Practices of Business Intelligence

機器學習與深度學習
(Machine Learning and Deep Learning)

1071BI10
MI4 (M2084) (2888)
Wed, 7, 8 (14:10-16:00) (B217)

Min-Yuh Day
戴敏育
Assistant Professor
Dept. of Information Management, Tamkang University

http://mail.tku.edu.tw/myday/
2018-12-05
<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  2018/09/12</td>
<td></td>
<td>商業智慧實務課程介紹  (Course Orientation for Practices of Business Intelligence)</td>
</tr>
<tr>
<td>2  2018/09/19</td>
<td></td>
<td>商業智慧、分析與資料科學  (Business Intelligence, Analytics, and Data Science)</td>
</tr>
<tr>
<td>3  2018/09/26</td>
<td></td>
<td>人工智慧、大數據與雲端運算  (ABC: AI, Big Data, and Cloud Computing)</td>
</tr>
<tr>
<td>4  2018/10/03</td>
<td></td>
<td>描述性分析I：數據的性質、統計模型與可視化  (Descriptive Analytics I: Nature of Data, Statistical Modeling, and Visualization)</td>
</tr>
<tr>
<td>5  2018/10/10</td>
<td></td>
<td>國慶紀念日 (放假一天)  (National Day) (Day off)</td>
</tr>
<tr>
<td>6  2018/10/17</td>
<td></td>
<td>描述性分析II：商業智慧與資料倉儲  (Descriptive Analytics II: Business Intelligence and Data Warehousing)</td>
</tr>
</tbody>
</table>
課程大綱 (Syllabus)

週次 (Week) 日期 (Date) 內容 (Subject/Topics)
7 2018/10/24 預測性分析 I：資料探勘流程、方法與演算法
  (Predictive Analytics I: Data Mining Process, Methods, and Algorithms)
8 2018/10/31 預測性分析 II：文本、網路與社群媒體分析
  (Predictive Analytics II: Text, Web, and Social Media Analytics)
9 2018/11/07 期中報告 (Midterm Project Report)
10 2018/11/14 期中考試 (Midterm Exam)
11 2018/11/21 處方性分析：最佳化與模擬
  (Prescriptive Analytics: Optimization and Simulation)
12 2018/11/28 社會網絡分析
  (Social Network Analysis)
課程大綱 (Syllabus)

週次 (Week) 日期 (Date) 內容 (Subject/Topics)

13 2018/12/05 機器學習與深度學習 (Machine Learning and Deep Learning)

14 2018/12/12 自然語言處理 (Natural Language Processing)

15 2018/12/19 AI交談機器人與對話式商務 (AI Chatbots and Conversational Commerce)

16 2018/12/26 商業分析的未來趨勢、隱私與管理考量 (Future Trends, Privacy and Managerial Considerations in Analytics)

17 2019/01/02 期末報告 (Final Project Presentation)

18 2019/01/09 期末考試 (Final Exam)
Business Intelligence (BI)

1. Introduction to BI and Data Science
2. Descriptive Analytics
3. Predictive Analytics
4. Prescriptive Analytics
5. Big Data Analytics
6. Future Trends
Machine Learning
and
Deep Learning
Outline

• Machine Learning

• Deep Learning
Artificial Intelligence (AI)

Machine Learning (ML)

Supervised Learning

Unsupervised Learning

Deep Learning (DL)

- CNN
- RNN
- LSTM
- GRU
- GAN

Semi-supervised Learning

Reinforcement Learning

Source: https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/deep_learning.html
3 Machine Learning Algorithms

Machine Learning Models

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

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

- **Machine Learning (ML)**
  - Supervised Learning
    - Decision Tree Classifiers
    - Linear Classifiers
    - Rule-based Classifiers
    - Probabilistic Classifiers
  - Unsupervised Learning
    - Deep Learning (DL)
      - Support Vector Machine (SVM)
      - Neural Network (NN)
      - Naïve Bayes (NB)
      - Bayesian Network (BN)
      - Maximum Entropy (ME)
  - Reinforcement Learning

# Data Mining Tasks & Methods

<table>
<thead>
<tr>
<th>Data Mining Tasks &amp; Methods</th>
<th>Data Mining Algorithms</th>
<th>Learning Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prediction</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td>Decision Trees, Neural Networks, Support Vector Machines, kNN, Naïve Bayes, GA</td>
<td>Supervised</td>
</tr>
<tr>
<td>Regression</td>
<td>Linear/Nonlinear Regression, ANN, Regression Trees, SVM, kNN, GA</td>
<td>Supervised</td>
</tr>
<tr>
<td>Time series</td>
<td>Autoregressive Methods, Averaging Methods, Exponential Smoothing, ARIMA</td>
<td>Supervised</td>
</tr>
<tr>
<td><strong>Association</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link analysis</td>
<td>Expectation Maximization, Apriori Algorithm, Graph-Based Matching</td>
<td>Unsupervised</td>
</tr>
<tr>
<td>Sequence analysis</td>
<td>Apriori Algorithm, FP-Growth, Graph-Based Matching</td>
<td>Unsupervised</td>
</tr>
<tr>
<td><strong>Segmentation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clustering</td>
<td>k-means, Expectation Maximization (EM)</td>
<td>Unsupervised</td>
</tr>
<tr>
<td>Outlier analysis</td>
<td>k-means, Expectation Maximization (EM)</td>
<td>Unsupervised</td>
</tr>
</tbody>
</table>

Source: Ramesh Sharda, Dursun Delen, and Efraim Turban (2017), *Business Intelligence, Analytics, and Data Science: A Managerial Perspective*, 4th Edition, Pearson
Data Mining Methods

• Classification
  – Classification
    • Class Label Prediction
  – Regression
    • Numeric Value Prediction

• Clustering

• Association

Source: Ramesh Sharda, Dursun Delen, and Efraim Turban (2017), Business Intelligence, Analytics, and Data Science: A Managerial Perspective, 4th Edition, Pearson
Scikit-Learn

Machine Learning in Python
Scikit-Learn

Scikit-Learn is a Python library for machine learning. It offers a wide range of algorithms for classification, regression, clustering, and dimensionality reduction. It is simple and efficient, accessible to everybody, and reusable in various contexts. It is built on NumPy, SciPy, and matplotlib, and is open source, commercially usable under a 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

classification

-start-

Naive Bayes
Text Data
Linear SVC

SVC
Ensemble Classifiers

kNeighbors Classifier
SGD Classifier

kernel approximation
NOT WORKING

<100K samples

NOT WORKING

>50 samples

get more data

regression

SGD Regressor

<100K samples

Lasso
ElasticNet

>50 samples

few features should be important

SVR(kernel="rbf")

Ensemble Regressors

clustering

Spectral Clustering
GMM

KMeans

<10K samples

<10K samples

MiniBatch KMeans
MeanShift
VBGMM

<10K categories known

just looking

predicted a quantity

predicted a category

do you have labeled data

number of categories known

<10K samples

<10K samples

tough luck

predicting structure

predicting a quantity

<100K samples

just looking

dimensionality reduction

Randomized PCA

<10K samples

kernel approximation

Isomap
Spectral Embedding
LLE

Scikit-Learn Machine Learning Map

Scikit-Learn Machine Learning Map

Scikit-Learn Machine Learning Map

Iris flower data set

setosa  versicolor  virginica

Source: https://en.wikipedia.org/wiki/Iris_flower_data_set
Iris Classification

iris.data


<table>
<thead>
<tr>
<th>Sepal Length</th>
<th>Sepal Width</th>
<th>Petal Length</th>
<th>Petal Width</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
Data Visualization in Google Colab

```python
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 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))
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>
df.tail(10)

```
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)

```python
print(df.info())

<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)
```
```python
def.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
Data Mining and Machine Learning in Google Colab

```python
# 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")

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

df.head(10)
df.tail(10)
df.describe()
df.info()
df.shape
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()
```
# 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())

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)
# 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
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>

[Link to Google Colab Notebook](https://colab.research.google.com/drive/1QE7fR2OxHiQ0_p6I1nnZDIFF354Nf_Lw)
# 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'
# 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

def 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]]

 precision   recall   f1-score    support

   Iris-setosa   1.00     1.00     1.00          7
  Iris-versicolor  0.85     0.92     0.88         12
  Iris-virginica  0.90     0.82     0.86         11

    avg / total  0.90     0.90     0.90         30
```

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]]

precision   recall   f1-score   support
Iris-setosa  1.00      1.00      1.00      7
Iris-versicolor  1.00     0.83      0.91     12
Iris-virginica   0.85     1.00      0.92     11

avg / total  0.94      0.93      0.93     30

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>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>0.85</td>
<td>0.92</td>
<td>0.88</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.90</td>
<td>0.82</td>
<td>0.86</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   1  10]]

<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)
```python
# 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]]

    precision  recall  f1-score  support

Iris-setosa       1.00    1.00    1.00           7
Iris-versicolor   1.00    0.92    0.96          12
Iris-virginica    0.92    1.00    0.96          11

    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
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  9  3]
 [ 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>0.75</td>
<td>0.86</td>
<td>12</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td>0.79</td>
<td>1.00</td>
<td>0.88</td>
<td>11</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.92</td>
<td>0.90</td>
<td>0.90</td>
<td>30</td>
</tr>
</tbody>
</table>

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
K-Means Clustering

https://colab.research.google.com/drive/1QE7fR2OxHiQ0_p6I1nnZDIFF354Nf_Lw

```python
# importing the libraries
import numpy as np
import matplotlib.pyplot as plt
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
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()
```

https://colab.research.google.com/drive/1QE7fR2OxHiQ0_p6I1nnZDIFF354Nf_Lw
# 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 ($k=3$)
kmeans = KMeans(n_clusters = 3, 
init = 'k-means++', max_iter = 300, 
n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(X)

#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-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

#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-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()
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

Source: http://www.asimovinstitute.org/neural-network-zoo/
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)
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/
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.

A group of people shopping at an outdoor market.

There are many vegetables at the fruit stand.

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)

X1

X2

Y

Source: https://www.youtube.com/watch?v=bx32T-V8XR8&index=1&list=PLiaHhY2iBX9hdHaiRt6b7XevZtgZRa1PoU
The Neuron

\[ x_1 \rightarrow w_1 \rightarrow \cdots \rightarrow y \]

\[ x_2 \rightarrow w_2 \rightarrow \cdots \]

\[ \cdots \]

\[ x_n \rightarrow w_n \rightarrow y \]

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
The Neuron

\[ y = F \left( \sum_{i} w_{i} x_{i} \right) \]

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

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

Source: https://www.youtube.com/watch?v=bxet-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=PLiaHhY2iBX9hdHaRt6b7XevZtgZRa1PoU
Neural Networks

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

Neuron

Synapse

X1

X2

Neuron

Source: https://www.youtube.com/watch?v=bx2T-V8XRs&index=1&list=PLiaHhY2lBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

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

Score

Hours
Sleep

Hours
Study

Source: https://www.youtube.com/watch?v=bx2TV8XR&index=1&list=PLiaHhY2lBX9hdHaRr6b7XevZtgZRa1PoU
Neural Networks

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

Source: https://www.youtube.com/watch?v=P2HPcj8lRJE&list=PLjJh1vIseyqGod9wWiydumYI8hOxiNu&index=2
Neural Networks

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

Source: https://www.youtube.com/watch?v=bxelV8XR&index=1&list=PLiaHhY2IBX9hdHaRr6b7XevZtZRa1PoU
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>Sleep</td>
<td></td>
</tr>
<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=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU
<table>
<thead>
<tr>
<th></th>
<th>Hours</th>
<th>Hours</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sleep</td>
<td>Study</td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td>5</td>
<td>1</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>Testing</td>
<td>8</td>
<td>3</td>
<td>?</td>
</tr>
</tbody>
</table>

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2lBX9hdHaRr6b7XevZtgZRa1PoU
\[ Y = WX + b \]
\[ Y = WX + b \]

Output: \[ Y \]

input: \[ WX \]

Weights: \[ W \]

bias: \[ b \]

Trained: \[ WX + b \]

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
\( W X + b = Y \)

Scores \( \rightarrow \) Probabilities

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

\[ W X + b = Y \]

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

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

Logits \rightarrow Scores \rightarrow Probabilities

Source: https://www.youtube.com/watch?v=G8eNWzxOgqE
$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} = \frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{2.7182^{2.0}}{2.7182^{2.0} + 2.7182^{1.0} + 2.7182^{0.1}} = 0.7$

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

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

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

Logits \hspace{1cm} Scores \hspace{1cm} Probabilities

Source: https://www.youtube.com/watch?v=G8eNWxzOgqE
Training a Network

= Minimize the Cost Function

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

= Minimize the **Cost** Function

Minimize the **Loss** Function

Source: https://www.youtube.com/watch?v=bxET-V8XR8&index=1&list=PLiaHhY2iBX9hdHaRt6b7XeVztgZRa1PoU
Error = Predict Y - Actual Y

Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bxe2T-V8XRs&index=1&list=PLiaHhY2iBX9hdHaRt6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y
Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bx2T-V8XR&index=1&list=PLiaHhY2lBX9hdHaR6b7XevZtgZRa1PoU
Error = Predict Y - Actual Y

Error : Cost : Loss

Source: https://www.youtube.com/watch?v=bx2T-V8XR8s&index=1&list=PLiaHhY2iBX9hRr6b7XevZtgZRa1PoU
Activation Functions
Activation Functions

Sigmoid

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

[0, 1]

TanH

\[ f(x) = \tanh(x) \]

[-1, 1]

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

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”

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

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

\[ y = \max \left( 0, -0.21 \times x_1 + 0.3 \times x_2 + 0.7 \times x_3 \right) \]
Optimizer:
Stochastic Gradient Descent (SGD)

$J(w)$

$w$

Initial weight

Global cost minimum

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

\[
\begin{align*}
(0.18 - 0.00)^2 + \\
(0.29 - 0.00)^2 + \\
(0.58 - 0.00)^2 + \\
(0.77 - 0.00)^2 + \\
(0.20 - 0.00)^2 + \\
(0.36 - 0.00)^2 + \\
(0.93 - 0.00)^2 + \\
(1.00 - 0.00)^2 + \\
(0.95 - 1.00)^2 + \\
(0.35 - 0.00)^2
\end{align*}
\]

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
Convolutional Neural Networks (CNN)

Architecture of LeNet-5 (7 Layers) (LeCun et al., 1998)


Convolutional Neural Networks (CNN)

• Convolution
• Pooling
• Fully Connection (FC) (Flattening)
A friendly introduction to Convolutional Neural Networks and Image Recognition

Convolution Layer

Pooling Layer

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-OL7ZB0MmU
A friendly introduction to Convolutional Neural Networks and Image Recognition

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-Ol7ZB0MmU
A friendly introduction to Convolutional Neural Networks and Image Recognition

Convolution Layer  Pooling Layer  Fully Connected Layer

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-OI7ZB0MmU
A friendly introduction to Convolutional Neural Networks and Image Recognition

Source: Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-O7ZB0MmU
CNN Architecture

**CNN Convolution Layer**

Convolution is a mathematical operation to merge two sets of information.

**Input**

```
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
```

**Filter / Kernel**

```
1 0 1
0 1 0
1 0 1
```

CNN Convolution Layer

Input x Filter --> Feature Map

receptive field: 3x3

Input x Filter

Feature Map

**CNN Convolution Layer**

**Input x Filter --> Feature Map**

receptive field: 3x3

<table>
<thead>
<tr>
<th></th>
<th>1x1</th>
<th>1x0</th>
<th>0x1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1x0</td>
<td>1x1</td>
<td>1x0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0x1</td>
<td>1x0</td>
<td>1x1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Input x Filter

Feature Map

### CNN Convolution Layer

**Example convolution operation shown in 2D using a 3x3 filter**

<table>
<thead>
<tr>
<th>1x1</th>
<th>1x0</th>
<th>1x1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>1x1</td>
<td>1x0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0x1</td>
<td>0x0</td>
<td>1x1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CNN Convolution Layer

10 different filters 10 feature maps of size 32x32x1

final output of the convolution layer:
a volume of size 32x32x10

CNN Convolution Layer
Sliding operation at 4 locations

CNN Convolution Layer

two feature maps
**CNN Convolution Layer**

**Stride** specifies how much we move the convolution filter at each step.

**Stride 1**

**Feature Map**

**CNN Convolution Layer**

**Stride** specifies how much we move the convolution filter at each step.

**Stride 2**

**Feature Map**

---

CNN Convolution Layer

Stride 1 with Padding

Feature Map

**CNN Pooling Layer**

Max Pooling

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

max pool with 2x2 window and stride 2

<table>
<thead>
<tr>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
CNN Architecture
4 convolution + pooling layers, followed by 2 fully connected layers

CNN Architecture
4 convolution + pooling layers, followed by 2 fully connected layers

https://gist.github.com/ardendertat/0fc5515057c47e7386fe04e9334504e3

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1',
                 input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2), name='maxpool_1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'))
model.add(MaxPooling2D((2, 2), name='maxpool_2'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'))
model.add(MaxPooling2D((2, 2), name='maxpool_3'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'))
model.add(MaxPooling2D((2, 2), name='maxpool_4'))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu', name='dense_1'))
model.add(Dense(128, activation='relu', name='dense_2'))
model.add(Dense(1, activation='sigmoid', name='output'))

Source: Arden Dertat (2017), Applied Deep Learning - Part 4: Convolutional Neural Networks,
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
Dropout

Model Performance

Recurrent Neural Network (RNN)
Course Description

Natural language processing (NLP) is one of the most important technologies of the information age. Understanding complex language utterances is also a crucial part of artificial intelligence. Applications of NLP are everywhere because people communicate most everything in language: web search, advertisement, emails, customer service, language translation, radiology reports, etc. There are a large variety of underlying tasks and machine learning models powering NLP applications. Recently, deep learning approaches have obtained very high performance across many different NLP tasks. These models can often be trained with a single end-to-end model and do not require traditional, task-specific feature engineering. In this spring quarter course students will learn to implement, train, debug, visualize and invent their own neural network models. The course provides a deep excursion into cutting-edge research in deep learning applied to NLP. The final project will involve training a complex recurrent neural network and applying it to a large scale NLP problem. On the model side we will cover word vector representations,
Recurrent Neural Networks (RNNs)
RNN

\[ h_t \]

\[ h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow \cdots \rightarrow h_t \]

\[ X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow \cdots \rightarrow X_t \]
RNN long-term dependencies

I grew up in France… I speak fluent French.

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long Short Term Memory (LSTM)

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Gated Recurrent Unit (GRU)

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
### LSTM vs GRU

**LSTM**

i, f and o are the input, forget and output gates, respectively. c and c
\(^\sim\) denote the memory cell and the new memory cell content.

**GRU**

r and z are the reset and update gates, and h and h
\(^\sim\) are the activation and the candidate activation.

LSTM Recurrent Neural Network

one to one

one to many

many to one

many to many

many to many

Source: https://github.com/Vict0rSch/deep_learning/tree/master/keras/recurrent
Long Short Term Memory (LSTM) for Time Series Forecasting

\[ X_t, X_{t-1}, h_{t-1}, h_t, h_{t+1}, h_{t+2} \]

\[ \text{LSTM} \]

\[ X_{t-2}, X_{t-1}, X_t, X_{t+1}, X_{t+2} \]
The Sequence to Sequence model (seq2seq)
Sequence to Sequence (Seq2Seq)

Source: https://google.github.io/seq2seq/
Financial Time Series Forecasting
Time Series Data
Time Series Data

\[ [100, 110, 120, 130, 140, 150] \]
The Quant Finance PyData Stack

- PyThalesians
- Zipline
- DX Analytics
- PyAlgoTrade
- QuantLib
- StatsModels
- NetworkX
- scikits-image
- PyMC
- SciPy
- NumPy
- Python
- SymPy
- Jupyter

Source: http://nbviewer.jupyter.org/format/slides/github/quantopian/pyfolio/blob/master/pyfolio/examples/overview_slides.ipynb
# !pip install pandas_datareader
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt

# Read Stock Data from Yahoo Finance
end = dt.datetime.now()
# start = dt.datetime(2016, 1, 1)
start = dt.datetime(2016 - 2, 1, 1)
df = web.DataReader('AAPL', 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.tail()

# Simple moving averages
df['MA5'] = 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

# Plot data
plt.figure(figsize=(12, 8))
top = plt.subplot2grid((12, 9), (0, 0), rowspan=10, colspan=9)
bottom = plt.subplot2grid((12, 9), (10, 0), rowspan=2, colspan=9)
top.ticks.set_visible(False)
top.set_title('AAPL')
top.set_xlabel('Adj Close')
top.set_ylabel('Volume')

bottom.ticks.set_visible(False)
bottom.set_xlabel('Volume')

sns.distplot(df['Adj Close'].dropna(), bins=50, color='purple')

# Set the labels
plt.figure(figsize=(12, 9))
sns.distplot(df['Adj Close'].dropna(), bins=50, color='purple')
pandas
Python Data Analysis Library
providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Source: http://pandas.pydata.org/
pandas Ecosystem

- **Statistics and Machine Learning**
  - Statsmodels
  - sklearn-pandas

- **Visualization**
  - Bokeh
  - yhat/ggplot
  - Seaborn
  - Vincent
  - IPython Vega
  - Plotly
  - Pandas-Qt

- **IDE**
  - IPython
  - quantopian/qgrid
  - Spyder

- **API**
  - pandas-datareader
  - quandl/Python
  - pydatastream
  - pandaSDMX
  - fredapi

- **Domain Specific**
  - Geopandas
  - xarray

- **Out-of-core**
  - Dask
  - Blaze
  - Odo

pandas-datareader

pandas-datareader

Up to date remote data access for pandas, works for multiple versions of pandas.

⚠️ Warning

As of v0.6.0 Yahoo!, Google Options, Google Quotes and EDGAR have been immediately deprecated due to large changes in their API and no stable replacement.

⚠️ Note

As of v0.6.0 Google finance is still functioning for historical price data, although there are frequent reports of failures. Failure is frequently encountered when bulk downloading historical price data.

Usage

Starting in 0.19.0, pandas no longer supports pandas.io.data or pandas.io.wb, so you must replace your imports from pandas.io with those from pandas_datareader:

```python
from pandas.io import data, wb
```

Many functions from the data module have been included in the top level API.

Get Financial Data Directly into Python

Get millions of financial and economic datasets from hundreds of publishers directly into Python.

Load Quandl Data Directly Into Python

All the Data You Want
Quandl unifies financial and economic datasets from hundreds of publishers on a single user-friendly platform.

Directly Into Python

https://www.quandl.com/tools/python
PyDatastream

PyDatastream 0.5.1

pip install PyDatastream

Python interface to the Thomson Reuters Dataworks Enterprise (Datastream) API

Project description

PyDatastream is a Python interface to the Thomson Dataworks Enterprise (DWE) SOAP API (non free), with some convenience functions for retrieving Datastream data specifically. This package requires valid credentials for this API.

For the documentation please refer to README.md inside the package or on the GitHub (https://github.com/vfillimonov/pydatastream/blob/master/README.md).

https://pypi.org/project/PyDatastream/
pandasSDMX: Statistical Data and Metadata eXchange in Python

pandaSDMX is an Apache 2.0-licensed Python client to retrieve and acquire statistical data and metadata disseminated in SDMX 2.1, an ISO-standard widely used by institutions such as statistics offices, central banks, and international organisations. pandaSDMX exposes datasets and related structural metadata including dataflows, code lists, and data structure definitions as pandas Series or multi-indexed DataFrames. Many other output formats and storage backends are available thanks to Odo.

Supported data providers

pandaSDMX ships with built-in support for the following agencies (others may be configured by the user):

- Australian Bureau of Statistics (ABS)
- European Central Bank (ECB)
- Eurostat
- French National Institute for Statistics (INSEE)
- Instituto Nacional de la Estadística y Geografía - INEGI (Mexico)
- International Monetary Fund (IMF) - SDMX Central only
- International Labour Organization (ILO)
- Italian statistics Office (ISTAT)
- Norges Bank (Norway)
- Organisation for Economic Cooperation and Development (OECD)
- United Nations Statistics Division (UNSD)
- UNESCO (free registration required)
- World Bank - World Integrated Trade Solution (WITS)

https://pandasdmx.readthedocs.io/en/latest/
Fred API

The FRED® API is a web service that allows developers to write programs and build applications that retrieve economic data from the FRED® and ALFRED® websites hosted by the Economic Research Division of the Federal Reserve Bank of St. Louis. Requests can be customized according to data source, release, category, series, and other preferences.

General Documentation

- Overview
- What is FRED®?
- What is ALFRED®?
- FRED® versus ALFRED®
- Real-Time Periods
- Errors

API

Categories

- fred/category – Get a category.
- fred/category/children – Get the child categories for a specified parent category.
- fred/category/related – Get the related categories for a category.
- fred/category/series – Get the series in a category.
- fred/category/tags – Get the tags for a category.
- fred/category/related_tags – Get the related tags for a category.

https://research.stlouisfed.org/docs/api/fred/
Python Pandas for Finance

Source: https://mapattack.wordpress.com/2017/02/12/using-python-for-stocks-1/
! pip install pandas_datareader

Collecting pandas_datareader
  Downloading https://files.pythonhosted.org/packages/cc/5c/ea5b6dcfd0f55c5f081e37f3b45335ec01c9e2199b9879339137f5ed269e0/pandas_datareader-0.7.0-py2.py3-none-any.whl (112kB)
  100% |-seeking| 112kB 2.7MB/s
Collecting lxml (from pandas_datareader)
  Downloading https://files.pythonhosted.org/packages/03/a4/9ee8035fc7c7670e5eab97f34f2ef00dd78ad491bf96df5acc6b0e63f5/lxml-4.2.5-cp36-none-any.whl (5.8MB 7.5MB/s)

Requirement already satisfied: pandas>=0.19.2 in /usr/local/lib/python3.6/dist-packages (from pandas_datareader) (0.22.0)
Requirement already satisfied: requests>=2.3.0 in /usr/local/lib/python3.6/dist-packages (from pandas_datareader) (2.18.4)
Requirement already satisfied: wrapt in /usr/local/lib/python3.6/dist-packages (from pandas_datareader) (1.10.11)
Requirement already satisfied: python-dateutil>=2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.19.2->pandas_datareader) (2.7.3)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.19.2->pandas_datareader) (1.14.6)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/dist-packages (from pandas>=0.19.2->pandas_datareader) (2018.5)
Requirement already satisfied: idna<=2.7,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests>=2.3.0->pandas_datareader) (2.5.0)
Requirement already satisfied: charset-normalize>=3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests>=2.3.0->pandas_datareader) (3.1.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests>=2.3.0->pandas_datareader) (2018.3.5)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests>=2.3.0->pandas_datareader) (1.22.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages (from python-dateutil>=2->pandas>=0.19.2->pandas_datareader) (1.11.0)
Installing collected packages: lxml, pandas-datareader
Successfully installed lxml-4.2.5 pandas-datareader-0.7.0
conda install pandas-datareader

```
[imyday-MacBook-Pro:~ imyday$ conda install pandas-datareader
Fetching package metadata ..........  
Solving package specifications: .  

Package plan for installation in environment /Users/imyday/anaconda:  

The following NEW packages will be INSTALLED:  

    pandas-datareader: 0.2.1-py36_0  
    requests-file: 1.4.1-py36_0  

Proceed ([y]/[n])? y

requests-file- 100% |################################| Time: 0:00:00 1.55 MB/s
pandas-datareader 100% |################################| Time: 0:00:00 409.66 kB/s

[imyday-MacBook-Pro:~ imyday$ conda list
# packages in environment at /Users/imyday/anaconda:
#
  _license         1.1                      py36_1
  alabaster        0.7.9                    py36_0
  anaconda         4.3.1                    np111py36_0
  anaconda-client  1.6.0                    py36_0
  anaconda-navigator 1.5.0                  py36_0
  anaconda-project 0.4.1                    py36_0
```
# !pip install pandas_datareader
import pandas_datareader.data as web
import datetime as dt

# Read Stock Data from Yahoo Finance
end = dt.datetime(2017, 12, 31)
start = dt.datetime(2016, 1, 1)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()
# !pip install pandas_datareader
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
%matplotlib inline

#Read Stock Data from Yahoo Finance
end = dt.datetime.now()
#start = dt.datetime(end.year-2, end.month, end.day)
start = dt.datetime(2016, 1, 1)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()
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') #df.index gives the dates
bottom.bar(df.index, df['Volume'])
# set the labels
top.axes.get_xaxis().set_visible(False)
top.set_title('AAPL')
top.set_ylabel('Adj Close')
bottom.set_ylabel('Volume')

plt.figure(figsize=(12,9))
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

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)

plt.show()
# !pip install pandas_datareader
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
%matplotlib inline

# Read Stock Data from Yahoo Finance
end = dt.datetime.now()
#start = dt.datetime(end.year-2, end.month, end.day)
start = dt.datetime(2016, 1, 1)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()

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), colspan=9)
bottom = plt.subplot2grid((12,9), (10,0), colspan=9)
top.plot(df.index, df['Adj Close'], color='blue')  # df.index gives the dates
bottom.bar(df.index, df['Volume'])

# set the labels
top.axes.get_xaxis().set_visible(False)
top.set_title('AAPL')
top.set_ylabel('Adj Close')
bottom.set_ylabel('Volume')

plt.figure(figsize=(12,9))
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
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)
plt.show()
# !pip install pandas_datareader
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt
%matplotlib inline

# Read Stock Data from Yahoo Finance
end = dt.datetime.now()
start = dt.datetime(end.year-2, end.month, end.day)
df = web.DataReader("AAPL", 'yahoo', start, end)
df.to_csv('AAPL.csv')
df.from_csv('AAPL.csv')
df.tail()

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')  # df.index gives the dates
bottom.bar(df.index, df['Volume'])

# set the labels
top.axes.get_xaxis().set_visible(False)
top.set_title('AAPL')
top.set_ylabel('Adj Close')
bottom.set_ylabel('Volume')

plt.figure(figsize=(12,9))
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
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)
plt.show()
# ! pip install quandl

```python
import quandl

# quandl.ApiConfig.api_key = "YOURAPIKEY"

df = quandl.get("WIKI/AAPL", start_date="2016-01-01", end_date="2017-12-31")

df.to_csv('AAPL.csv')

df.from_csv('AAPL.csv')

df.tail()
```

Source: [https://www.quandl.com/tools/python](https://www.quandl.com/tools/python)
Deep Learning with TensorFlow
Get started with TensorFlow

There are new tutorials to get started with TensorFlow using tf.keras and eager execution. Run the Colab notebooks directly in the browser.

TensorFlow 1.12 is here!

TensorFlow 1.12 is available, see the release notes for the latest updates.

Announcing TensorFlow.js

Learn about our JavaScript library for machine learning in the browser.

https://www.tensorflow.org/
Deep Learning Software

• TensorFlow
  – TensorFlow™ is an open source software library for high performance numerical computation.
• Keras
  – Deep Learning library for TensorFlow, CNTK
• PyTorch
  – An open source deep learning platform that provides a seamless path from research prototyping to production deployment.
• CNTK
  – Computational Network Toolkit by Microsoft Research

Source: http://deeplearning.net/software_links/
tf.keras

Keras:
High-level API for TensorFlow
Keras: The Python Deep Learning library

You have just found Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Keras is compatible with: Python 2.7-3.6.

http://keras.io/
FROM RESEARCH TO PRODUCTION

An open source deep learning platform that provides a seamless path from research prototyping to production deployment.

http://pytorch.org/
Keras is a high-level neural networks API

Written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

It was developed with a focus on enabling fast experimentation.

Being able to go from idea to result with the least possible delay is key to doing good research.

Source: https://keras.io/
Google TensorFlow

TensorFlow is an Open Source Software Library for Machine Intelligence

About TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

https://www.tensorflow.org/
TensorFlow Playground

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

http://playground.tensorflow.org/
TensorFlow is an Open Source Software Library for Machine Intelligence

https://www.tensorflow.org/
numerical computation using data flow graphs

https://www.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
Nodes:
mathematical operations

edges:
multidimensional data arrays
(tensors)
communicated between nodes
Computation is a Dataflow Graph

Graph of **Nodes**, also called **Operations** or **ops**.

- Bias
- Weights
- Examples
- Labels

- MatMul
- Add
- Relu
- Xent

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Computation is a Dataflow Graph

Edges are N-dimensional arrays: Tensors

bias
weights
inputs
targets

MatMul → Add → Relu → Xent

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Logistic Regression as Dataflow Graph

Nodes: bias, weights, inputs, targets, X, Y
Operations: MatMul, Add, Softmax, Xent
Ops: ops

Edges are N-dimensional arrays: Tensors

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Computation is a Dataflow Graph

with state

‘Biases’ is a variable

Some ops compute gradients

|--= updates biases

Source: Jeff Dean (2016), Large-Scale Deep Learning For Building Intelligent Computer Systems, WSDM 2016
Data Flow Graph

Source: https://www.tensorflow.org/
Data Flow Graph

Source: https://www.tensorflow.org/
Data Flow Graph
TensorFlow
Getting Started With TensorFlow

This guide gets you started programming in TensorFlow. Before using this guide, install TensorFlow. To get the most out of this guide, you should know the following:

- How to program in Python.
- At least a little bit about arrays.
- Ideally, something about machine learning. However, if you know little or nothing about machine learning, then this is still the first guide you should read.

TensorFlow provides multiple APIs. The lowest level API—TensorFlow Core—provides you with complete programming control. We recommend TensorFlow Core for machine learning researchers and others who require fine levels of control over their models. The higher level APIs are built on top of TensorFlow Core. These higher level APIs are typically easier to learn and use than TensorFlow Core. In addition, the higher level APIs make repetitive tasks easier and more consistent between different users. A high-level API like tf.estimator helps you manage data sets, estimators, training and inference.

This guide begins with a tutorial on TensorFlow Core. Later, we demonstrate how to implement the same model in tf.estimator. Knowing TensorFlow Core principles will give you a great mental model of how things are working internally when you use the more compact higher level API.

Source: https://www.tensorflow.org/get_started/get_started
Try your first TensorFlow

$ python

```python
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> sess.run(hello)
'Hello, TensorFlow!'
>>> a = tf.constant(10)
>>> b = tf.constant(32)
>>> sess.run(a+b)
42
```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
sess.run(hello)

b'Hello, TensorFlow!'
```python
import tensorflow as tf
sess = tf.Session()
a = tf.constant(10)
b = tf.constant(32)
sess.run(a+b)
```

42

https://github.com/tensorflow/tensorflow
```
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-3], dtype=tf.float32)

# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W*x + b
y = tf.placeholder(tf.float32)

# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares

# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]

# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
    sess.run(train, {x: x_train, y: y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: \$s b: \$s loss: \$s"%(curr_W, curr_b, curr_loss))

W: [-0.9999969] b: [ 0.99999082] loss: 5.69997e-11
```

Source: https://www.tensorflow.org/get_started/get_started
import numpy as np
import tensorflow as tf

feature_columns = [tf.feature_column.numeric_column("x", shape=[1])]

estimator = tf.estimator.LinearRegressor(feature_columns=feature_columns)

x_train = np.array([1., 2., 3., 4.])
y_train = np.array([0., -1., -2., -3.])
x_eval = np.array([2., 5., 8., 1.])
y_eval = np.array([-1.01, -4.1, -7, 0.])

input_fn = tf.estimator.inputs.numpy_input_fn(
    {"x": x_train}, y_train, batch_size=4, num_epochs=None, shuffle=True)

train_input_fn = tf.estimator.inputs.numpy_input_fn(
    {"x": x_train}, y_train, batch_size=4, num_epochs=1000, shuffle=False)

eval_input_fn = tf.estimator.inputs.numpy_input_fn(
    {"x": x_eval}, y_eval, batch_size=4, num_epochs=1000, shuffle=False)

estimator.train(input_fn=input_fn, steps=1000)

train_metrics = estimator.evaluate(input_fn=train_input_fn)
eval_metrics = estimator.evaluate(input_fn=eval_input_fn)

print("train metrics: \
train metrics: {'average_loss': 2.7210228e-07, 'loss': 1.0884091e-06, 'global_step': 1000}
eval metrics: {'average_loss': 0.0025725411, 'loss': 0.010290165, 'global_step': 1000}
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

```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

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

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

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

<table>
<thead>
<tr>
<th>X</th>
<th></th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>[10 20 30]</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>[20 30 40]</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>[30 40 50]</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>[40 50 60]</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>[50 60 70]</td>
<td></td>
<td>80</td>
</tr>
<tr>
<td>[60 70 80]</td>
<td></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
# split a univariate sequence into samples
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://colab.research.google.com/drive/1aEK0eSev8Q-Y0nNY32geFk7CB8pVgSQM

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

```python
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
import matplotlib.pyplot as plt

# split a univariate sequence into samples

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)
# 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=(n_steps, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

# fit model
history = model.fit(X, y, epochs=500, verbose=0)

# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, 1))
yhat = model.predict(x_input, verbose=0)
print(yhat)
print('yhat', yhat)
print(model.summary())
```

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
AI + VDI
POC
AI + VDI POS
TensorFlow Models

• M1: Basic Classification (Image Classification) (65 Seconds)

• M2: Basic Text Classification (Text Classification) (46 Seconds)

• M3: Basic Regression (Predict House Prices) (43 Seconds)

• M4: Pix2Pix Eager (Option) (7-8 Hours)

• M5. NMT with Attention (Option) (20-30 minutes)
Basic Classification

Fashion MNIST Image Classification

https://colab.research.google.com/drive/19PJOji1vn1kjcutlZNHjRSLbevl4kd5z

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.

Copyright 2018 The TensorFlow Authors.

Copyright 2018 The TensorFlow Authors.

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

MIT License

Train your first neural network: basic classification

Import the Fashion MNIST dataset

Explore the data

Preprocess the data

Build the model

Setup the layers

Compile the model

Train the model

Evaluate accuracy

Make predictions

Source: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/basic_classification.ipynb
Text Classification

IMDB Movie Reviews

https://colab.research.google.com/drive/1x16h1GhHsLIrLYtPCvCHaoO1W-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

Text classification with 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.

Copyright 2018 The TensorFlow Authors.
Basic Regression
Predict House Prices

https://colab.research.google.com/drive/1v4c8ZHTnRtg2_25K_AURjR6SCVBRdlj

Table of contents

Copyright 2018 The TensorFlow Authors.

Predict house prices: regression

The Boston Housing Prices dataset

Examples and features

Labels

Normalize features

Create the model

Train the model

Predict

Conclusion

Copyright 2018 The TensorFlow Authors.

→ 2 cells hidden

Predict house prices: regression

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 predict a discrete label (for example, where a picture contains an apple or an orange).

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
!

!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: 
print("GPU RAM Free: {0:.0f}MB | Used: {1:.0f}MB | Util {2:.0f}% | Total {3:.0f}MB".format(gpu.memory_free, gpu.memory_used, gpu.memory_percent, gpu.memory_total))
```
AI+VDI POC
ISAC+TKU Test

• AI+VDI POC Folder (3+1 ipynb) (v3.0.20181120)
  – https://drive.google.com/open?id=1qHOemktbEmUz-ot8eFxIKbGwJvXlrjtc

• run3models.ipynb
  – https://colab.research.google.com/drive/1HQ1GrqQUUPCct7_AVgoMwMrh0UqMm0f
Summary

• Machine Learning

• Deep Learning
References

- Luis Serrano (2017), A friendly introduction to Convolutional Neural Networks and Image Recognition, https://www.youtube.com/watch?v=2-Ol7ZB0MmU
- Martin Gorner (2017), TensorFlow and Deep Learning without a PhD, Part 2 (Google Cloud Next '17), https://www.youtube.com/watch?v=fTUwdXUFFl8
- 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=aircAruvnKk
- 3Blue1Brown (2017), Gradient descent, how neural networks learn | Chapter 2, deep learning, https://www.youtube.com/watch?v=IHZwWFHWa-w
- 3Blue1Brown (2017), What is backpropagation really doing? | Chapter 3, deep learning, https://www.youtube.com/watch?v=lIg3gGewQ5U
- TensorFlow: https://www.tensorflow.org/
- Keras: http://keras.io/
- Udacity, Deep Learning, https://www.youtube.com/playlist?list=PLAwxTw4SYaPn_OWPFmT9uIXLbQrImzHfOY
- https://github.com/leriomaggio/deep-learning-keras-tensorflow