

Machine Learning with Python – SciKit-learn Part 3

Ramesh Shankar
UConn

Dimensionality reduction - PCA

- Principal Component Analysis – unsupervised data compression
 - Can help with computational efficiency – use lesser data for similar effect
 - Can also help reduce curse of dimensionality:
 - More dimensions – more volume of space – data becomes sparse
 - Therefore, it hinders efficient prediction or classification
 - PCA converts a set of possibly correlated variables (d variables) into fewer orthogonal (uncorrelated) variables (k variables):

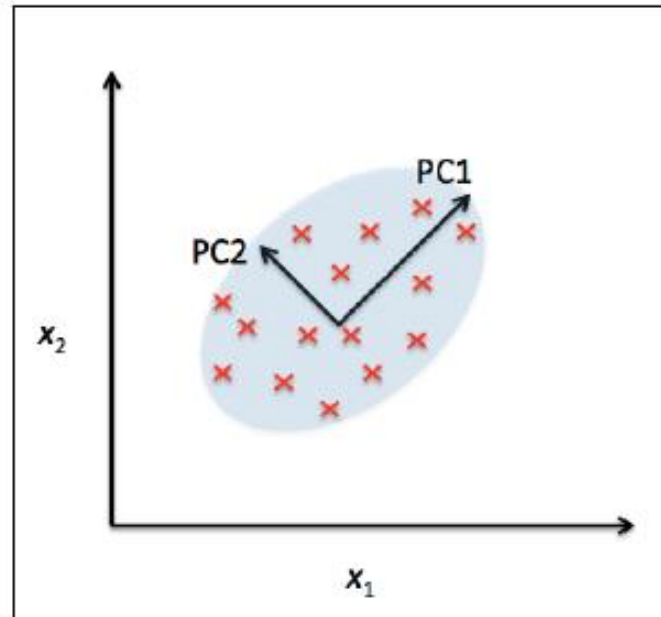
$$\mathbf{x} = [x_1, x_2, \dots, x_d], \quad \mathbf{x} \in \mathbb{R}^d$$

$$\downarrow \mathbf{x}W, \quad W \in \mathbb{R}^{d \times k}$$

$$\mathbf{z} = [z_1, z_2, \dots, z_k], \quad \mathbf{z} \in \mathbb{R}^k$$

PCA

- These principal components are the “directions of maximum variance” in the dataset:



Wine dataset

```
In [6]: df_wine = pd.read_csv('https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/wine/wine.data')  
  
df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',  
                  'Alcalinity of ash', 'Magnesium', 'Total phenols',  
                  'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', |  
                  'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']  
df_wine.head()
```

Out[6]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

PCA - steps

- Standardize the d-dimensional dataset
- Construct covariance matrix:

$$\sigma_{jk} = \frac{1}{n} \sum_{i=1}^n (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k) \quad \Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 \end{bmatrix}$$

- Obtain the eigenvectors (which are the principal components – the directions of maximum variance)
- And the eigenvalues (the scalar magnitudes of the eigenvectors)

Standardize the d-dimensional dataset

```
5 #Download data
6 df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
7 ..... 'machine-learning-databases/wine/wine.data',
8 ..... header=None)
9 df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
10 'Alcalinity of ash', 'Magnesium', 'Total phenols',
11 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
12 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline']
13 df_wine.head()
14
15 #Split data into train-test
16 from sklearn.model_selection import train_test_split
17 X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values
18 X_train, X_test, y_train, y_test = \
19 ..... train_test_split(X, y, test_size=0.3, random_state=0)
20
21 #Standardize the data
22 from sklearn.preprocessing import StandardScaler
23 sc = StandardScaler()
24 X_train_std = sc.fit_transform(X_train)
25 X_test_std = sc.transform(X_test)
```

Construct covariance matrix, obtain eigen values

```
27 #Obtain eigenvalues
28 import numpy as np
29 cov_mat = np.cov(X_train_std.T)
30 eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
31
32 print('\nEigenvalues\n%s' % eigen_vals)
```

```
Eigenvalues
[ 4.8923083  2.46635032  1.42809973  1.01233462  0.84906459  0.60181514
 0.52251546  0.08414846  0.33051429  0.29595018  0.16831254  0.21432212
 0.2399553 ]
```

```
In [14]: eigen_vecs[0]
Out[14]:
array([ 1.46698114e-01,  5.04170789e-01, -1.17235150e-01,
        2.06254611e-01, -1.87815947e-01, -1.48851318e-01,
       -1.79263662e-01, -5.54687162e-02, -4.03054922e-01,
       -4.17197583e-01,  2.75660860e-01,  4.03567189e-01,
        4.13320786e-04])
```

```
In [16]: eigen_vals[0]
Out[16]: 4.8923083032737411
```

Computing a graph of variance explained

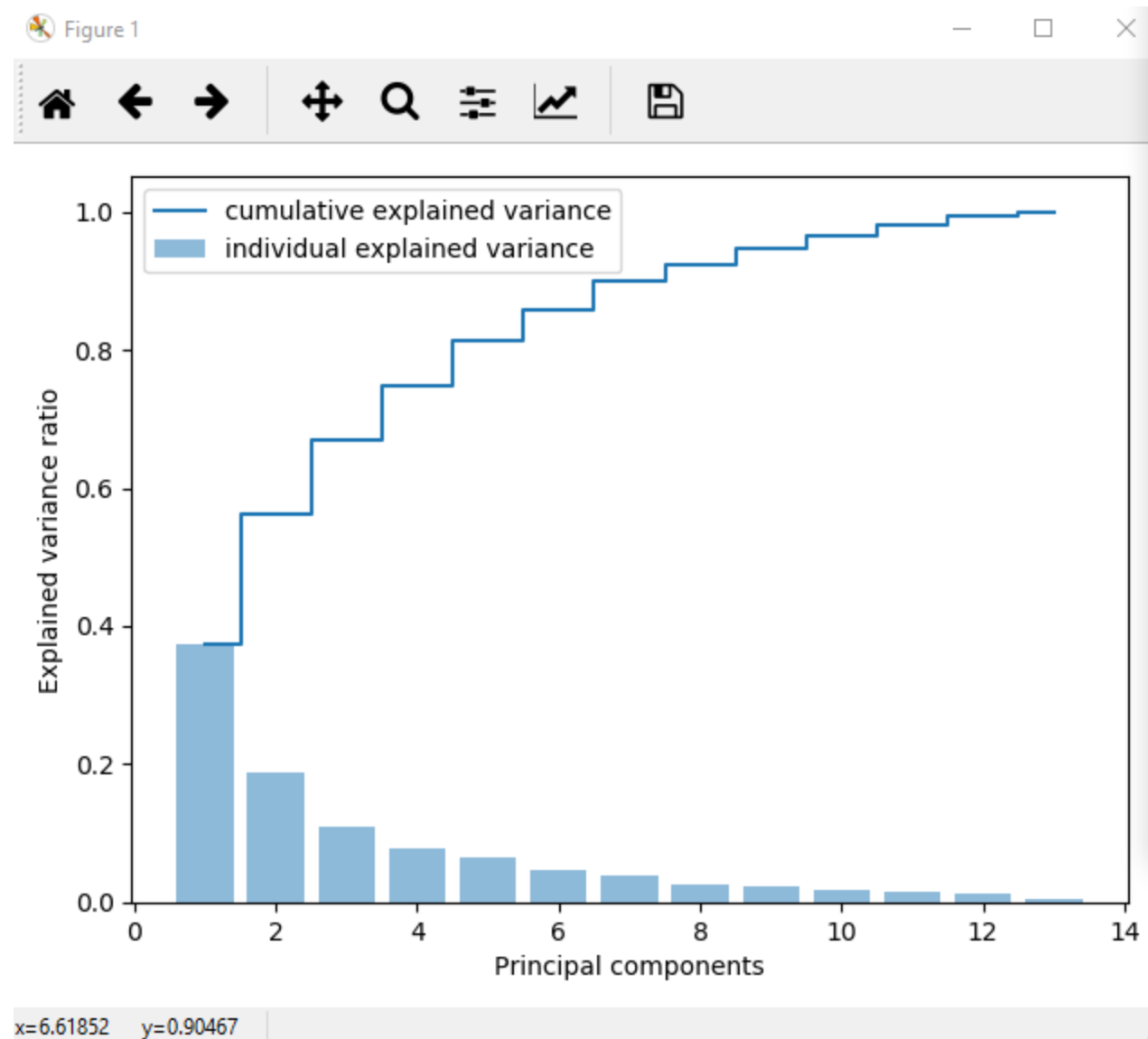
Variance explained ratio:

$$\frac{\lambda_j}{\sum_{j=1}^d \lambda_j}$$

```
34 # Total and explained variance
35 tot = sum(eigen_vals)
36 var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
37 cum_var_exp = np.cumsum(var_exp)
```

```
39 # Plot figure
40 import matplotlib.pyplot as plt
41 plt.bar(range(1, 14), var_exp, alpha=0.5, align='center',
42         label='individual explained variance')
43 plt.step(range(1, 14), cum_var_exp, where='mid',
44         label='cumulative explained variance')
45 plt.ylabel('Explained variance ratio')
46 plt.xlabel('Principal components')
47 plt.legend(loc='best')
48 plt.tight_layout()
49 # plt.savefig('./figures/pca1.png', dpi=300)
50 plt.show()
51
```


Plot cumulative explained variance



Feature transformation

Sort eigen vectors in descending order of eigen values:

```
In [17]: eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:, i])
...:                     for i in range(len(eigen_vals))]
...:
...: # Sort the (eigenvalue, eigenvector) tuples from high to low
...: eigen_pairs.sort(key=lambda k: k[0], reverse=True)
...:
```

Construct pairs of eigen vectors (eigen_pairs):

```
In [18]: w = np.hstack((eigen_pairs[0][1][:, np.newaxis],
...:                     eigen_pairs[1][1][:, np.newaxis]))
...: print('Matrix W:\n', w)
...:
```

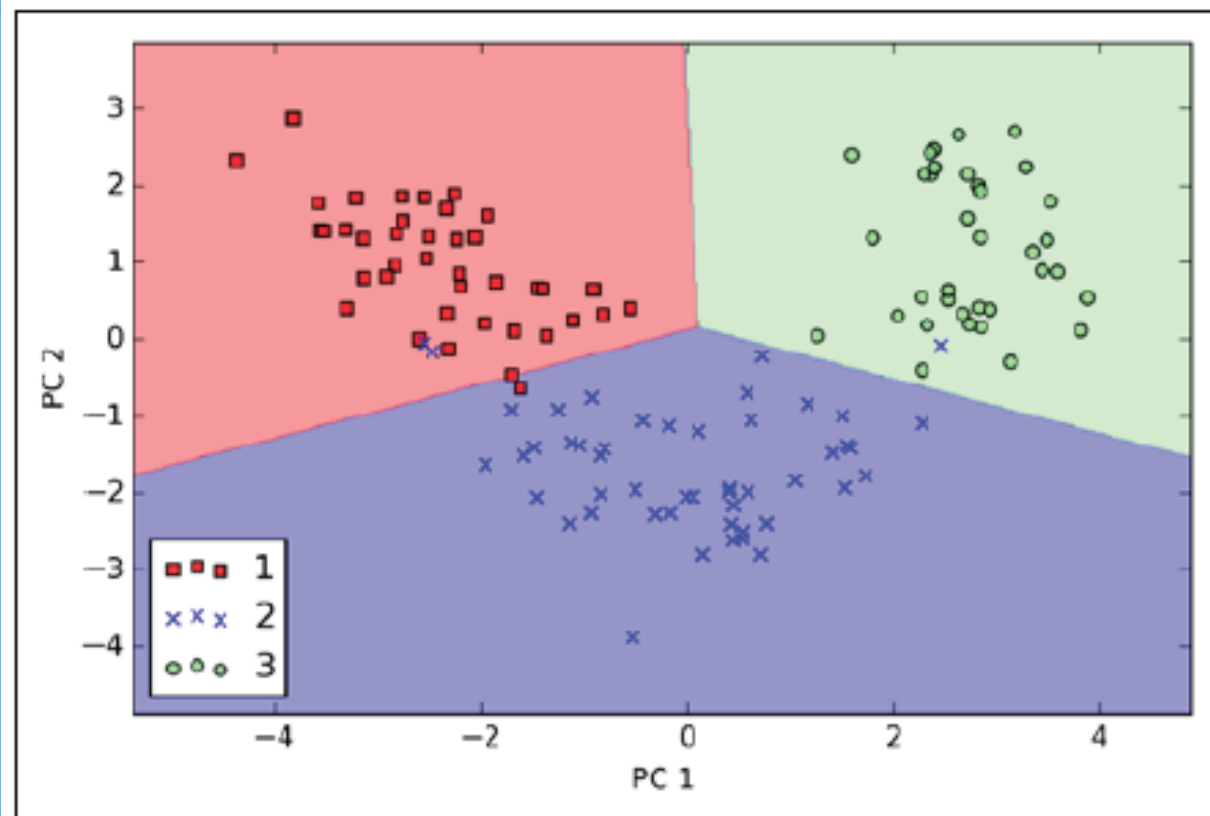
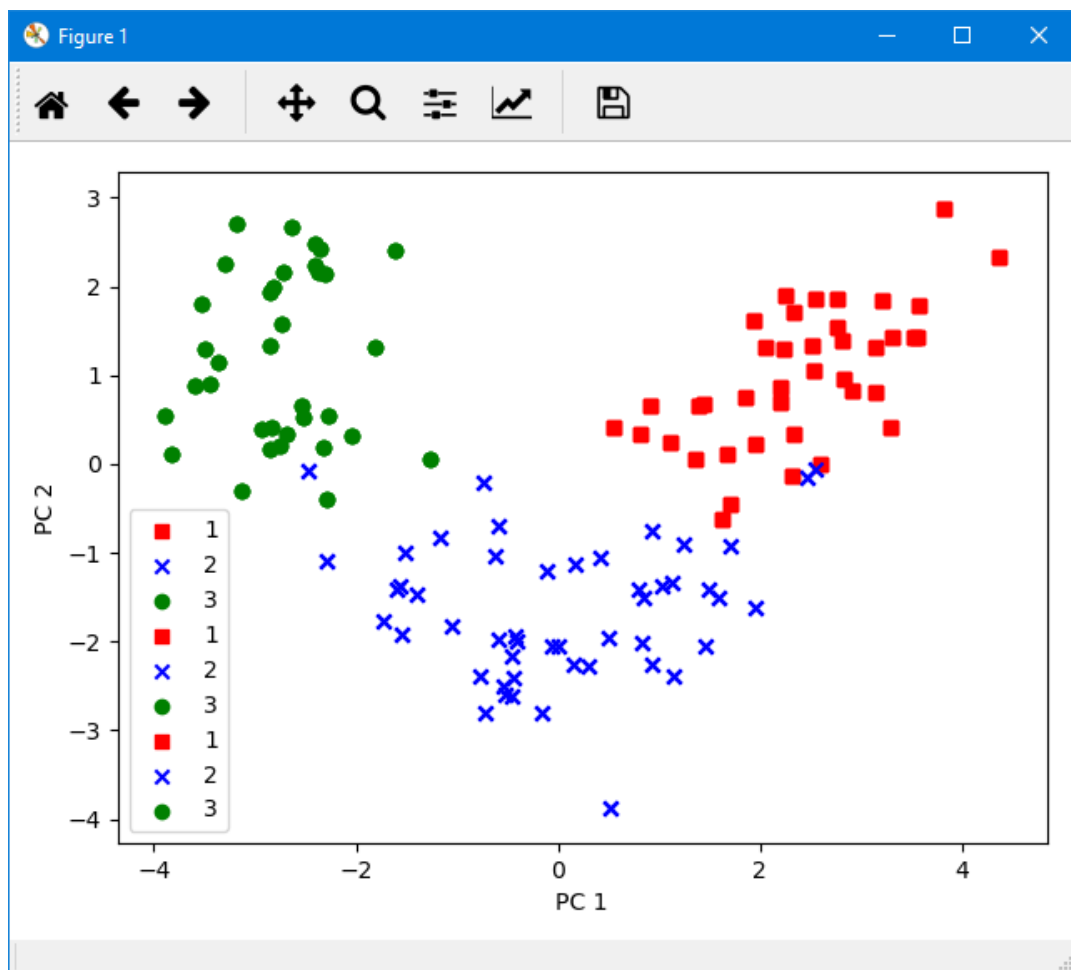
```
In [20]: eigen_pairs
Out[20]:
[(4.8923083032737411,
  array([ 0.14669811, -0.24224554, -0.02993442,
         0.38934455,  0.42326486, -0.30634956,
         0.30032535,  0.36821154,  0.29259713]),
 (2.4663503157592306,
  array([ 0.50417079,  0.24216889,  0.28698484,
         0.09363991,  0.01088622,  0.01870216,
        -0.27924322, -0.174365  ,  0.36315461]),
 (1.4280997275048442,
  array([-0.11723515,  0.14994658,  0.65639172,
         0.18080442,  0.14295933,  0.17223111,
         0.25519002, -0.06468718,  0.12079772]),
 (0.54527081,
  array([ 0.03040352,  0.00986919,  0.03057221,
         0.00986919,  0.03040352,  0.00986919,
         0.03040352,  0.00986919,  0.03040352]),
 (0.00011111,
  array([ 0.00011111,  0.00011111,  0.00011111,
         0.00011111,  0.00011111,  0.00011111,
         0.00011111,  0.00011111,  0.00011111])])
```

```
Matrix W:
[[ 0.14669811  0.50417079]
 [-0.24224554  0.24216889]
 [-0.02993442  0.28698484]
 [-0.25519002 -0.06468718]
 [ 0.12079772  0.22995385]
 [ 0.38934455  0.09363991]
 [ 0.42326486  0.01088622]
 [-0.30634956  0.01870216]
 [ 0.30572219  0.03040352]
 [-0.09869191  0.54527081]
 [ 0.30032535 -0.27924322]
 [ 0.36821154 -0.174365  ]
 [ 0.29259713  0.36315461]]
```

Create the new features:

$$x' = xW$$

```
60 # Create the new features with our eigen vectors
61 X_train_pca = X_train_std.dot(w) ←
62 colors = ['r', 'b', 'g']
63 markers = ['s', 'x', 'o']
64
65 import matplotlib.pyplot as plt
66
67 for l, c, m in zip(np.unique(y_train), colors, markers):
68     plt.scatter(X_train_pca[y_train == l, 0],
69                 X_train_pca[y_train == l, 1],
70                 c=c, label=l, marker=m)
71
72 plt.xlabel('PC 1')
73 plt.ylabel('PC 2')
74 plt.legend(loc='lower left')
75 plt.tight_layout()
76 # plt.savefig('./figures/pca2.png', dpi=300)
77 plt.show()
```



PCA in Scikit-learn

```
# PCA in Scikit-learn:

from sklearn.decomposition import PCA
pca = PCA()
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_

import matplotlib.pyplot as plt
plt.bar(range(1, 14), pca.explained_variance_ratio_, alpha=0.5, align='center')
plt.step(range(1, 14), np.cumsum(pca.explained_variance_ratio_), where='mid')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.show()

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1])
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()
```

Streamlining workflows with pipelines

- Can add different preprocessing techniques into a pipeline – e.g.
 - Standardization for feature scaling
 - Data compression – e.g. principal component analysis – for dimensionality reduction
- Example: Wisconsin Breast Cancer dataset
 - 569 samples of malignant and benign tumor cells
 - Column 1: Unique ID of sample
 - Column 2: diagnosis (M = malignant, B = benign)
 - Columns 3 – 32: 30 real-value features computed from digitized images of cell nuclei – can be used to build model to predict whether tumor B or M

Loading dataset

```
# Import Wisconsin Breast Cancer dataset
import pandas as pd

# Read dataset from website
df = pd.read_csv('https://archive.ics.uci.\
edu/ml/machine-learning-databases/breast-cancer-\
wisconsin/wdbc.data', header=None)
```

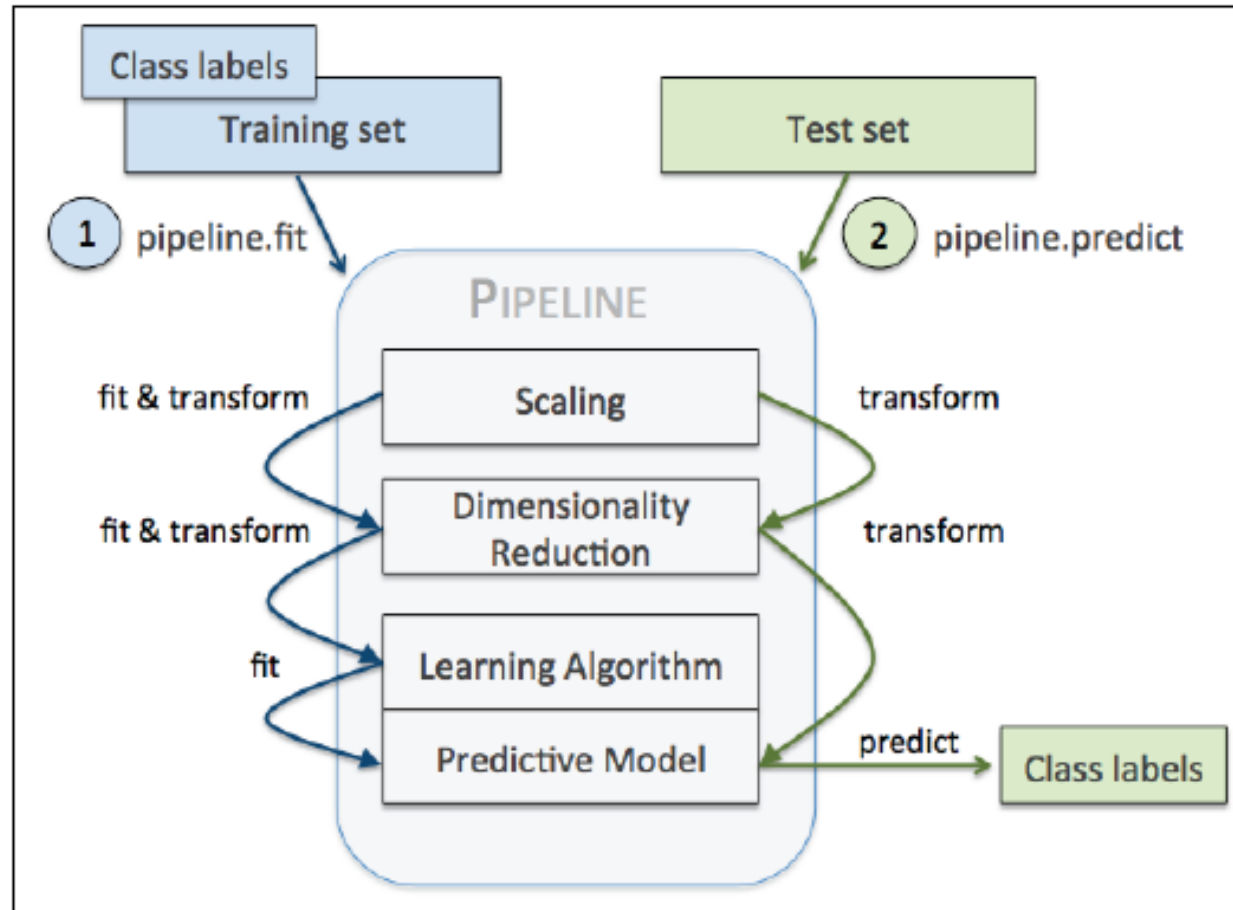
```
# Assign columns 2 through 30 to the variable X
X = df.loc[:, 2:].values
# Assign column 1 to the variable y
y = df.loc[:, 1].values
# Using LabelEncoder, transform class labels to integers
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
# The above transforms M as 1, B as 0
# Check that using this command:
le.transform(['M', 'B'])
```

```
# Divide dataset into separate training (80%)  
# and test dataset  
from sklearn.cross_validation import train_test_split  
X_train, X_test, y_train, y_test = \  
train_test_split(X, y, test_size=0.20, random_state=1)
```


Combining steps into pipeline

- Standardize columns to ensure they are on same scale
- Reduce dimensionality of 30 attributes using Principal Component Analysis (PCA)
- Pipeline:
 - (1) Identifier
 - (2) Transformer or estimator

```
# Combine standardization and PCA
# into one pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', LogisticRegression(random_state=1))])
pipe_lr.fit(X_train, y_train)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```



Cross-validation to assess model performance

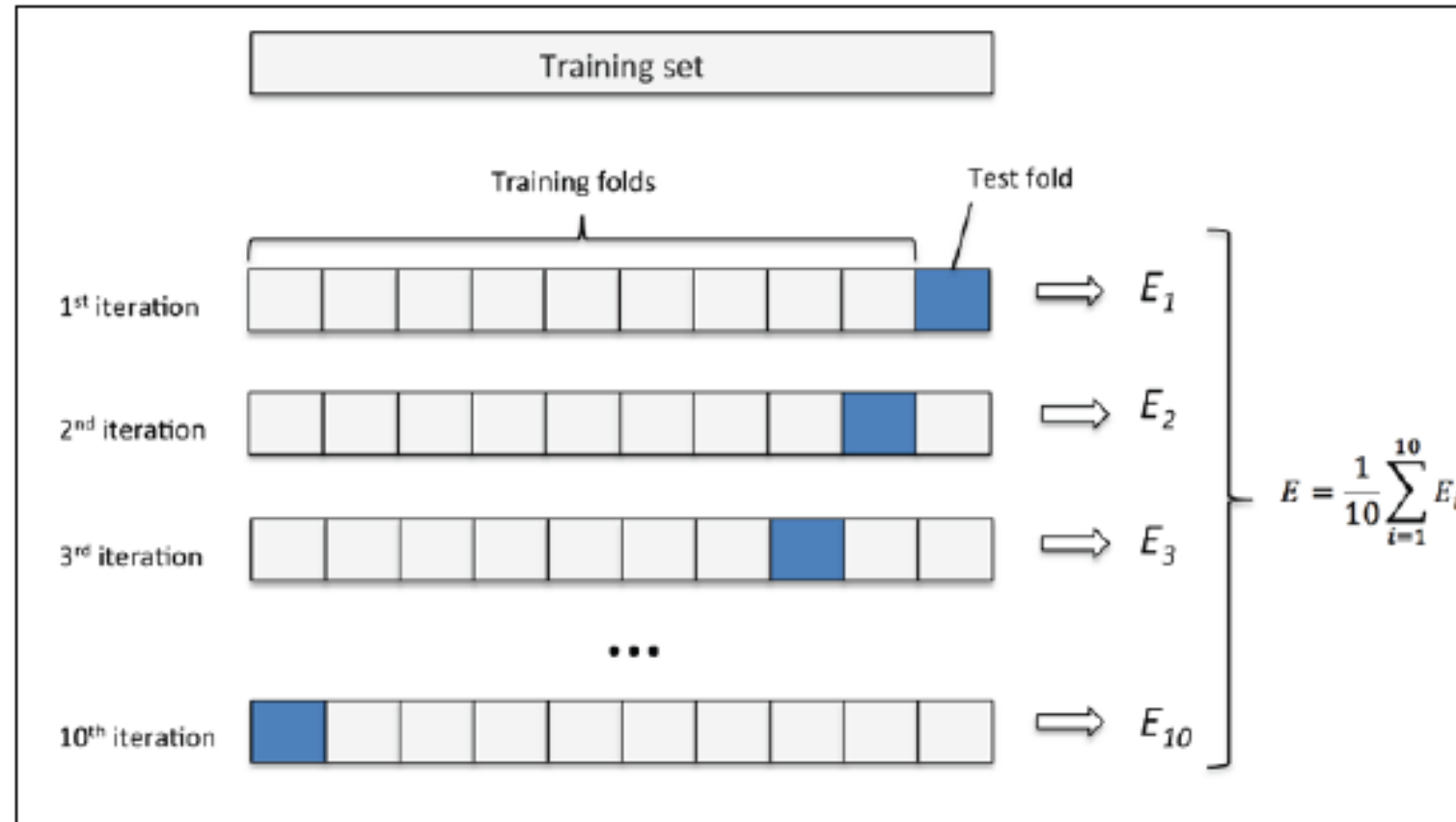
- Key goal: estimate performance on data the model has not seen
- Model can suffer from underfitting (high bias) or overfitting (high variance)
- To find acceptable bias-variance tradeoff, evaluate model
- Hold-out cross-validation, K-fold cross-validation

Holdout method

- Split initial dataset into training, validation and test datasets
- Use training dataset for model training
- Use validation set for model selection
 - Select optimal values of tuning parameters (hyperparameters)
- Use test dataset to estimate performance – test ability of model to generalize to new data
- Disadvantage: performance estimate is sensitive to how we partition training set into training and validation subsets.

K-fold cross validation

- Randomly split training dataset into k folds without replacement
 - Typically $k = 10$
- Use $k-1$ folds for model training
- Use 1 fold for validation
- Calculate average performance of models based on different folds – performance estimate less sensitive to data partitioning
- Perform model tuning – find hyperparameters with satisfactory general performance
- Then, retrain model on complete training set, obtain performance estimate using independent test set



K-fold cross validation

- Stratified k-fold cross-validation: yields better bias and variance estimates. (Kohavi et al., 1995).
- Each fold is representative of the class-proportions of the training dataset.

```
# Stratified K-fold cross-validation
import numpy as np
from sklearn.model_selection import StratifiedKFold
# Initialize the StratifiedKFold iterator
# with class-labels y_train in training set
skf = StratifiedKFold(n_splits=10)
skf.get_n_splits(X_train, y_train)

# Define array "scores"
scores = []
# Loop through each of the k folds
k = 0
for train, test in skf.split(X_train, y_train):
    k += 1
    pipe_lr.fit(X_train[train], y_train[train])
    score = pipe_lr.score(X_train[test], y_train[test])
    scores.append(score)
    print('Fold: %s, Class dist.: %s, Acc: %.3f' % (k,
                                                    np.bincount(y_train[train]), score))
```

Alternate method:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator=pipe_lr,
                          X=X_train,
                          y=y_train,
                          cv=10,
                          n_jobs=1)
print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```


Fine-tuning models via grid search

- Two types of parameters
 - Those learned from data
 - Those we specify – hyperparameters (e.g. regularization parameter in Logistic Regression)
- How to choose optimum hyperparameters?
 - Grid Search

```
# Initialize GridSearchCV object from
# sklearn.grid_search module
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC

# Create pipeline for SVM
pipe_svc = Pipeline([('scl', StandardScaler()),
                      ('clf', SVC(random_state=1))])
# Set parameter range
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
# Set param_grid to a list of
# dictionaries - specify parameters to tune
param_grid = [{'clf__C': param_range,
               'clf__kernel': ['linear']},
               {'clf__C': param_range,
               'clf__gamma': param_range,
               'clf__kernel': ['rbf']}]
```

```
# set up k-fold cross-validation
gs = GridSearchCV(estimator=pipe_svc,
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)

# Use training data to perform grid search
gs = gs.fit(X_train, y_train)

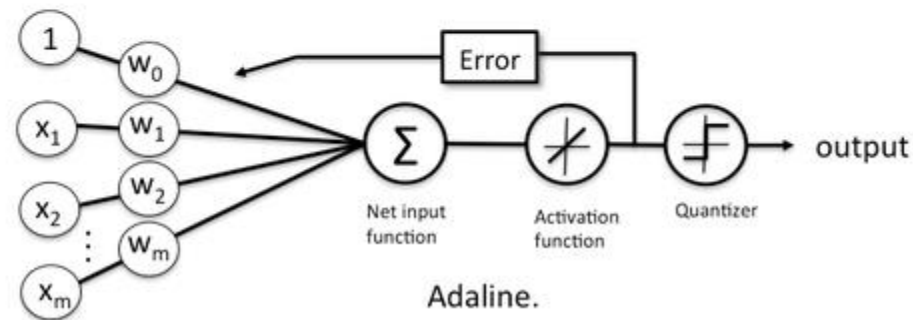
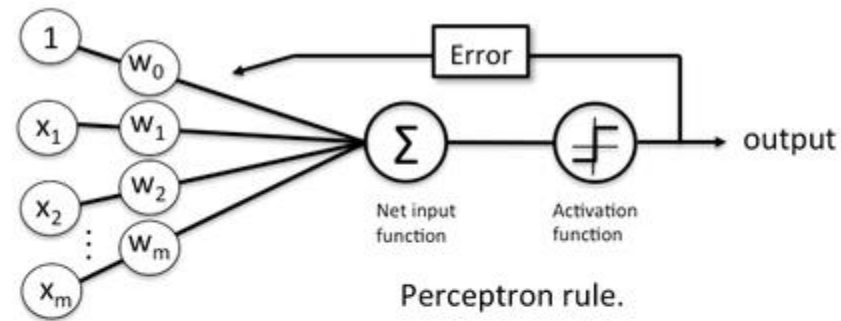
# Obtain the score of the best-performing model
print(gs.best_score_)

# print the parameters of the best-performing model
print(gs.best_params_)

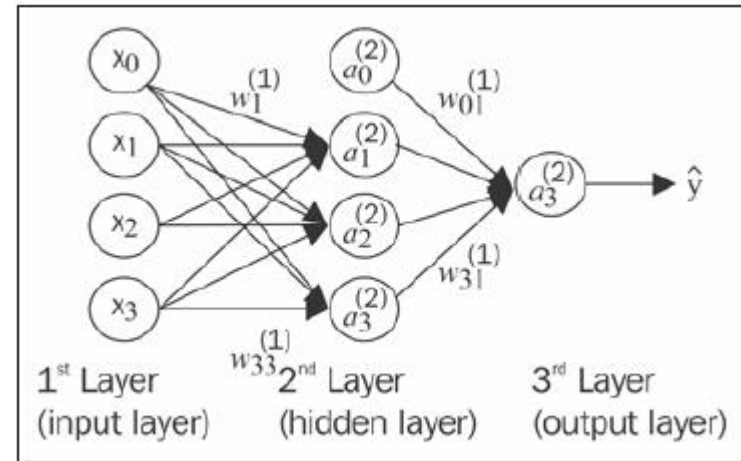
# Use independent test dataset
# to estimate performance of best-selected model
clf = gs.best_estimator_
clf.fit(X_train, y_train)
print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

Introduction to Deep Learning

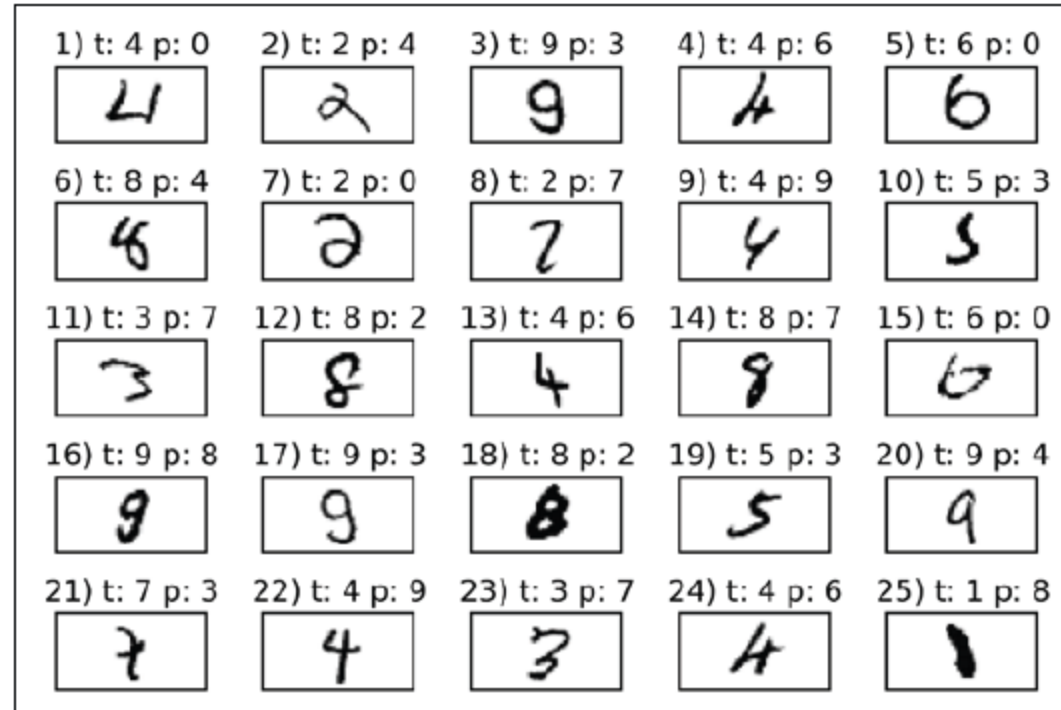
- Perceptron vs Adaline (single layer neural network)



Multi-layer neural network



MNIST dataset

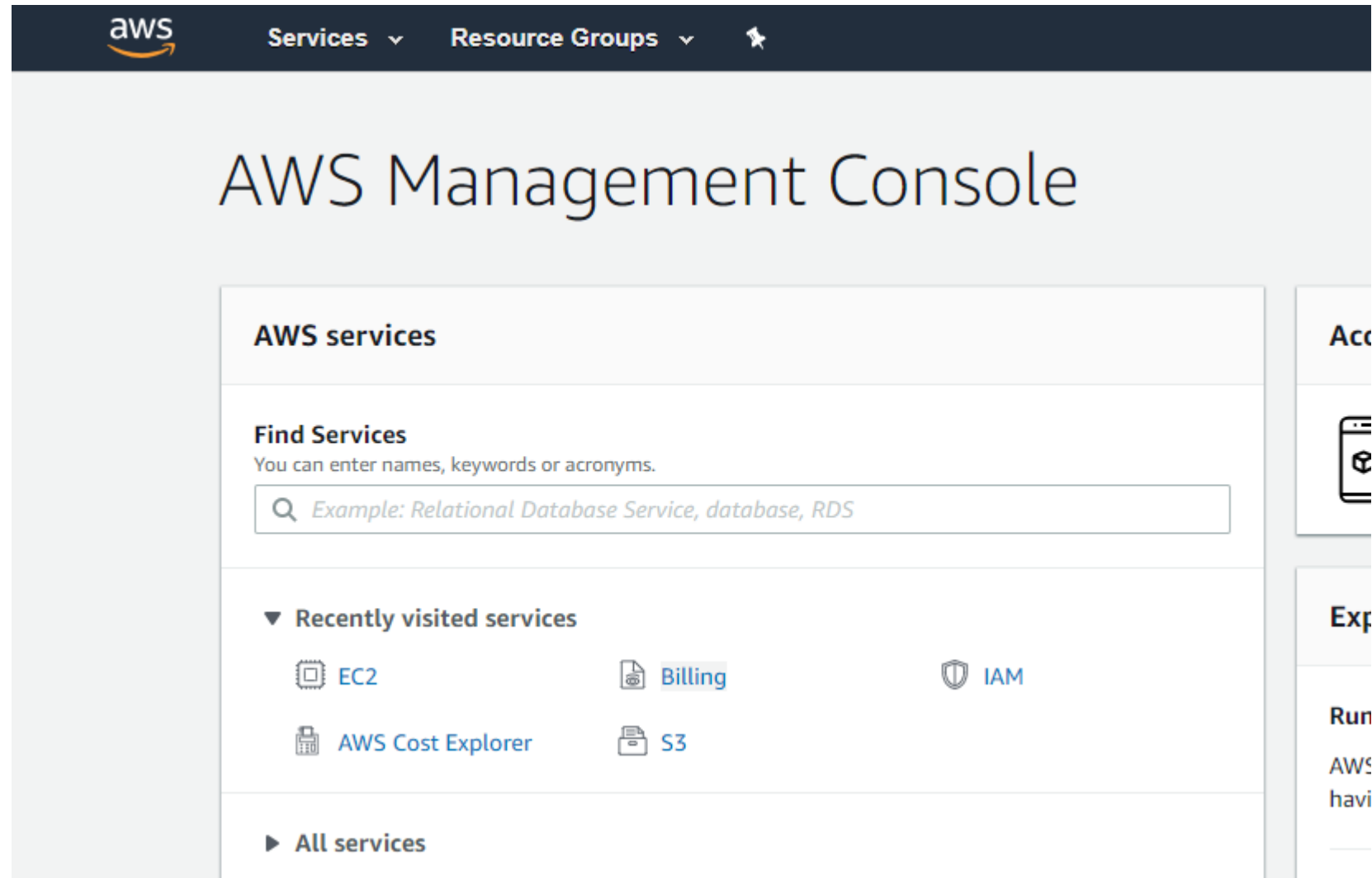


TensorFlow or Keras?

- <https://medium.com/implodinggradients/tensorflow-or-keras-which-one-should-i-learn-5dd7fa3f9ca0>

Deep learning with Python on AWS EC2

- Login:



Navigate to EC2

The screenshot displays the AWS Management Console interface for the EC2 service. The top navigation bar includes the AWS logo, 'Services', and 'Resource Groups'. The left-hand navigation pane is titled 'EC2 Dashboard' and lists various options: Events, Tags, Reports, Limits, INSTANCES (with a sub-menu for Instances, Launch Templates, Spot Requests, Reserved Instances, Dedicated Hosts, Scheduled Instances, and Capacity Reservations), IMAGES (with sub-menus for AMIs and Bundle Tasks), and ELASTIC BLOCK STORE (with sub-menus for Volumes, Snapshots, and Lifecycle Manager). The main content area is titled 'Resources' and shows a summary of EC2 resources in the US East (N. Virginia) region: 1 Running Instances, 0 Elastic IPs, 0 Dedicated Hosts, 4 Snapshots, 4 Volumes, 0 Load Balancers, 3 Key Pairs, 33 Security Groups, and 0 Placement Groups. Below this is a blue box with a link to 'Learn more about the latest in AWS Compute from AWS re:Invent by viewing the EC2 Videos.' The 'Create Instance' section follows, with a description and a 'Launch Instance' button. A red arrow points to this button. Below the button is a note about the region. The bottom section is split into 'Service Health' (showing 'Service Status' as 'US East (N. Virginia):' with a green checkmark) and 'Scheduled Events' (showing 'US East (N. Virginia):' with 'No events').

aws Services Resource Groups

EC2 Dashboard

- Events
- Tags
- Reports
- Limits
- INSTANCES
 - Instances
 - Launch Templates
 - Spot Requests
 - Reserved Instances
 - Dedicated Hosts
 - Scheduled Instances
 - Capacity Reservations
- IMAGES
 - AMIs
 - Bundle Tasks
- ELASTIC BLOCK STORE
 - Volumes
 - Snapshots
 - Lifecycle Manager

Resources

You are using the following Amazon EC2 resources in the US East (N. Virginia) region:

- 1 Running Instances
- 0 Elastic IPs
- 0 Dedicated Hosts
- 4 Snapshots
- 4 Volumes
- 0 Load Balancers
- 3 Key Pairs
- 33 Security Groups
- 0 Placement Groups

Learn more about the latest in AWS Compute from AWS re:Invent by viewing the [EC2 Videos](#).

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the US East (N. Virginia) region

Service Health

Service Status:

US East (N. Virginia):

Scheduled Events

US East (N. Virginia):

No events

Choose the appropriate AMI

1. Choose AMI2. Choose Instance Type3. Configure Instance4. Add Storage5. Add Tags6. Configure Security Group7. Review

Step 1: Choose an Amazon Machine Image (AMI)

Cancel and Exit

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

deep learning

Quick Start (5)

My AMIs (0)

AWS Marketplace (124)

Community AMIs (221)

☐ Free tier only ⓘ

Deep Learning AMI (Ubuntu) Version 21.2 - ami-0d96d570269578cd7

MXNet-1.3, TensorFlow-1.12, PyTorch-1.0, Keras-2.2, Chainer-5.2, Caffe/2-0.8, Theano-1.0 & CNTK-2.6, configured with NVIDIA CUDA, cuDNN, NCCL, Intel MKL-DNN, Docker & NVIDIA-Docker. For a fully managed experience, check: <https://aws.amazon.com/sagemaker>

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Deep Learning AMI (Amazon Linux) Version 21.2 - ami-087379093eeda94ae

MXNet-1.3, TensorFlow-1.12, PyTorch-1.0, Keras-2.2, Chainer-5.1, Caffe/2-0.8, Theano-1.0 & CNTK-2.6, configured with NVIDIA CUDA, cuDNN, NCCL, Intel MKL-DNN, Docker & NVIDIA-Docker. For a fully managed experience, check: <https://aws.amazon.com/sagemaker>

1 to 5 of 5 AMIs

Select

64-bit (x86)

Select

64-bit (x86)

Choose the appropriate instance type (GPU)

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Gr

Step 2: Choose an Instance Type

<input type="checkbox"/>	GPU instances	g3.4xlarge	16	122	EBS only
<input type="checkbox"/>	GPU instances	g3.8xlarge	32	244	EBS only
<input type="checkbox"/>	GPU instances	g3.16xlarge	64	488	EBS only
<input checked="" type="checkbox"/>	GPU instances	p2.xlarge	4	61	EBS only
<input type="checkbox"/>	GPU instances	p2.8xlarge	32	488	EBS only
<input type="checkbox"/>	GPU instances	p2.16xlarge	64	732	EBS only
<input type="checkbox"/>	GPU instances	p3.2xlarge	8	61	EBS only
<input type="checkbox"/>	GPU instances	p3.8xlarge	32	244	EBS only
<input type="checkbox"/>	GPU instances	p3.16xlarge	64	488	EBS only

Click “review and launch”

Launching:

1. Choose a SSH key – existing pair or create a new pair
2. Choose the name
3. Acknowledge (check box)
4. Launch instance

Select an existing key pair or create a new key pair ×

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

1

Choose an existing key pair

Select a key pair

2

msbapm_hadoop_trial1

3

☐ I acknowledge that I have access to the selected private key file (msbapm_hadoop_trial1.pem), and that without this file, I won't be able to log into my instance.

4

Cancel

Launch Instances

Login via Putty

The screenshot displays the AWS Management Console interface. On the left, the navigation sidebar includes sections for EC2 Dashboard, INSTANCES, IMAGES, ELASTIC BLOCK STORE, and NETWORK & SECURITY. The 'Instances' section is highlighted with a red circle '1'. The main content area shows a list of EC2 instances. The instance 'ras1' with ID 'i-03316b0049132c4e4' is selected, indicated by a red circle '2'. Below the list, the instance details for 'i-03316b0049132c4e4' are shown, including its state (running) and type (p2.xlarge). On the right, a table shows status checks for the instance, indicating that 2/2 checks passed. Below this, a section for network information shows the public IP address (54.210.235.92) and other network details. A red circle '3' highlights the public IP address.

Instances List:

Name	Instance ID
ras1	i-017b932bfb3fa54c8
	i-0c1e8beb5a3652302
	i-0e1dc7ff529f310e6
	i-03316b0049132c4e4

Instance Details for i-03316b0049132c4e4:

Property	Value
Instance ID	i-03316b0049132c4e4
Instance state	running
Instance type	p2.xlarge
Elastic IPs	
Availability zone	us-east-1b

Status Checks:

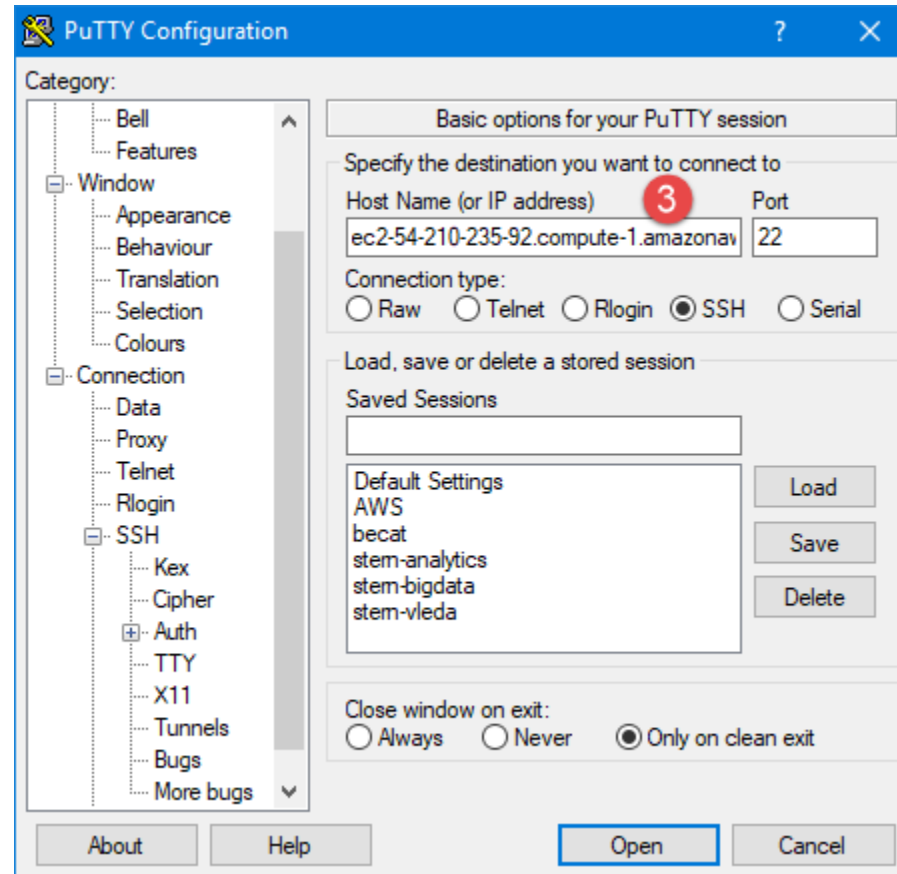
Status Checks	Alarm Status	Public DNS (IP)
	None	
	None	
	None	
2/2 checks passed	None	ec2-54-210-235-92.compute-1.amazonaws.com

Network Information:

Property	Value
Public IP	54.210.235.92
Private IP	ip-172-31-89-40.ec2.internal
Private DNS	172.31.89.40

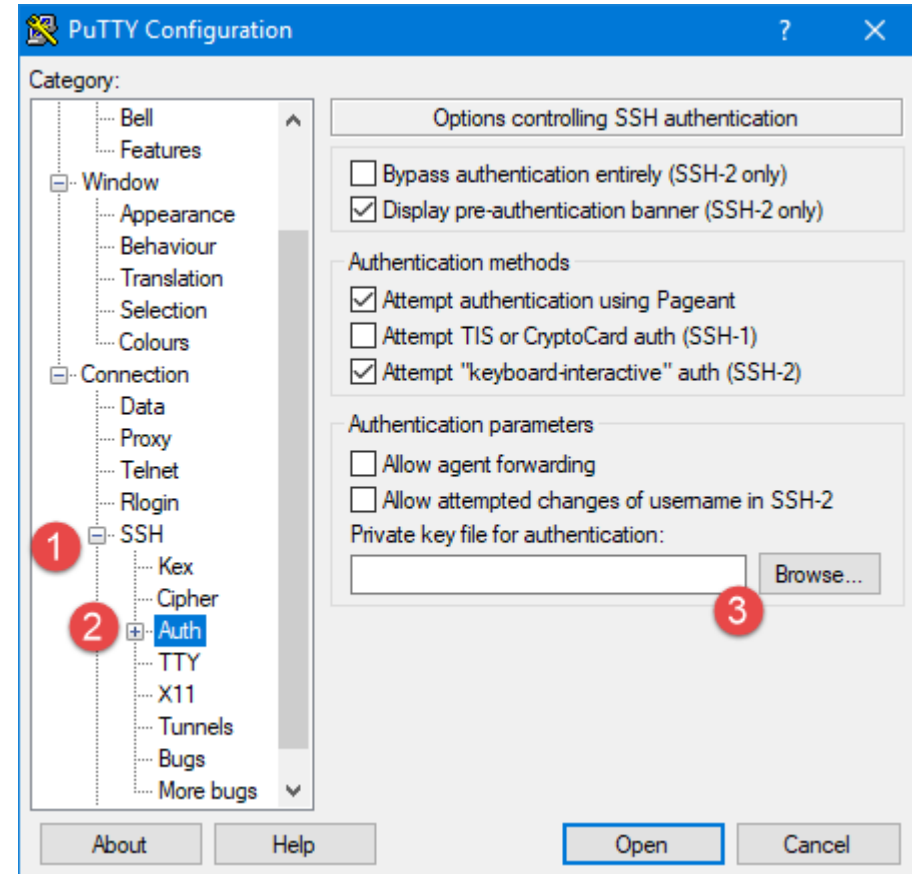
Login via putty

- Paste #3 from previous image: the EC2 instance URL

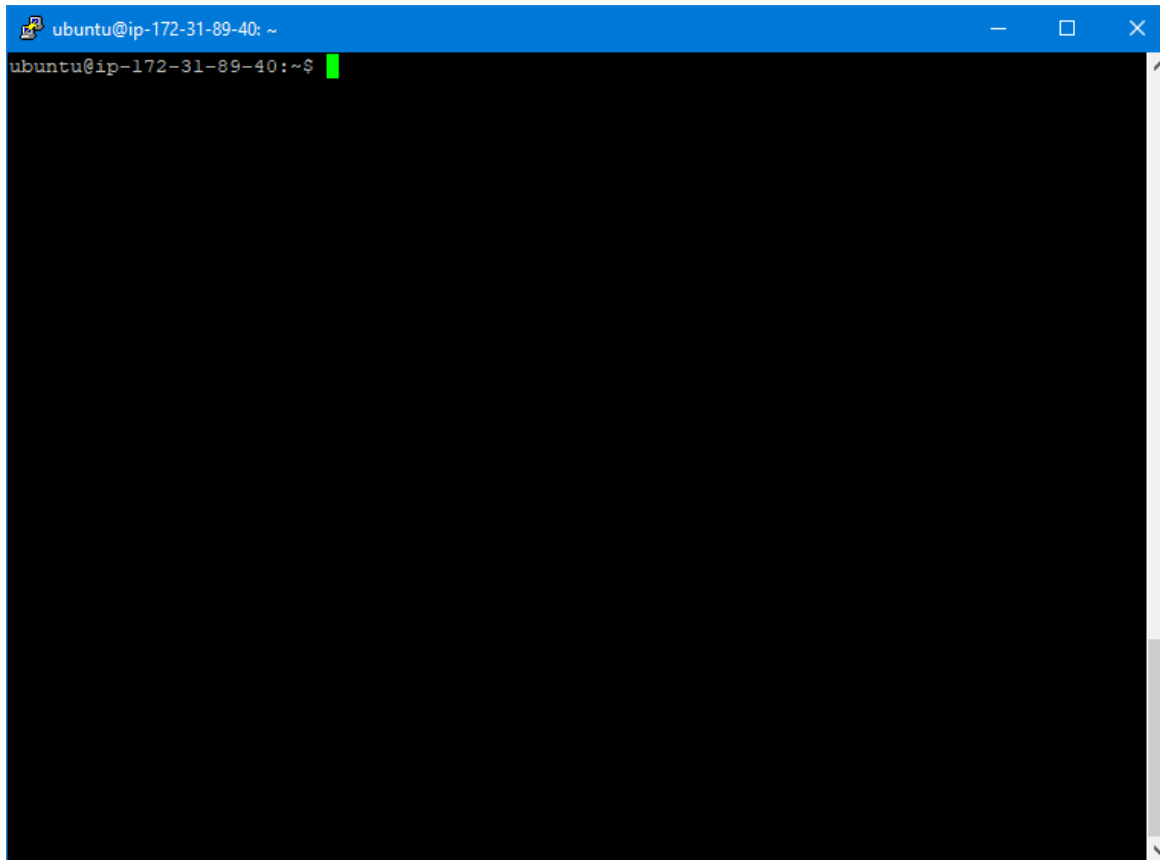


Login via putty

1. Click on SSH
2. Click on Auth
3. Choose (browse to) the .ppk file that you already created
4. (If necessary, use puttygen to convert .pem to .ppk)



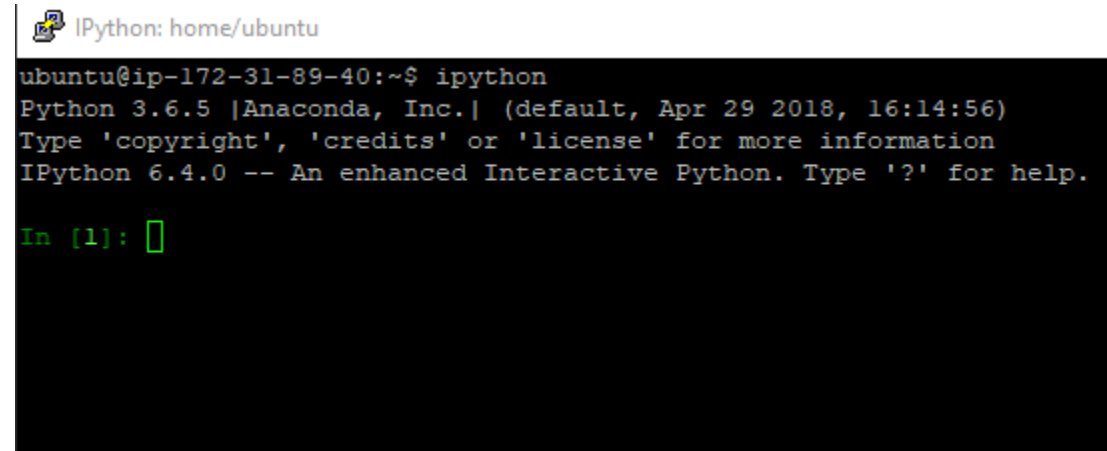
Your AWS EC2 Ubuntu screen



- Run these three commands:

```
sudo pip3 install keras
sudo pip3 install --upgrade tensorflow-gpu
source activate tensorflow_p36
```

- Then, run iPython:



Code here: <https://nbviewer.jupyter.org/github/fchollet/deep-learning-with-python-notebooks/blob/master/2.1-a-first-look-at-a-neural-network.ipynb>

```
IPython: home/ubuntu

(tensorflow_p36) ubuntu@ip-172-31-89-40:~$ ipython
Python 3.6.5 |Anaconda custom (64-bit)| (default, Apr 29 2018, 16:14:56)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import keras
Using TensorFlow backend.

In [2]: from keras.datasets import mnist

In [3]:
...: (train_images, train_labels), (test_images, test_labels) = mnist.load_data()

In [4]: train_images.shape
Out[4]: (60000, 28, 28)

In [5]: len(train_labels)
Out[5]: 60000

In [6]: train_labels
Out[6]: array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)

In [7]: test_images.shape
Out[7]: (10000, 28, 28)

In [8]: len(test_labels)
Out[8]: 10000

In [9]: test_labels
Out[9]: array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)

In [10]: █
```

```
IPython: home/ubuntu

In [10]: from keras import models
...:     from keras import layers
...:
...:     network = models.Sequential()
...:     network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
...:     network.add(layers.Dense(10, activation='softmax'))

In [11]: network.compile(optimizer='rmsprop',
...:                     loss='categorical_crossentropy',
...:                     metrics=['accuracy'])

In [12]: train_images = train_images.reshape((60000, 28 * 28))
...:     train_images = train_images.astype('float32') / 255
...:
...:     test_images = test_images.reshape((10000, 28 * 28))
...:     test_images = test_images.astype('float32') / 255
...:

In [13]: from keras.utils import to_categorical
...:
...:     train_labels = to_categorical(train_labels)
...:     test_labels = to_categorical(test_labels)
```

```

In [14]:

In [14]: network.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [=====] - 5s 80us/step - loss: 0.2586 - acc: 0.9248
Epoch 2/5
60000/60000 [=====] - 2s 26us/step - loss: 0.1051 - acc: 0.9691
Epoch 3/5
60000/60000 [=====] - 2s 26us/step - loss: 0.0687 - acc: 0.9794
Epoch 4/5
60000/60000 [=====] - 2s 25us/step - loss: 0.0501 - acc: 0.9850
Epoch 5/5
60000/60000 [=====] - 2s 25us/step - loss: 0.0376 - acc: 0.9889
Out[14]: <keras.callbacks.History at 0x7f441071b3c8>

In [15]:

In [15]: test_loss, test_acc = network.evaluate(test_images, test_labels)
10000/10000 [=====] - 0s 37us/step

In [16]:

In [16]: print('test_acc:', test_acc)
test_acc: 0.9714

```