Big Data Mining

巨量資料探勘

機器學習與深度學習

(Machine Learning and Deep Learning)

1082DM10
MI4 (M2244) (2744)
Tue 3, 4 (10:10-12:00) (B218)

Min-Yuh Day
戴敏育
Associate Professor
副教授

Dept. of Information Management, Tamkang University

http://mail.tku.edu.tw/myday/
2020-05-19
<table>
<thead>
<tr>
<th>週次 (Week)</th>
<th>日期 (Date)</th>
<th>內容 (Subject/Topics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2020/03/03</td>
<td>巨量資料探勘課程介紹 (Course Orientation for Big Data Mining)</td>
</tr>
<tr>
<td>2</td>
<td>2020/03/10</td>
<td>AI人工智慧與大數據分析 (Artificial Intelligence and Big Data Analytics)</td>
</tr>
<tr>
<td>3</td>
<td>2020/03/17</td>
<td>分群分析 (Cluster Analysis)</td>
</tr>
<tr>
<td>4</td>
<td>2020/03/24</td>
<td>個案分析與實作一 (SAS EM 分群分析)：Case Study 1 (Cluster Analysis - K-Means using SAS EM)</td>
</tr>
<tr>
<td>5</td>
<td>2020/03/31</td>
<td>關連分析 (Association Analysis)</td>
</tr>
<tr>
<td>6</td>
<td>2020/04/07</td>
<td>個案分析與實作二 (SAS EM 關連分析)：Case Study 2 (Association Analysis using SAS EM)</td>
</tr>
<tr>
<td>7</td>
<td>2020/04/14</td>
<td>分類與預測 (Classification and Prediction)</td>
</tr>
<tr>
<td>8</td>
<td>2020/04/21</td>
<td>期中報告 (Midterm Project Presentation)</td>
</tr>
</tbody>
</table>
## 課程大綱 (Syllabus)

<table>
<thead>
<tr>
<th>週次 (Week)</th>
<th>日期 (Date)</th>
<th>內容 (Subject/Topics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2020/04/28</td>
<td>期中考試週</td>
</tr>
</tbody>
</table>
| 10         | 2020/05/05 | 個案分析與實作三 (SAS EM 決策樹、模型評估)：  
Case Study 3 (Decision Tree, Model Evaluation using SAS EM) |
| 11         | 2020/05/12 | 個案分析與實作四 (SAS EM 迴歸分析、類神經網路)：  
Case Study 4 (Regression Analysis,  
Artificial Neural Network using SAS EM) |
| 12         | 2020/05/19 | 機器學習與深度學習 (Machine Learning and Deep Learning) |
| 13         | 2020/05/26 | 期末報告 (Final Project Presentation) |
| 14         | 2020/06/02 | 畢業考試週           |
| 15         | 2020/06/09 | 教師彈性補充教學     |
Machine Learning and Deep Learning
Outline

• Machine Learning
• Deep Learning
3 Machine Learning Algorithms

Machine Learning Models

- Deep Learning
- Association rules
- Decision tree
- Clustering
- Bayesian
- Kernel
- Ensemble
- Dimensionality reduction
- Regression Analysis
- Instance based

Source: Sunila Gollapudi (2016), Practical Machine Learning, Packt Publishing
Machine Learning (ML) / Deep Learning (DL)

Supervised Learning
- Decision Tree Classifiers
- Linear Classifiers
- Rule-based Classifiers
- Probabilistic Classifiers

Deep Learning (DL)
- Support Vector Machine (SVM)
- Neural Network (NN)
- Naïve Bayes (NB)
- Bayesian Network (BN)
- Maximum Entropy (ME)

Unsupervised Learning
- Reinforcement Learning

Scikit-Learn
Machine Learning in Python
Scikit-Learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification
- Identifying to which category an object belongs to.
  - **Applications**: Spam detection, Image recognition.
  - **Algorithms**: SVM, nearest neighbors, random forest, ...

Regression
- Predicting a continuous-valued attribute associated with an object.
  - **Applications**: Drug response, Stock prices.
  - **Algorithms**: SVR, ridge regression, Lasso, ...

Clustering
- Automatic grouping of similar objects into sets.
  - **Applications**: Customer segmentation, Grouping experiment outcomes
  - **Algorithms**: k-Means, spectral clustering, mean-shift, ...

Dimensionality reduction
- Reducing the number of random variables to consider.
  - **Applications**: Visualization, Increased efficiency
  - **Algorithms**: PCA, feature selection, non-negative matrix factorization.

Model selection
- Comparing, validating and choosing parameters and models.
  - **Goal**: Improved accuracy via parameter tuning
  - **Modules**: grid search, cross validation, metrics.

Preprocessing
- Feature extraction and normalization.
  - **Application**: Transforming input data such as text for use with machine learning algorithms.
  - **Modules**: preprocessing, feature extraction.

Source: [http://scikit-learn.org/](http://scikit-learn.org/)
Scikit-Learn Machine Learning Map

Scikit-Learn Machine Learning Map

START

get more data

>50 samples

predicting a category

<100K samples

predicting a quantity

regression

SGD Regressor

Lasso
ElasticNet

few features should be important

SVR(kernel='rbf')

Ensemble Regressors

RidgeRegression

SVR(kernel='linear')

scikit-learn algorithm cheat-sheet

Iris flower data set

setosa    versicolor    virginica

Source: https://en.wikipedia.org/wiki/Iris_flower_data_set
<table>
<thead>
<tr>
<th>Sepal Length (cm)</th>
<th>Sepal Width (cm)</th>
<th>Petal Length (cm)</th>
<th>Petal Width (cm)</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.7</td>
<td>1.5</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.8</td>
<td>3.4</td>
<td>1.6</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.8</td>
<td>3.0</td>
<td>1.4</td>
<td>0.1</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.3</td>
<td>3.0</td>
<td>1.1</td>
<td>0.1</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.8</td>
<td>4.0</td>
<td>1.2</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.7</td>
<td>4.4</td>
<td>1.5</td>
<td>0.4</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.3</td>
<td>0.4</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.3</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.7</td>
<td>3.8</td>
<td>1.7</td>
<td>0.3</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.1</td>
<td>3.8</td>
<td>1.5</td>
<td>0.3</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.4</td>
<td>1.7</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.1</td>
<td>3.7</td>
<td>1.5</td>
<td>0.4</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.6</td>
<td>1.0</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.1</td>
<td>3.3</td>
<td>1.7</td>
<td>0.5</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.8</td>
<td>3.4</td>
<td>1.9</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.0</td>
<td>1.6</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.6</td>
<td>0.4</td>
<td>Iris-setosa</td>
</tr>
</tbody>
</table>
Iris Data Visualization

Source: https://seaborn.pydata.org/generated/seaborn.pairplot.html
import seaborn as sns
iris = sns.load_dataset("iris")
g = sns.pairplot(iris, hue="species")

Source: https://seaborn.pydata.org/generated/seaborn.pairplot.html
import seaborn as sns
sns.set(style="ticks", color_codes=True)
iris = sns.load_dataset("iris")
g = sns.pairplot(iris, hue="species")

Source: https://seaborn.pydata.org/generated/seaborn.pairplot.html
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import import scatter_matrix

# Load dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
df = pd.read_csv(url, names=names)

print(df.head(10))
print(df.tail(10))
print(df.describe())
print(df.info())
print(df.shape)
print(df.groupby('class').size())

plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()

df.hist()
plt.show()

scatter_matrix(df)
plt.show()
sns.pairplot(df, hue="class", size=2)

import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix

# Import Libraries
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
print('imported')

imported
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']

df = pd.read_csv(url, names=names)
print(df.head(10))
```python
print(df.describe())
```

<table>
<thead>
<tr>
<th></th>
<th>sepal-length</th>
<th>sepal-width</th>
<th>petal-length</th>
<th>petal-width</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>150.000000</td>
<td>150.000000</td>
<td>150.000000</td>
<td>150.000000</td>
</tr>
<tr>
<td>mean</td>
<td>5.843333</td>
<td>3.054000</td>
<td>3.758667</td>
<td>1.198667</td>
</tr>
<tr>
<td>std</td>
<td>0.828066</td>
<td>0.433594</td>
<td>1.764420</td>
<td>0.763161</td>
</tr>
<tr>
<td>min</td>
<td>4.300000</td>
<td>2.000000</td>
<td>1.000000</td>
<td>0.100000</td>
</tr>
<tr>
<td>25%</td>
<td>5.100000</td>
<td>2.800000</td>
<td>1.600000</td>
<td>0.300000</td>
</tr>
<tr>
<td>50%</td>
<td>5.800000</td>
<td>3.000000</td>
<td>4.350000</td>
<td>1.300000</td>
</tr>
<tr>
<td>75%</td>
<td>6.400000</td>
<td>3.300000</td>
<td>5.100000</td>
<td>1.800000</td>
</tr>
<tr>
<td>max</td>
<td>7.900000</td>
<td>4.400000</td>
<td>6.900000</td>
<td>2.500000</td>
</tr>
</tbody>
</table>
```python
print(df.tail(10).)
```

<table>
<thead>
<tr>
<th>sepal-length</th>
<th>sepal-width</th>
<th>petal-length</th>
<th>petal-width</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>140</td>
<td>6.7</td>
<td>3.1</td>
<td>5.6</td>
<td>2.4 Iris-virginica</td>
</tr>
<tr>
<td>141</td>
<td>6.9</td>
<td>3.1</td>
<td>5.1</td>
<td>2.3 Iris-virginica</td>
</tr>
<tr>
<td>142</td>
<td>5.8</td>
<td>2.7</td>
<td>5.1</td>
<td>1.9 Iris-virginica</td>
</tr>
<tr>
<td>143</td>
<td>6.8</td>
<td>3.2</td>
<td>5.9</td>
<td>2.3 Iris-virginica</td>
</tr>
<tr>
<td>144</td>
<td>6.7</td>
<td>3.3</td>
<td>5.7</td>
<td>2.5 Iris-virginica</td>
</tr>
<tr>
<td>145</td>
<td>6.7</td>
<td>3.0</td>
<td>5.2</td>
<td>2.3 Iris-virginica</td>
</tr>
<tr>
<td>146</td>
<td>6.3</td>
<td>2.5</td>
<td>5.0</td>
<td>1.9 Iris-virginica</td>
</tr>
<tr>
<td>147</td>
<td>6.5</td>
<td>3.0</td>
<td>5.2</td>
<td>2.0 Iris-virginica</td>
</tr>
<tr>
<td>148</td>
<td>6.2</td>
<td>3.4</td>
<td>5.4</td>
<td>2.3 Iris-virginica</td>
</tr>
<tr>
<td>149</td>
<td>5.9</td>
<td>3.0</td>
<td>5.1</td>
<td>1.8 Iris-virginica</td>
</tr>
</tbody>
</table>
```
print(df.info())
print(df.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal-length  150 non-null float64
sepal-width   150 non-null float64
petal-length  150 non-null float64
petal-width   150 non-null float64
class         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
None

print(df.shape)
```

```
(150, 5)
```
df.groupby('class').size()

```
print(df.groupby('class').size())
```

class
Iris-setosa    50
Iris-versicolor 50
Iris-virginica 50
dtype: int64
```python
plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```
df.hist()
plt.show()
scatter_matrix(df)
plt.show()
sns.pairplot(df, hue="class", size=2)
Machine Learning
Supervised Learning
Classification
and
Prediction
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix

# Import sklearn
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

print("Imported")

data = pd.read_csv(url, names=names)
print(data.head(10))
print(data.tail(10))
print(data.describe())
print(data.info())
print(data.shape)
print(data.groupby('class').size())
plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
df.hist()
plt.show()}
# Import sklearn
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
print("Imported")
# Load dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
df = pd.read_csv(url, names=names)

print(df.head(10))
print(df.tail(10))
print(df.describe())
print(df.info())
print(df.shape)
print(df.groupby('class').size())

plt.rcParams["figure.figsize"] = (10, 8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()

df.hist()
plt.show()

scatter_matrix(df)
plt.show()

sns.pairplot(df, hue="class", size=2).
```python
# Load dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
df = pd.read_csv(url, names=names)

print(df.head(10))
print(df.tail(10))
print(df.describe())
print(df.info())
print(df.shape)
print(df.groupby('class').size())

plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()

df.hist()
plt.show()
scatter_matrix(df)
plt.show()
sns.pairplot(df, hue="class", size=2).
```

<table>
<thead>
<tr>
<th>sepal-length</th>
<th>sepal-width</th>
<th>petal-length</th>
<th>petal-width</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.3</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.4</td>
<td>2.9</td>
<td>1.4</td>
<td>0.2</td>
<td>Iris-setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td>Iris-setosa</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sepal-length</th>
<th>sepal-width</th>
<th>petal-length</th>
<th>petal-width</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7</td>
<td>3.1</td>
<td>5.6</td>
<td>2.4</td>
<td>Iris-virginica</td>
</tr>
<tr>
<td>6.9</td>
<td>3.1</td>
<td>5.1</td>
<td>2.3</td>
<td>Iris-virginica</td>
</tr>
<tr>
<td>5.8</td>
<td>2.7</td>
<td>5.1</td>
<td>1.9</td>
<td>Iris-virginica</td>
</tr>
</tbody>
</table>

[Link to Google Colab Notebook](https://colab.research.google.com/drive/1QE7fR2OxHiQ0_p6I1nnZDlFF354Nf_Lw)
```python
df.corr()
```

<table>
<thead>
<tr>
<th></th>
<th>sepal-length</th>
<th>sepal-width</th>
<th>petal-length</th>
<th>petal-width</th>
</tr>
</thead>
<tbody>
<tr>
<td>sepal-length</td>
<td>1.000000</td>
<td>-0.109369</td>
<td>0.871754</td>
<td>0.817954</td>
</tr>
<tr>
<td>sepal-width</td>
<td>-0.109369</td>
<td>1.000000</td>
<td>-0.420516</td>
<td>-0.356544</td>
</tr>
<tr>
<td>petal-length</td>
<td>0.871754</td>
<td>-0.420516</td>
<td>1.000000</td>
<td>0.962757</td>
</tr>
<tr>
<td>petal-width</td>
<td>0.817954</td>
<td>-0.356544</td>
<td>0.962757</td>
<td>1.000000</td>
</tr>
</tbody>
</table>
# Split-out validation dataset
array = df.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = 
model_selection.train_test_split(X, Y,
test_size=validation_size, random_state=seed)
scoring = 'accuracy'

len(Y_validation),
# Models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results =
model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %.4f (%.4f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# Models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))

# evaluate each model in turn
results = []
names = []
for name, model in models:
kfold = model_selection.KFold(n_splits=10, random_state=seed)
cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
results.append(cv_results)
names.append(name)
msg = "%s: %.4f (%.4f)" % (name, cv_results.mean(), cv_results.std())
print(msg)

LR: 0.9667 (0.0408)
LDA: 0.9750 (0.0382)
KNN: 0.9833 (0.0333)
DT: 0.9750 (0.0382)
NB: 0.9750 (0.0534)
SVM: 0.9917 (0.0250)
# Make predictions on validation dataset
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)
# Make predictions on validation dataset
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)

0.9000

[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>7</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>0.85</td>
<td>0.92</td>
<td>0.88</td>
<td>12</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.90</td>
<td>0.82</td>
<td>0.86</td>
<td>11</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.90</td>
<td>0.90</td>
<td>0.90</td>
<td>30</td>
</tr>
</tbody>
</table>

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=5, p=2,
weights='uniform')
# Make predictions on validation dataset
model = SVC()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)
model = SVC()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# Make predictions on validation dataset
model = SVC()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("% .4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)

0.9333
[[ 7  0  0]
 [ 0 10  2]
 [ 0  0 11]]

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>7</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>1.00</td>
<td>0.83</td>
<td>0.91</td>
<td>12</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.85</td>
<td>1.00</td>
<td>0.92</td>
<td>11</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.94</td>
<td>0.93</td>
<td>0.93</td>
<td>30</td>
</tr>
</tbody>
</table>

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)
# Make predictions on validation dataset
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)

0.9000
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>7</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>0.85</td>
<td>0.92</td>
<td>0.88</td>
<td>12</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.90</td>
<td>0.82</td>
<td>0.86</td>
<td>11</td>
</tr>
</tbody>
</table>

avg / total       0.90    0.90    0.90    30

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
max_features=None, 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, presort=False, random_state=None,
splitter='best')
# Make predictions on validation dataset
model = GaussianNB()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)

0.8333
[[7 0 0]
 [0 9 3]
 [0 2 9]]

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>7</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>0.82</td>
<td>0.75</td>
<td>0.78</td>
<td>12</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.75</td>
<td>0.82</td>
<td>0.78</td>
<td>11</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.84</td>
<td>0.83</td>
<td>0.83</td>
<td>30</td>
</tr>
</tbody>
</table>

GaussianNB(priors=None)
# Make predictions on validation dataset
model = LogisticRegression()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)

```
0.8000
[[ 7  0  0]
 [ 0  7  5]
 [ 0 10  0]]

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>7</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>0.88</td>
<td>0.58</td>
<td>0.70</td>
<td>12</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.67</td>
<td>0.91</td>
<td>0.77</td>
<td>11</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.83</td>
<td>0.80</td>
<td>0.80</td>
<td>30</td>
</tr>
</tbody>
</table>
```

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
# Make predictions on validation dataset
model = LinearDiscriminantAnalysis()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model)

0.9667
[[ 7  0  0]
 [ 0 11  1]
 [ 0  0 11]]

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris-setosa</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td>1.00</td>
<td>0.92</td>
<td>0.96</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.92</td>
<td>1.00</td>
<td>0.96</td>
</tr>
</tbody>
</table>

avg / total | 0.97 | 0.97 | 0.97 | 30 |

LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None, solver='svd', store_covariance=False, tol=0.0001)
# Make predictions on validation dataset

```python
model = MLPClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
print(model).
```

```
0.9000
[[  7   0   0]
 [  0  99  3]
 [  0   0  11]]

 precision   recall   f1-score   support

   Iris-setosa   1.00     1.00      1.00        7
   Iris-versicolor  1.00   0.75       0.86       12
   Iris-virginica  0.79     1.00      0.88       11

   avg / total  0.92     0.90      0.90       30

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant', learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)
```
Machine Learning
Unsupervised Learning
Cluster Analysis
K-Means Clustering
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt

# importing pandas
import pandas as pd

# Importing the Iris dataset with pandas
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
df = pd.read_csv(url, names=names)

array = df.values
X = array[:,0:4]
Y = array[:,4]

# Finding the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 8):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.rcParams["figure.figsize"] = (10,8)
plt.plot(range(1, 8), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # within cluster sum of squares
plt.show().
# importing the libraries
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd

# importing the Iris dataset with pandas
# Load dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
df = pd.read_csv(url, names=names)

array = df.values
X = array[:,0:4]
Y = array[:,4]
#Finding the optimum number of clusters for k-means classification

```python
from sklearn.cluster import KMeans

wcss = []

for i in range(1, 8):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
                        max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.rcParams["figure.figsize"] = (10,8)
plt.plot(range(1, 8), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```
**K-Means Clustering**

The elbow method \((k=3)\)
```python
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(X)
```

1 #Applying kmeans to the dataset / Creating the kmeans classifier
2 kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
3 y_kmeans = kmeans.fit_predict(X).
# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')
plt.legend()
K-Means Clustering

```python
# Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(X).

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolor')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
```

https://colab.research.google.com/drive/1QE7fR20xHiQ0_p6I1nnZD1FF354Nf_Lw
Aurélien Géron (2019), 
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: 
Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Edition 
O’Reilly Media, 2019

https://github.com/ageron/handson-ml2

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

Notebooks
1. The Machine Learning landscape
2. End-to-end Machine Learning project
3. Classification
4. Training Models
5. Support Vector Machines
6. Decision Trees
7. Ensemble Learning and Random Forests
8. Dimensionality Reduction
9. Unsupervised Learning Techniques
10. Artificial Neural Nets with Keras
11. Training Deep Neural Networks
12. Custom Models and Training with TensorFlow
13. Loading and Preprocessing Data
14. Deep Computer Vision Using Convolutional Neural Networks
15. Processing Sequences Using RNNs and CNNs
16. Natural Language Processing with RNNs and Attention
17. Representation Learning Using Autoencoders
18. Reinforcement Learning
19. Training and Deploying TensorFlow Models at Scale

https://github.com/ageron/handson-ml2
Sequences using RNNs and CNNs

```python
np.random.seed(43)

series = generate_time_series(1, 50 + 10)
X_new, Y_new = series[:, 50:], series[:, 50:]
Y_pred = model.predict(X_new)[..., -1][..., np.newaxis]

plot_multiple_forecasts(X_new, Y_new, Y_pred)
plt.show()
```

TensorFlow

An end-to-end open source machine learning platform

The core open source library to help you develop and train ML models. Get started quickly by running Colab notebooks directly in your browser.

Get started with TensorFlow

https://www.tensorflow.org/
• An end-to-end open source machine learning platform.
• The core open source library to help you develop and train ML models.
• Get started quickly by running Colab notebooks directly in your browser.

https://www.tensorflow.org/
Why TensorFlow 2.0

Why TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

About →

Easy model building
Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere
Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Powerful experimentation for research
A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.
# TensorFlow 2.0 vs. 1.X

# TensorFlow 2.0
outputs = f(input)

# TensorFlow 1.X
outputs = session.run(f(placeholder), feed_dict={placeholder: input})

Source: [https://www.tensorflow.org/guide/effective_tf2](https://www.tensorflow.org/guide/effective_tf2)
```python
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
                                     tf.keras.layers.Dense(128, activation='relu'),
                                     tf.keras.layers.Dropout(0.2),
                                     tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

https://www.tensorflow.org/overview/
TensorFlow 2 quickstart for beginners

This short introduction uses Keras to:

1. Build a neural network that classifies images.
2. Train this neural network.
3. And, finally, evaluate the accuracy of the model.

This is a Google Colaboratory notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select CONNECT.
2. Run all the notebook code cells: Select Runtime > Run all.

Download and install the TensorFlow 2 package. Import TensorFlow into your program:

```python
from __future__ import absolute_import, division, print_function, unicode_literals

# Install TensorFlow
try:
    # `tensorflow_version` only exists in Colab.
    tf_version
except Exception:
    pass
```

Basic classification: Classify images of clothing

This guide trains a neural network model to classify images of clothing, like sneakers and shirts. It's okay if you don't understand all the details; this is a fast-paced overview of a complete TensorFlow program with the details explained as you go.

This guide uses tf.keras, a high-level API to build and train models in TensorFlow.

```python
from __future__ import absolute_import, division, print_function, unicode_literals

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

https://www.tensorflow.org/tutorials/keras/classification
Image Classification

Fashion MNIST dataset

https://www.tensorflow.org/tutorials/keras/classification
Text classification with TensorFlow Hub: Movie reviews

This notebook classifies movie reviews as *positive* or *negative* using the text of the review. This is an example of binary—or two-class—classification, an important and widely applicable kind of machine learning problem.

The tutorial demonstrates the basic application of transfer learning with TensorFlow Hub and Keras.

We'll use the IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are balanced, meaning they contain an equal number of positive and negative reviews.

This notebook uses `tf.keras`, a high-level API to build and train models in TensorFlow, and `TensorFlow Hub`, a library and platform for transfer learning. For a more advanced text classification tutorial using `tf.keras`, see the MLCC Text Classification Guide.
Text classification with preprocessed text: Movie reviews

This notebook classifies movie reviews as positive or negative using the text of the review. This is an example of binary—or two-class—classification, an important and widely applicable kind of machine learning problem.

We'll use the IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are balanced, meaning they contain an equal number of positive and negative reviews.

This notebook uses tf.keras, a high-level API to build and train models in TensorFlow. For a more advanced text classification tutorial using tf.keras, see the MLCC Text Classification Guide.

Setup

from __future__ import absolute_import, division, print_function, unicode_literals

https://www.tensorflow.org/tutorials/keras/text_classification
Basic regression: Predict fuel efficiency

In a regression problem, we aim to predict the output of a continuous value, like a price or a probability. Contrast this with a classification problem, where we aim to select a class from a list of classes (for example, where a picture contains an apple or an orange, recognizing which fruit is in the picture).

This notebook uses the classic Auto MPG Dataset and builds a model to predict the fuel efficiency of late-1970s and early 1980s automobiles. To do this, we'll provide the model with a description of many automobiles from that time period. This description includes attributes like: cylinders, displacement, horsepower, and weight.

This example uses the `tf.keras` API, see this guide for details.

```python
# Use seaborn for pairplot
!pip install -q seaborn
```

```python
from __future__ import absolute_import, division, print_function, unicode_literals
import pathlib
```
Time series forecasting

This tutorial is an introduction to time series forecasting using Recurrent Neural Networks (RNNs). This is covered in two parts: first, you will forecast a univariate time series, then you will forecast a multivariate time series.

```python
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
```
TensorFlow Playground

Tinker With a Neural Network Right Here in Your Browser.
Don’t Worry, You Can’t Break It. We Promise.

[Interactive Neural Network Diagram]

DATA
Which dataset do you want to use?

INPUT
Which properties do you want to feed in?

OUTPUT
Test loss 0.000
Training loss 0.000

http://playground.tensorflow.org/
Tensor

• 3
  – # a rank 0 tensor; this is a scalar with shape []
• [1., 2., 3.]
  – # a rank 1 tensor; this is a vector with shape [3]
• [[1., 2., 3.], [4., 5., 6.]]
  – # a rank 2 tensor; a matrix with shape [2, 3]
• [[[1., 2., 3.]], [[[7., 8., 9.]]]]
  – # a rank 3 tensor with shape [2, 1, 3]

https://www.tensorflow.org/
Scalar

Vector

Matrix

Tensor
Time Series Data

```python
df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')
```

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/
Time Series Data

\[ [100, 110, 120, 130, 140, 150] \]
Long Short Term Memory (LSTM) for Time Series Forecasting
### Time Series Data

\[ [10, 20, 30, 40, 50, 60, 70, 80, 90] \]

<table>
<thead>
<tr>
<th>X</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10 20 30]</td>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[20 30 40]</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[30 40 50]</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[40 50 60]</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[50 60 70]</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[60 70 80]</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Deep Learning
and
Neural Networks
Deep Learning
Foundations:
Neural Networks
Deep Learning and Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

X1

X2

Y
Deep Learning and Neural Networks

Input Layer (X)

Hidden Layer (H)

Output Layer (Y)
Deep Learning and Neural Networks

Input Layer (X)

Hidden Layers (H)

Output Layer (Y)

Deep Neural Networks
Deep Learning
Deep Learning
and
Deep Neural Networks
Neural Networks (NN)
A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org
Convolutional Neural Networks
(CNN or Deep Convolutional Neural Networks, DCNN)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Recurrent Neural Networks (RNN)

Source: http://www.asimovinstitute.org/neural-network-zoo/
Long / Short Term Memory (LSTM)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Gated Recurrent Units (GRU)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Generative Adversarial Networks (GAN)


Source: http://www.asimovinstitute.org/neural-network-zoo/
Support Vector Machines (SVM)


Source: http://www.asimovinstitute.org/neural-network-zoo/
From image to text

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

From image to text

Image: deep convolution neural network (CNN)
Text: recurrent neural network (RNN)

A group of **people** sitting on a boat in the water.

Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

Source: https://www.youtube.com/watch?v=bx2T-V8XR&index=1&list=PLiaHhY2lBX9hdHaRr6b7XevZtgZRa1PoU
The Neuron

\[
\begin{align*}
x_1 \quad w_1 \\
x_2 \quad w_2 \\
\vdots \\
x_n \quad w_n \\
y
\end{align*}
\]
Neuron and Synapse

Source: https://en.wikipedia.org/wiki/Neuron
The Neuron

\[ y = F \left( \sum_{i} w_i x_i \right) \]

- \( x_1, x_2, \ldots, x_n \) are inputs.
- \( w_1, w_2, \ldots, w_n \) are weights.
- \( F(x) = \max(0, x) \) is the activation function.

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
\[ y = \max \left( 0, -0.21 \times x_1 + 0.3 \times x_2 + 0.7 \times x_3 \right) \]
Neural Networks
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

Input Layer (X)  Hidden Layers (H)  Output Layer (Y)

Deep Neural Networks
Deep Learning

Source: https://www.youtube.com/watch?v=bxetV8XRs&index=1&list=PLiaHhY2iBXhdHaRt6b7XevZtgZRd1PoU
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

Neuron

X1  Synapse  Synapse

X2

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

Source: https://www.youtube.com/watch?v=bx2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

Input Layer (X)  

Hidden Layer (H)

Output Layer (Y)

Source: https://www.youtube.com/watch?v=P2HPcj8lRJE&list=PLjJh1vlSEYqvGod9wWiydumYl8hOXixNu&index=2
Neural Networks

Input Layer (X)  Hidden Layer (H)  Output Layer (Y)

Source: https://www.youtube.com/watch?v=bxet-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRv6b7XevZtgZRa1PoU
<table>
<thead>
<tr>
<th>Hours Sleep</th>
<th>Hours Study</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

Source: https://www.youtube.com/watch?v=bxе2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRт6б7XevZtgZRа1PoU
<table>
<thead>
<tr>
<th>Hours Sleep</th>
<th>Hours Study</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>82</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>93</td>
</tr>
</tbody>
</table>

Training

Testing

| 8 | 3 | ? |
$Y = WX + b$
\[ Y = W X + b \]

Output \[\rightarrow\] input

Weights \[\leftarrow\] bias

Trained

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
\[ \mathbf{W} \mathbf{X} + \mathbf{b} = \mathbf{Y} \]

Scores \rightarrow Probabilities

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
SoftMAX

\[ W \cdot X + b = Y \]

\[
\begin{bmatrix}
2.0 \\
1.0 \\
0.1 \\
\end{bmatrix}
\]

\[ S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \]

\[
\begin{bmatrix}
0.7 \\
0.2 \\
0.1 \\
\end{bmatrix}
\]

Logits \quad \rightarrow \quad \text{Scores} \quad \rightarrow \quad \text{Probabilities}

Source: https://www.youtube.com/watch?v=G8eNWzOqgE
\[
S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} = \frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{2.7182^{2.0}}{2.7182^{2.0} + 2.7182^{1.0} + 2.7182^{0.1}} = 0.7
\]

\[
S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} = \frac{e^{1.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{2.7182^{1.0}}{2.7182^{2.0} + 2.7182^{1.0} + 2.7182^{0.1}} = 0.2
\]

\[
S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} = \frac{e^{0.1}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{2.7182^{0.1}}{2.7182^{2.0} + 2.7182^{1.0} + 2.7182^{0.1}} = 0.1
\]

\[
W \times X + b = Y
\]

Logits  \xrightarrow{\text{Scores}} \xrightarrow{\text{Probabilities}}

Source: https://www.youtube.com/watch?v=G8eNWzOGqE
Training a Network

= Minimize the Cost Function
Training a Network

= Minimize the Cost Function

Minimize the Loss Function

Source: https://www.youtube.com/watch?v=bxetV8XR&index=1&list=PLiaHhY2iBXhHaRr6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y

Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bx2T-V8XR&index=1&list=PLiaHhY2iBX9hdHaR6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y

Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bx2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y

Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bxw2T-V8XRg&index=1&list=PLiaHhY2lBX9hdHaRt6b7XevZtgZRa1PoU
Activation Functions
Activation Functions

**Sigmoid**

**TanH**

**ReLU**

(Rectified Linear Unit)

**f(x) = max(0, x)**

Activation Functions

Sigmoid

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

TanH

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

ReLU

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

Source: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/
Loss Function
Binary Classification: 2 Class

Activation Function: Sigmoid

Loss Function: Binary Cross-Entropy
Multiple Classification: 10 Class

Activation Function: SoftMAX

Loss Function: Categorical Cross-Entropy
Dropout: a simple way to prevent neural networks from overfitting

(a) Standard Neural Net

(b) After applying dropout.

Learning Algorithm

While not done:

Pick a random training example “(input, label)”
Run neural network on “input”
Adjust weights on edges to make output closer to “label”
\[ y = \max \left( 0, -0.21 \times x_1 + 0.3 \times x_2 + 0.7 \times x_3 \right) \]
Next time:

\[ y = \max(0, -0.23 \times x_1 + 0.31 \times x_2 + 0.65 \times x_3) \]

\[ y = \max(0, -0.21 \times x_1 + 0.3 \times x_2 + 0.7 \times x_3) \]

Weights

Inputs

\( x_1 \)

\( x_2 \)

\( x_3 \)

\( y \)
Optimizer:
Stochastic Gradient Descent (SGD)

\[ J(w) \]

Global cost minimum

Initial weight
This shows a function of 2 variables: real neural nets are functions of hundreds of millions of variables!

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Neural Network and Deep Learning

Source: 3Blue1Brown (2017), But what *is* a Neural Network? | Chapter 1, deep learning, https://www.youtube.com/watch?v=aircAruvnKk
Gradient Descent
how neural networks learn

Average cost of all training data...

Cost of 8

What's the "cost" of this difference?

Source: 3Blue1Brown (2017), Gradient descent, how neural networks learn | Chapter 2, deep learning, https://www.youtube.com/watch?v=IHZwWFHwa-w
Backpropagation

Source: 3Blue1Brown (2017), What is backpropagation really doing? | Chapter 3, deep learning, https://www.youtube.com/watch?v=Ilg3gGewQ5U
Learning Algorithm

While not done:

Pick a random training example “(input, label)”
Run neural network on “input”
Adjust weights on edges to make output closer to “label”

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Deep Learning for Financial Time Series Forecasting
Deep Learning for Financial Market Prediction
Stock Market Prediction
Stock Price Prediction
Time Series Prediction
Time Series Data

df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/
Time Series Data

[100, 110, 120, 130, 140, 150]
Long Short Term Memory (LSTM) for Time Series Forecasting

\[ h_{t-2} \rightarrow \text{LSTM} \rightarrow h_{t-1} \rightarrow \text{LSTM} \rightarrow h_t \rightarrow \text{LSTM} \rightarrow h_{t+1} \rightarrow \text{LSTM} \rightarrow h_{t+2} \]

\[ X_{t-2} \rightarrow \text{LSTM} \rightarrow X_{t-1} \rightarrow \text{LSTM} \rightarrow X_t \rightarrow \text{LSTM} \rightarrow X_{t+1} \rightarrow \text{LSTM} \rightarrow X_{t+2} \]
## Time Series Data

\[ [10, 20, 30, 40, 50, 60, 70, 80, 90] \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10 20 30]</td>
<td>40</td>
</tr>
<tr>
<td>[20 30 40]</td>
<td>50</td>
</tr>
<tr>
<td>[30 40 50]</td>
<td>60</td>
</tr>
<tr>
<td>[40 50 60]</td>
<td>70</td>
</tr>
<tr>
<td>[50 60 70]</td>
<td>80</td>
</tr>
<tr>
<td>[60 70 80]</td>
<td>90</td>
</tr>
</tbody>
</table>
Deep Learning for Financial Time Series Forecasting

https://colab.research.google.com/drive/1aEK0eSev8Q-Y0nNY32geFk7CB8pVgSQM

```python
# univariate data preparation
from numpy import array
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Source: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
Deep Learning for Financial Time Series Forecasting

https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

LSTM for Time Series Forecasting

```python
# univariate lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
import matplotlib.pyplot as plt
%matplotlib inline

# define dataset
X = array([[100, 110, 120], [110, 120, 130], [120, 130, 140], [130, 140, 150], [140, 150, 160]])
y = array([130, 140, 150, 160, 170])
# reshape from [samples, timesteps] into [samples, timesteps, features]
X = X.reshape((X.shape[0], X.shape[1], 1))

# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(3, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# fit model
history = model.fit(X, y, epochs=2000, verbose=0)

# demonstrate prediction
x_input = array([[150, 160, 170]])
x_input = x_input.reshape((1, 3, 1))
yhat = model.predict(x_input, verbose=0)
print('yhat', yhat)
print(model.summary())

# list all data in history
print(history.history.keys())
# summarize history for loss
print('loss:', history.history['loss'][-1])
print('loss:', history.history['val_loss'][-1])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()

yhat [[181.34615]]
```

Source: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
Deep Learning for Financial Time Series Forecasting

https://colab.research.google.com/drive/1aEK0eSev8Q-Y0nNY32geFk7CB8pVgSQM

Using TensorFlow backend.
[[102.31296]]
yhat [[102.31296]]

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>lstm_1 (LSTM)</td>
<td>(None, 50)</td>
<td>10400</td>
</tr>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 1)</td>
<td>51</td>
</tr>
</tbody>
</table>

Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0

None
dict_keys(['loss'])
loss: 0.000000
loss: 1.2578432517784677e-07
Deep Learning for Financial Time Series Forecasting

https://colab.research.google.com/drive/1aEK0eSev8Q-Y0nNY32geFk7CB8pVgSQM

Source: https://github.com/yash-1337/AAPL_LSTM_Stock_Predictor/blob/master/AAPL_daily_LSTM_stock_predictor.ipynb
Basic Classification
Fashion MNIST Image Classification

https://colab.research.google.com/drive/19PJOJi1vn1kjcuiNshjRSLbeVI4kd5z

Copyright 2018 The TensorFlow Authors.
Licensed under the Apache License, Version 2.0 (the "License");
MIT License

Train your first neural network: basic classification

This guide trains a neural network model to classify images of clothing, like sneakers and shirts. It's okay if you don't understand all the details, this is a fast-paced overview of a complete TensorFlow program with the details explained as we go.

This guide uses tf.keras, a high-level API to build and train models in TensorFlow.

```
# memory footprint support libraries/code
1 ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
2 ln -sf /usr/lib/nvidia-modprobe /usr/lib/modprobe
3 pip install gputil
4 pip install psutil
5 pip install humanize
6 import psutil
7 import humanize
8 import os
9 import GPUtil as GPU
10 GPUs = GPU.getGPUs()
11 gpu = GPUs[0]
12 def printm():
13     process = psutil.Process(os.getpid())
14     print("Gen RAM Free: " + humanize.naturalsize( psutil.virtual_memory().available ), "|
15     print("GPU RAM Free: {0:.0f}MB | Used: {1:.0f}MB | Util {2:.0f}% | Total {3:.0f}MB".format
16     printm()
```
Text classification
IMDB Movie Reviews

[Link to source code](https://colab.research.google.com/drive/1x16h1GhHsLrLYtPCvCHaoO1W-i_gror)

Copyright 2018 The TensorFlow Authors.

Licensed under the Apache License, Version 2.0 (the "License");

MIT License

Text classification with movie reviews

Download the IMDB dataset
Explore the data
Convert the integers back to words
Prepare the data
Build the model
Hidden units
Loss function and optimizer
Create a validation set
Train the model
Evaluate the model

This notebook classifies movie reviews as positive or negative using the text of the review. This is an example of binary—or two-class—classification, an important and widely applicable kind of machine learning problem.

We'll use the IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are balanced, meaning they contain an equal number of positive and negative reviews.

This notebook uses tf.keras, a high-level API to build and train models in TensorFlow. For a more advanced text classification tutorial using tf.keras, see the MLCC Text Classification Guide.
Basic Regression
Predict House Prices

https://colab.research.google.com/drive/1v4c8ZHTnRtgdr2_25K_AURjR6SCVBRdlj

This notebook builds a model to predict the median price of homes in a Boston suburb during the mid-1970s. To do this, we'll provide the model with some data points about the suburb, such as the crime rate and the local property tax rate.

This example uses the tf.keras API, see this guide for details.

```python
# memory footprint support libraries/code
!ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
!pip install gputil
!pip install psutil
import psutil
import humanize
import os
import GPUtil as GPU
GPUs = GPU.getGPUs()
gpu = GPUs[0]
def printm():
    process = psutil.Process(os.getpid())
    print("Gen RAM Free: ",humanize.naturalsize( psutil.virtual_memory().available )," | Proc size: "
          ,humanize.naturalsize( process.memory_info().rss )," | Total ",humanize.naturalsize( process.memory_info().vms ))
```

Source: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/basic_regression.ipynb
```python
# Python in Google Colab (Python101)

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT

# Read Stock Data from Yahoo Finance
end = dt.datetime.now()
start = dt.datetime(2017, 1, 1)
df = web.DataReader('AAPL', 'yahoo', start, end)
df.to_csv('AAPL.csv')
print(df.tail())
print(df2.tail())

# plots
df['Adj Close'].plot(legend=True, figsize=(12, 8), title='AAPL', label='Adj Close')
plt.figure(figsize=(12, 9))
top = plt.subplot2grid((12, 9), (0, 0), rowspan=10, colspan=9)
bottom = plt.subplot2grid((12, 9), (10, 0), rowspan=2, colspan=9)
top.plot(df.index, df['Adj Close'], color='blue')
bottom.plot(df.index, df['Volume'])
bottom.bar(df.index, df['Volume'])

# set the labels
top.set_xlabel('').set_visible(False)
top.set_title('AAPL')
bottom.set_xlabel('Adj Close')
bottom.set_ylabel('Volume')
plt.figure(figsize=(12, 9))
sns.despine(left=True)
sns.distplot(df['Adj Close'], dropna(), bins=50, color='purple')

# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean()  # 5 days
df['MA20'] = df['Adj Close'].rolling(20).mean()  # 20 days
df['MA60'] = df['Adj Close'].rolling(60).mean()  # 60 days

# simple moving averages
df2 = pd.DataFrame({
    'Adj Close': df['Adj Close'],
    'MA05': df['MA05'],
    'MA20': df['MA20'],
    'MA60': df['MA60']
})
df2.plot(figsize=(12, 9), legend=True, title='AAPL')
df2.to_csv('AAPL_MA.csv')

fig = plt.gcf()
fig.set_size_inches(12, 9)
fig.savefig('AAPL_plot.png', dpi=300)

https://tinyurl.com/imtkupython101
```
np.where
(df['MA20'] > df['MA60'],
12000,
9000)

# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean()  # 5 days
df['MA20'] = df['Adj Close'].rolling(20).mean()  # 20 days
df['MA60'] = df['Adj Close'].rolling(60).mean()  # 60 days
df['Positions'] = np.where(df['MA20'] > df['MA60'], 12000, 9000)
df2 = pd.DataFrame({'Adj Close': df['Adj Close'], 'MA05': df['MA05'], 'MA20': df['MA20'], 'MA60': df['MA60'], 'Positions': df['Positions']})
np.where
(df['MA20'] > df['MA60'], 1, 0)

# simple moving averages
df['MA05'] = df['Adj Close'].rolling(5).mean() #5 days
df['MA20'] = df['Adj Close'].rolling(20).mean() #20 days
df['MA60'] = df['Adj Close'].rolling(60).mean() #60 days

df['Positions'] = np.where(df['MA20'] > df['MA60'], 1, 0)
df2 = pd.DataFrame({'Adj Close': df['Adj Close'], 'MA05': df['MA05'], 'MA20': df['MA20'], 'MA60': df['MA60'], 'Positions': df['Positions']})
Summary

• Machine Learning
• Deep Learning
References

- Ties de Kok (2017), Learn Python for Research, https://github.com/TiesdeKok/LearnPythonforResearch
- Data School (2015), Machine learning in Python with scikit-learn, https://www.youtube.com/playlist?list=PL5-da3qGB5IcMbQugbbCOQWcS6OYbr5A
- Deep Learning Basics: Neural Networks Demystified, https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
- Deep Learning SIMPLIFIED, https://www.youtube.com/playlist?list=PLjJh1vlSEYgvGod9wWiydumYl8hOXixNu
- 3Blue1Brown (2017), But what *is* a Neural Network? | Chapter 1, deep learning, https://www.youtube.com/watch?v=aicrAruvnKk
- 3Blue1Brown (2017), Gradient descent, how neural networks learn | Chapter 2, deep learning, https://www.youtube.com/watch?v=IHzWtFHWa-w
- 3Blue1Brown (2017), What is backpropagation really doing? | Chapter 3, deep learning, https://www.youtube.com/watch?v=Ilg3gGewQ5U
- Jay Alammar (2019), The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning), http://jalammar.github.io/illustrated-bert/
- TensorFlow: https://www.tensorflow.org/
- Keras: http://keras.io/
- Udacity, Deep Learning, https://www.youtube.com/playlist?list=PLAwxTw4SYaPn_OWPT9uIXLuQrlmzHfOV
- https://github.com/leriomaggio/deep-learning-keras-tensorflow