = x | Y = j) = \frac{1}{\sigma_j\sqrt{2\pi}} \exp(-\frac{(x-\mu_j)^2}{2\sigma_j^2}). \tag{9}$$

It is defined by a mean and standard deviation specific to X_i and j. The estimations of the mean and standard deviation are

$$\mu_j = \mathbb{E}[X_i | Y = j], \quad \sigma_j^2 = \mathbb{E}[(X_i - \mu_j)^2 | Y = j]. \tag{10}$$

Gaussian Naïve Bayes (cont)

The maximum likelihood estimator for (10) are

$$\begin{aligned}\widehat{\mu}_j &= \frac{1}{\sum_{i=1}^n I(y_i = j)} \sum_{k=1}^n X_{ki} I(Y = j), \\ s_j &= \frac{1}{(\sum_{i=1}^n I(y_i = j)) - 1} \sum_{k=1}^n (X_{ki} - \widehat{\mu}_j)^2 I(y_i = j).\end{aligned} \tag{11}$$

Gaussian Naïve Bayes (Classifier) is available in Python as `GaussianNB` of the form:

```
from sklearn.naive_bayes import GaussianNB
GaussianNB(priors=None, var_smoothing=1e-09)
```

Gaussian Naïve Bayes (cont)

Example

The table below shows the data collected for predicting whether a customer will default on the credit card or not:

customer	balance	student	Default
1	500	No	N
2	1980	Yes	Y
3	60	No	N
4	2810	Yes	Y
5	1400	No	N
6	300	No	N
7	2000	Yes	Y
8	940	No	N
9	1630	No	Y
10	2170	Yes	Y

Gaussian Naïve Bayes (cont)

(a) Compute the probability density of customer with balance 2080, given Default=Y.
(b) Compute the probability of customer who is a student, given Default=Y.
(c) Calculate the "probability density" of default for a student customer with balance 2080 by using the Naïve Bayes assumption.

Gaussian Naïve Bayes (cont)

(a) **Solution**:

$$\begin{aligned}&\mathbb{P}(\texttt{balance} = 2080 \mid \texttt{Default} = Y)\\ =&\frac{1}{s_Y\sqrt{2\pi}}\exp(-\frac{(2080-\mu_Y)^2}{2s_Y^2}) = 0.0009162\end{aligned}$$

where $\mu_Y = \frac{1980+2810+2000+1630+2170}{5} = 2118$;
$s_Y = 433.7857$

(b) **Solution**: $\mathbb{P}(\texttt{student} = Yes \mid \texttt{Default} = Y) = \dfrac{4}{5}$

Gaussian Naïve Bayes (cont)

(c) **Solution**:

$\mathbb{P}(\texttt{student} = \mathit{Yes} \mid \texttt{Default} = N) = \frac{0}{5} = 0$

$$\begin{aligned}
&\mathbb{P}(\texttt{Default} = Y \mid \texttt{balance} = 2080,\ \texttt{student} = \mathit{Yes}) \\
=&\frac{\mathbb{P}(\texttt{balance} = 2080,\ \texttt{student} = \mathit{Yes} \mid \texttt{Default} = Y)\mathbb{P}(\texttt{Default} = \texttt{Y})}{\mathbb{P}(\texttt{balance} = 2080,\ \texttt{student} = \mathit{Yes}) =: \mathbb{P}(\ldots)} \\
=&\frac{\mathbb{P}(\texttt{balance} = 2080,\ \texttt{student} = \mathit{Yes} \mid \texttt{Default} = Y)\mathbb{P}(\texttt{Default} = \texttt{Y})}{\mathbb{P}(\ldots \mid \texttt{Default} = Y)\mathbb{P}(\texttt{Default} = \texttt{Y}) + \mathbb{P}(\ldots \mid \texttt{Default} = N)\mathbb{P}(\texttt{Default} = \texttt{N})} \\
=&\frac{0.0009162 \times \frac{4}{5} \times \frac{5}{10}}{0.0009162 \times \frac{4}{5} \times \frac{5}{10} + \mathbb{P}(\texttt{balance} = 2080 | \texttt{Default} = N) \times 0 \times \frac{5}{10}} = 1
\end{aligned}$$

Outline

kNN Theory

The *k-nearest neighbours* (kNN) algorithm is a non-parametric method used for classification and regression.

The assumption of kNN is “similar inputs have similar outputs”. Based on this assumption, a test input x should be assigned the most common label amongst its k most similar training inputs.

kNN Theory (cont)

Given a positive integer k and an input x, the kNN algorithm first identifies the k points in the training data (x_i, y_i) that are "closest" to x , represented by $N(\mathrm{x})$.

- For kNN classifier, the prediction is

$$h(\mathrm{x}) = \mathrm{mode}(\{y'' : (\mathrm{x}'', y'') \in N(\mathrm{x})\}),$$

$$\mathbb{P}(Y = j|\mathrm{X} = \mathrm{x}) = \frac{1}{k} \sum_{\mathrm{x}_i \in N(\mathrm{x})} I(y_i = j).$$

- For kNN regressor, the prediction is

$$h(\mathrm{x}) = \frac{1}{k} \sum_{(\mathrm{x}'', y'') \in N(\mathrm{x})} y''.$$

kNN in Python

For classification problem:

```
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *,
weights='uniform', algorithm='auto', leaf_size=30, p=2,
metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

where metric is either a string (default='minkowski') or callable (distance function). Refer to `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html` for details.

For regression problem:

```
sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, *,
weights='uniform', algorithm='auto', leaf_size=30, p=2,
metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

Example of kNN Classifier

A sport school would like to group their new enrolled students into 2 groups, as according to the existing students' weight and height. The weight and height of 7 existing students with group are shown in the table below.

Student	Weight (kg)	Height (cm)	Group
A	29	118	A
B	53	137	B
C	38	127	B
D	49	135	B
E	28	111	A
F	24	111	A
G	30	121	A

Example (cont)

(a) Perform and use $k = 3$-NN method (with Euclidean distance) to predict which group the following students will be grouped into, based on **cut-off of 0.7** on group A.

Student	Weight (kg)	Height (cm)
H	35	120
I	47	131
J	22	115
K	38	119
L	31	136

Example (cont)

(b) The actual groups of the students are {A, B, A, B, B} for students {H, I, J, K, L} respectively. Construct a confusion matrix and calculate the accuracy measurements for a cut-off of 0.5 and a cut-off of 0.7.

(c) Write a Python script to solve the above problem.

Outline

1. Supervised Learning (cont)
   - SVM
   - Naive Bayes
   - kNN
   - Decision Tree
   - Neural Network
2. Applications

Decision Tree

According to `https://en.wikipedia.org/wiki/Decision_tree`, a decision tree is a "flowchart"-like structure in which each *internal node* (the decision node) represents a "test" on a "predictor" or an "attribute", each branch represents the outcome of the test, and each **leaf node** represents a class label (decision taken after computing all attributes).

Decision Tree (cont)

The construction of a tree is equivalent to the "splitting" of the data S. The most popular way of splitting a data is to "split" based on a selected predictor. Since a predictor can be categorical or numerical, the **splitting criteria** differ as follows:

predictors	qualitative (categorical)	quantitative (numerical)
splits	disjoint subsets of the predictor categories	disjoint ranges of the predictors value

Decision Tree (cont)

The goal in splitting the data to build a decision tree is to make sure it is “maximally compact” and “only has pure leaves”.

A consistent tree is possible if no two input vectors have identical features but different labels.

“Impurity” measures are used to choose the “best” predictor to perform “splitting” for the “decision-tree construction algorithm”.

Decision Tree (cont)

The “impurity” measures generally measure the homogeneity of the target variable within the subsets. According to `https://en.wikipedia.org/wiki/Decision_tree_learning`, we have

- Gini Impurity Index (default in ‘rpart’)
- Entropy and Information Gain
- Gain Ratio
- Deviance (?)
- Variance Reduction (?)

Decision Tree (cont)

For classification problem:

```
sklearn.tree.DecisionTreeClassifier(*, criterion='gini',
splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
class_weight=None, presort='deprecated', ccp_alpha=0.0)
```

Supported criteria: “gini” and “entropy” for the information gain.

For regression problem:

```
sklearn.tree.DecisionTreeRegressor(*, criterion='mse',
...almost the same...)
```

Supported criteria are “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node, “friedman_mse”, which uses mean squared error with Friedman’s improvement score for potential splits, and “mae” for the mean absolute error, which minimizes the L1 loss using the median of each terminal node.

Decision Tree (cont)

Example (Play Tennis Data)

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree (cont)

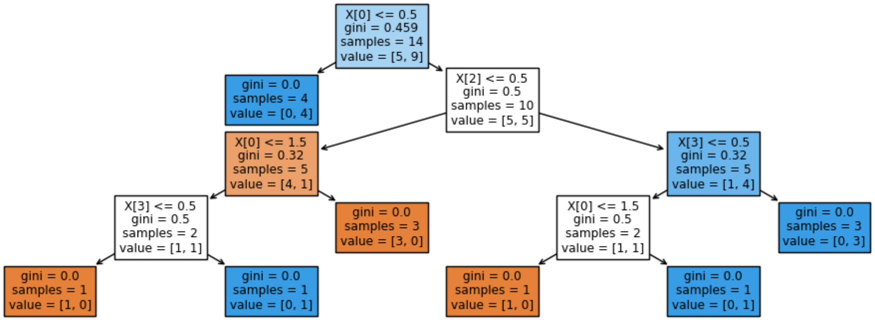
```
import pandas as pd, matplotlib.pylab as plt
from sklearn.preprocessing import LabelEncoder

# https://camlcity.org/gerd/invtables/blob/17dfb2519e19fc692f94442f72
df = pd.read_csv('playtennis.csv')

lb = LabelEncoder()
df['outlook_'] = lb.fit_transform(df['outlook'])
df['temp_'] = lb.fit_transform(df['temperature'] )
df['humidity_'] = lb.fit_transform(df['humidity'] )
df['windy_'] = lb.fit_transform(df['wind'] )
df['play_'] = lb.fit_transform(df['playtennis'] )
X_train = df.iloc[:,6:10]
y_train = df.iloc[:,10]

from sklearn.tree import DecisionTreeClassifier, plot_tree
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
fig=plt.figure(figsize=(10,4))
axes=fig.add_axes([0,0,1,0.8])
plot_tree(model, filled=True, ax=axes)
plt.show()
```

Decision Tree (cont)



Outline

Neural Network

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function

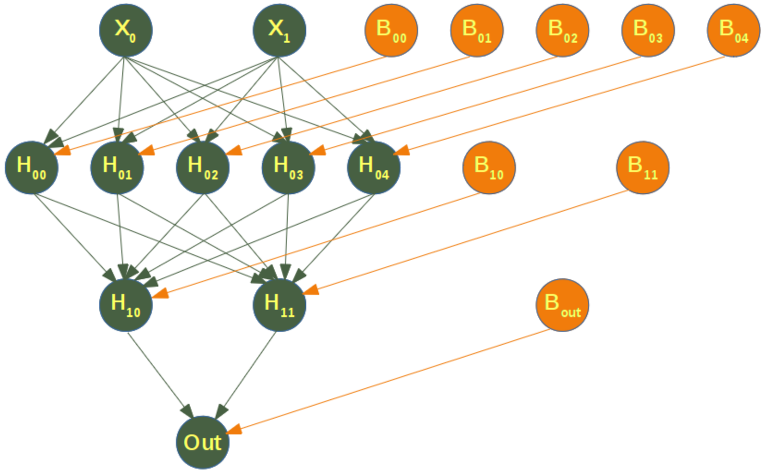
$$f(\cdot) : R^p \to R^K$$

by training on a dataset, where

- p is the number of dimensions for input: $X = x_1, x_2, ..., x_p$
- K is the number of dimensions for output: target y
- In between input & output — hidden layers

Neural Network (cont)

Layer Link: $x_i^{(\ell+1)} = S(w_{i,1}^{(\ell)} x_1^{(\ell)} + \ldots + w_{i,n_\ell}^{(\ell)} x_{n_\ell}^{(\ell)} + B_i^{(\ell)})$



Source: `https://www.python-course.eu/neural_networks_with_scikit.php`

Neural Network (cont)

Software Implementations:

```
sklearn.neural_network.MLPClassifier(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, max_fun=15000)
```

- S = `activation` = identity, logistic, tanh, relu
- solver = lbfgs, sgd, adam (default)
  - lbfgs is an optimizer in the family of quasi-Newton methods. For small datasets, 'lbfgs' can converge faster and perform better.
  - sgd refers to stochastic gradient descent.
  - adam refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba. It works pretty well on relatively large datasets.

Outline

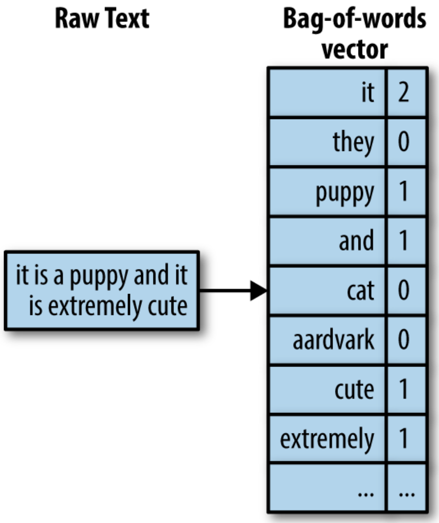
Remark on Real-World Applications

(Time-independent) Structured Data from Database system can be used as a training data after cleaning. But **unstructured data** needs more complex maths to transform into structure data:

- Text Mining: Bag of Words, (Bag of) N-grams, TF-IDF (Term Frequency, Inverse Document Frequency)
- Image Recognition using CNN architectures: Inception v3, Xception, DenseNet, LeNet-5, AlexNet, VGG16, ResNet, SqueezeNet, EfficientNet, MobileNet

Real-World Applications (cont)

Bag of Words:



Real-World Applications (cont)

TF-IDF is a scoring of how rare the word is across documents. It is used in scenario where highly recurring words may not contain as much informational content as the domain specific words. For example, words like “the” that are frequent across all documents therefore need to be less weighted. The TF-IDF score highlights words that are distinct (contain useful information) in a given document.

Real-World Applications (cont)

Text mining is a special type of data mining in which the knowledge discovery is applied to text. The source of text can be from RDBMS, XML, JSON or word documents, PDF documents, etc. There are clustering-based methods, supervised learning methods (bag of words vector representation), probabilistic methods (probabilistic latent semantic analysis, Latent Dirichlet Allocation), sentiment analysis, etc.

Closely related to text mining is the information retrieval. Information retrieval focuses on the facilitation of information access (named entity recognition and relation extraction) rather than analysing information and finding hidden patterns which are the focuses of text mining.

Real-World Applications (cont)

NLP (natural language processing techniques) transforms unstructured text to some sort of structured data using techniques such as tokenisation, filtering stop-words and articles, lemmatisation (morphological analysis of words), etc. There are used in the analysis of Blogs, YouTube comments, Facebook messages, Discus comments, etc.

Text clustering is the task of segmenting a collection of documents into partitions where documents in the same group (cluster) are more similar to each other than those in other clusters.

Real-World Applications (cont)

CNNs (convolutional neural networks) have become the go-to method for solving any image data challenge. Their use is being extended to video analytics as well. Any data that has spatial relationships is ripe for applying CNN.

Let's take a 6×6 grayscale image (i.e. only one channel):

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Real-World Applications (cont)

The type of **filter** are used to detect the vertical or horizontal edges. We can use the following filters to detect different edges:

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

$$\begin{array}{|c|c|c|c|c|c|}\hline 3 & 0 & 1 & 2 & 7 & 4 \\ \hline 1 & 5 & 8 & 9 & 3 & 1 \\ \hline 2 & 7 & 2 & 5 & 1 & 3 \\ \hline 0 & 1 & 3 & 1 & 7 & 8 \\ \hline 4 & 2 & 1 & 6 & 2 & 8 \\ \hline 2 & 4 & 5 & 2 & 3 & 9 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline -5 & -4 & 0 & 8 \\ \hline -10 & -2 & 2 & 3 \\ \hline 0 & -2 & -4 & -7 \\ \hline -3 & -2 & -3 & -16 \\ \hline \end{array}$$

Note: $-5 = 3(1) + 0(0) + 1(-1) + 1(1) + 5(0) + 8(-1) + 2(1) + 7(0) + 2(-1)$

Real-World Applications (cont)

Some of the commonly used filters are:

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

The Sobel filter puts a little bit more weight on the central pixels. Instead of using these filters, we can create our own as well and treat them as a parameter which the model will learn using backpropagation.

Real-World Applications (cont)

Consider an $n \times n$ image with 3 colours. Convolve it with 10 filters of size 3×3, and take the stride as 1 and no padding. This will give us an output of

$$(n+1-3) \times (n+1-3) \times 10$$

We convolve this output further in multiple levels to get something smaller. Finally, let's say that we obtain something like

$$7 \times 7 \times 40 = 1960$$

which will be unrolled into a large vector, and be passed to a classifier that will make predictions. This is roughly how a convolutional network works.

Real-World Applications (cont)

There are primarily two major advantages of using convolutional layers over using just fully connected layers:

- Parameter sharing (using filters)
- Sparsity of connections