Machine Learning: Data Foundations + Algorithms & Applications

Day 5

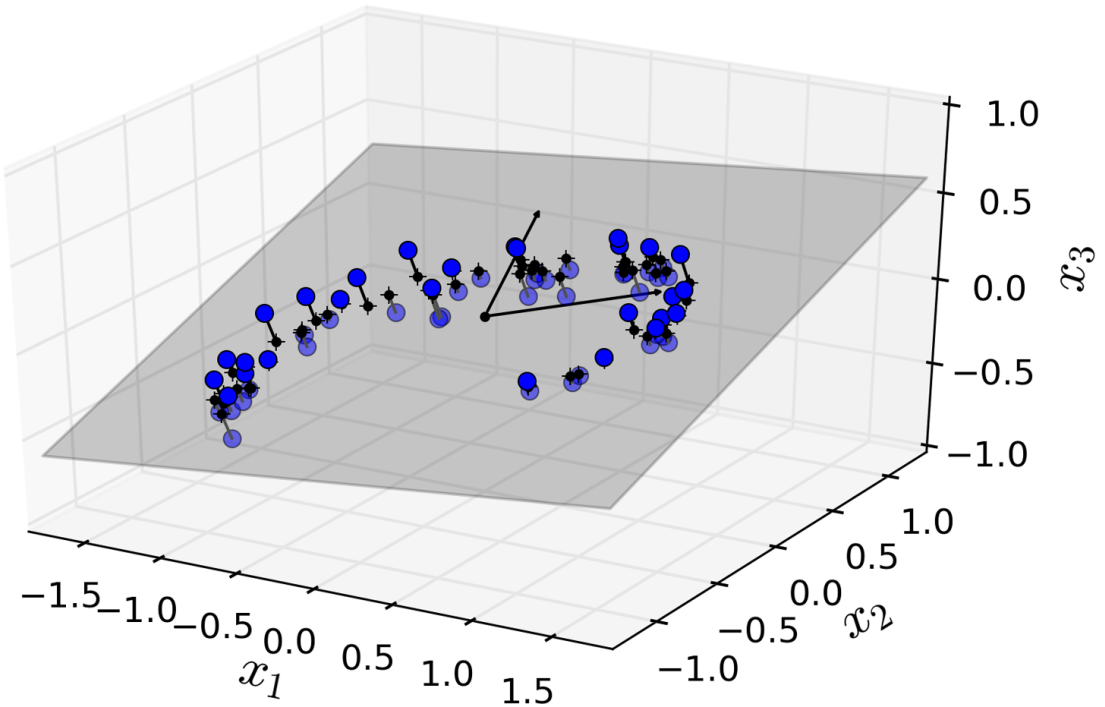
Agenda - Day 3

- Dimensionality Reduction
- Date and Time-Handling
- Model Evaluation
- Performance
- Product-based Exercise

Dimensionality Reduction

Dimensionality Reduction

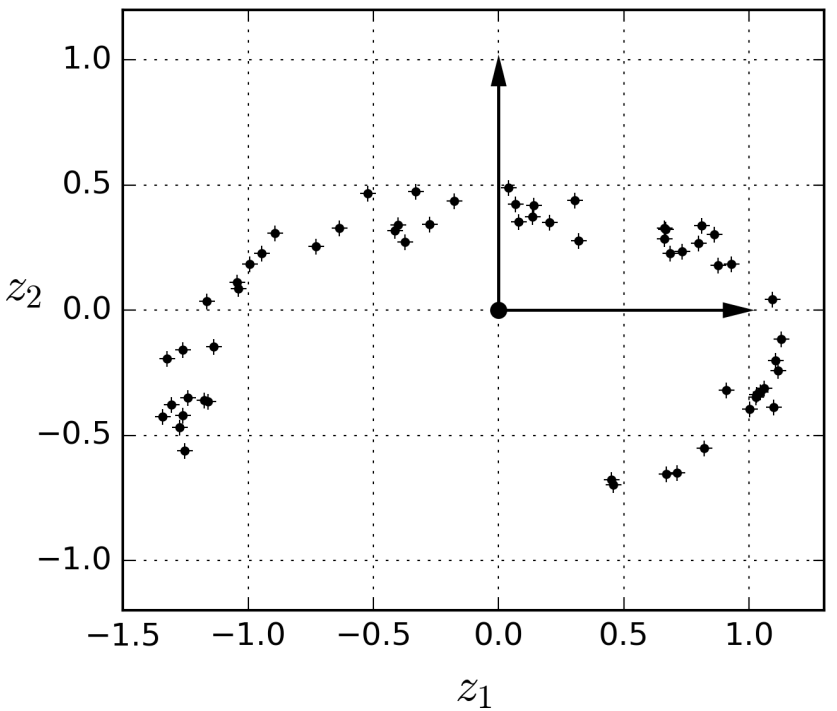
- Consider the following 3-D data: most of the variation lies in the x1 and x2 dimensions, with a small amount in x3



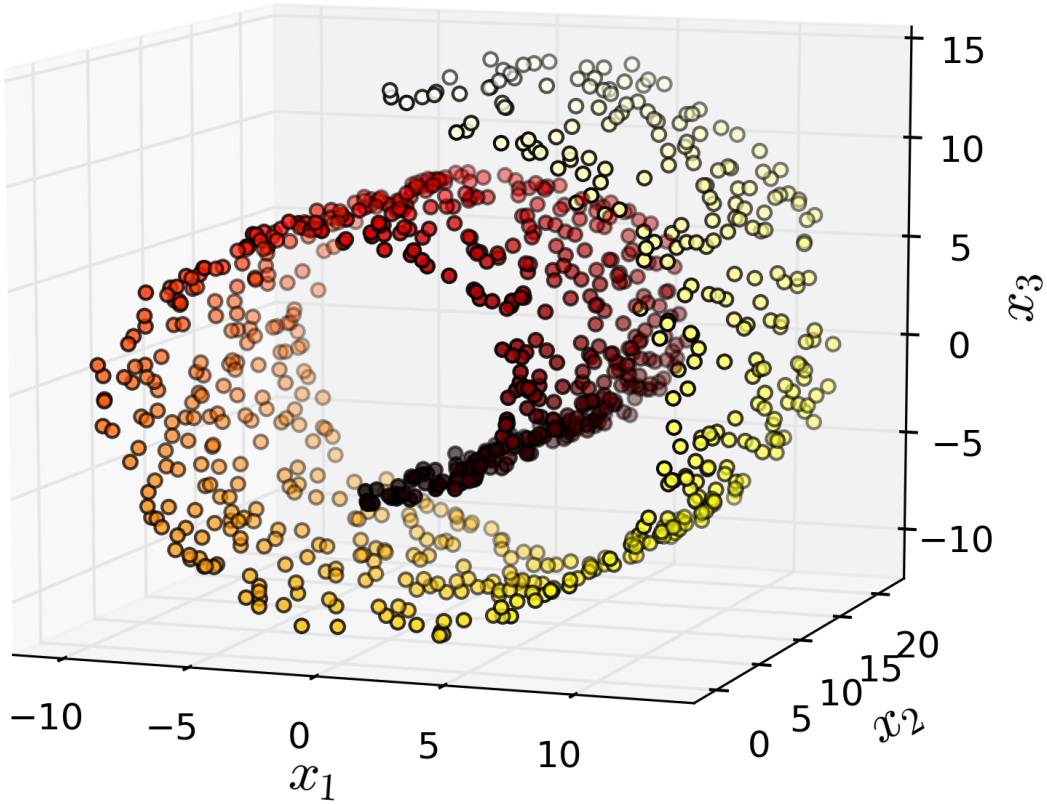
"Hands-On Machine Learning With SciKit-Learn and Tensorflow"

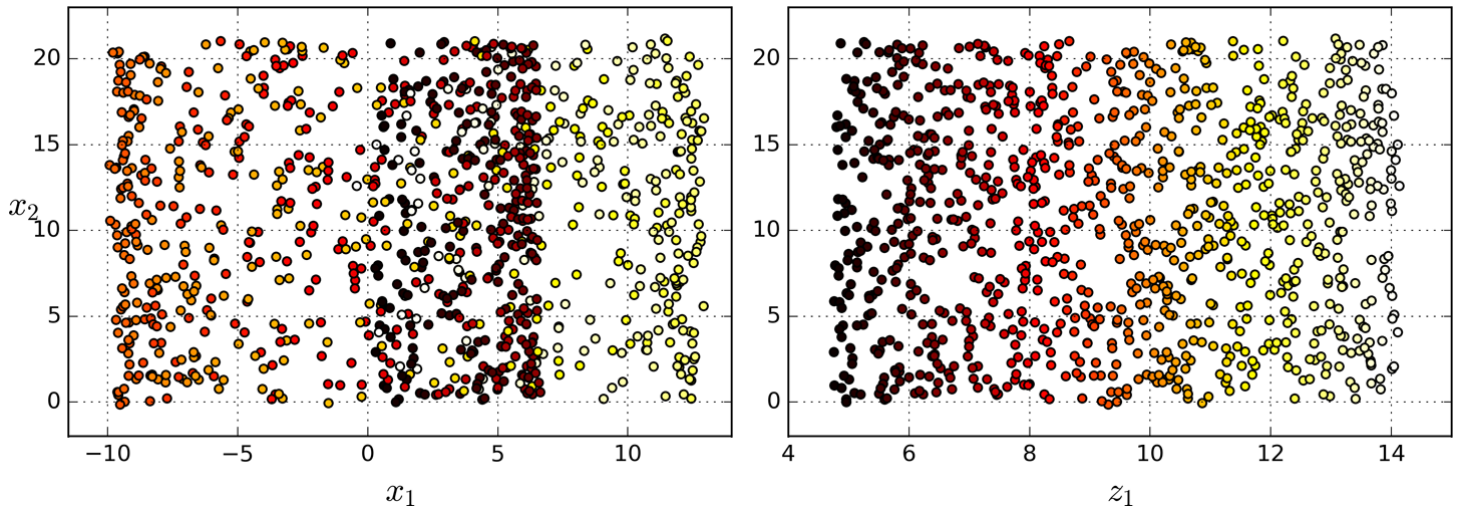
Dimensionality Reduction (cont'd)

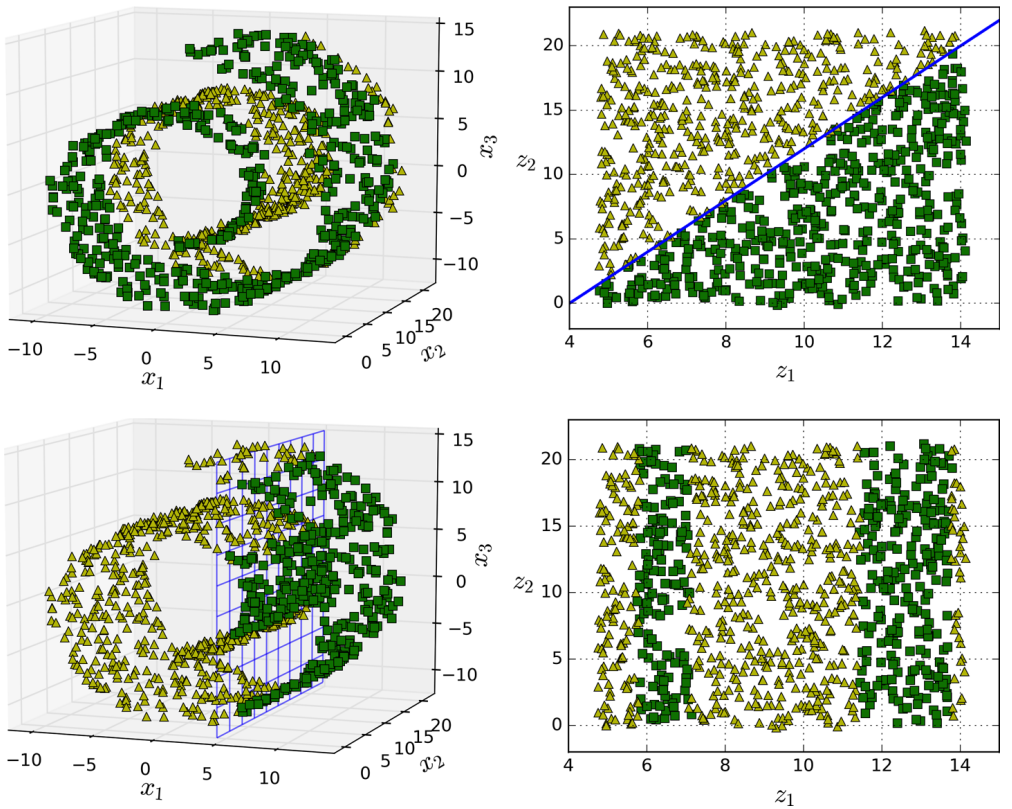
- we project the 3-dimensional data onto 2 new dimensions, the ones which preserve the majority of the variance
- notice the axes...we did not just throw away a dimension



"Hands-On Machine Learning With SciKit-Learn and Tensorflow"

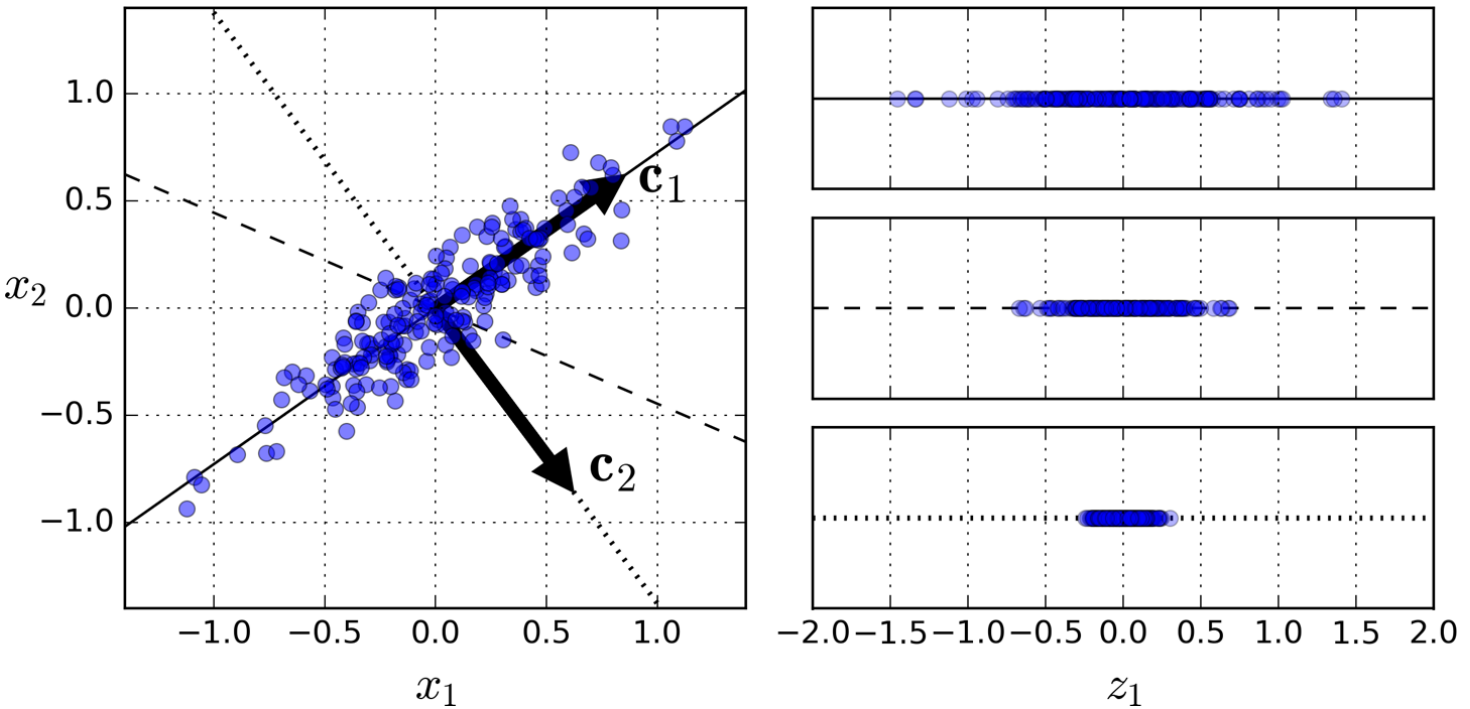

15
10
5
0
−5
−10
x_3
−10
−5
0
5
10
x_1
0
5
10
15
20
x_2


20
15
10
5
0
x_2
−10
−5
0
5
10
x_1
20
15
10
5
0
4
6
8
10
12
14
z_1


15
10
5
0
−5
−10
x_3
−10
−5
0
5
10
x_1
0
5
10
15
20
x_2
20
15
10
5
0
z_2
4
6
8
10
12
14
z_1
15
10
5
0
−5
−10
x_3
−10
−5
0
5
10
x_1
0
5
10
15
20
x_2
20
15
10
5
0
z_2
4
6
8
10
12
14
z_1

Principal Component Analysis (PCA)

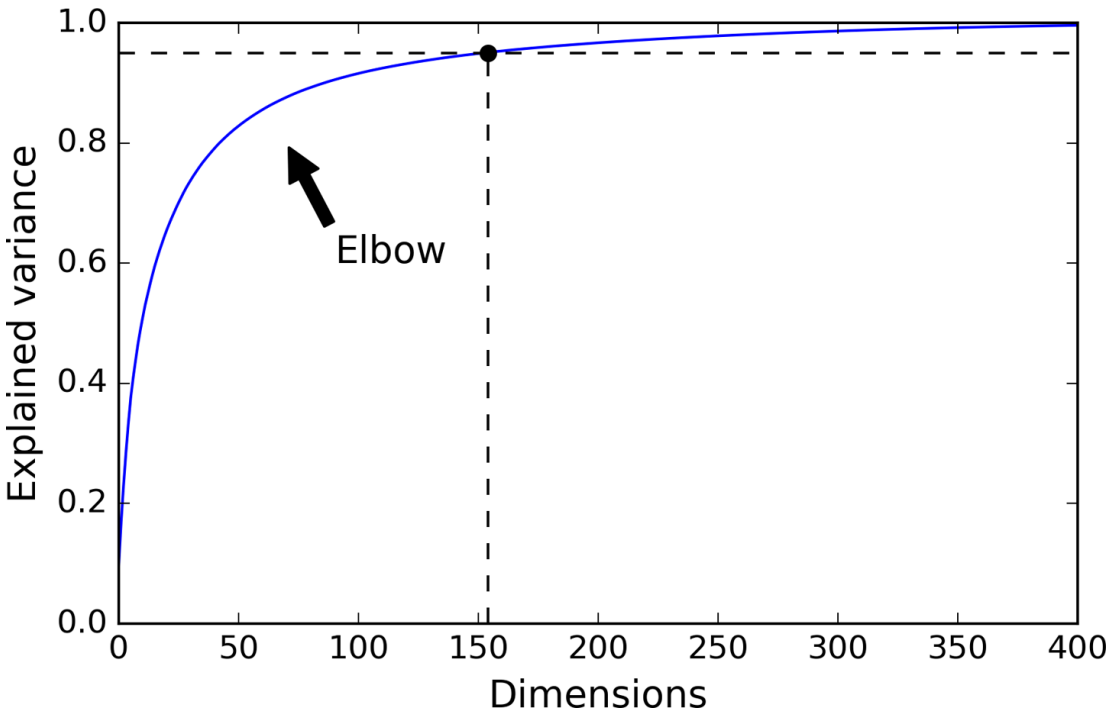
- find vectors (or principal components) that capture variation in the data
- sort them in descending order of how much variance they capture



"Hands-On Machine Learning With SciKit-Learn and Tensorflow"

Reduce Dimensionality

- How many principal components do we need to keep in order to have a reasonable approximation of original data?
- Or…how much of the variance in the data do we need to capture…95%?



"Hands-On Machine Learning With SciKit-Learn and Tensorflow"

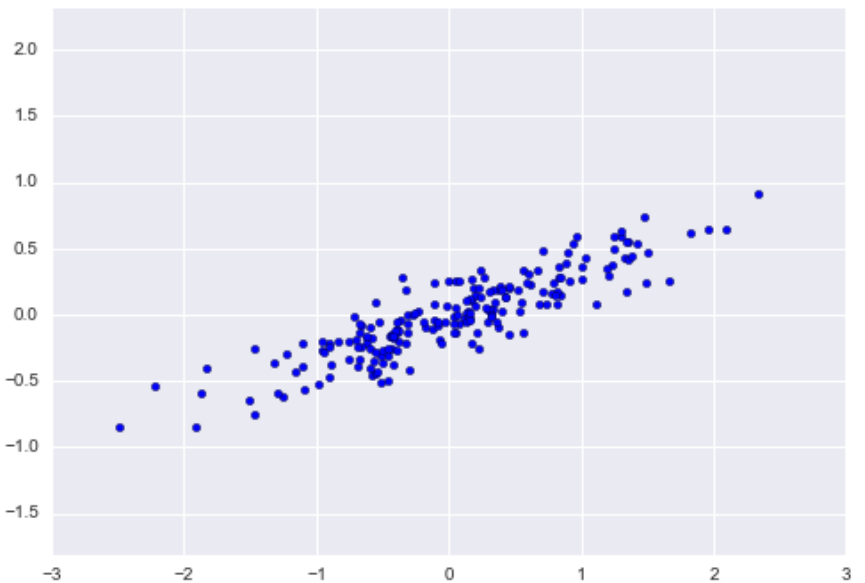
Demo: Principal Component Analysis

Principal Component Analysis

- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **`Demo - Principal Component Analysis.ipynb`** and go through it together
- when done, click here to skip screenshots

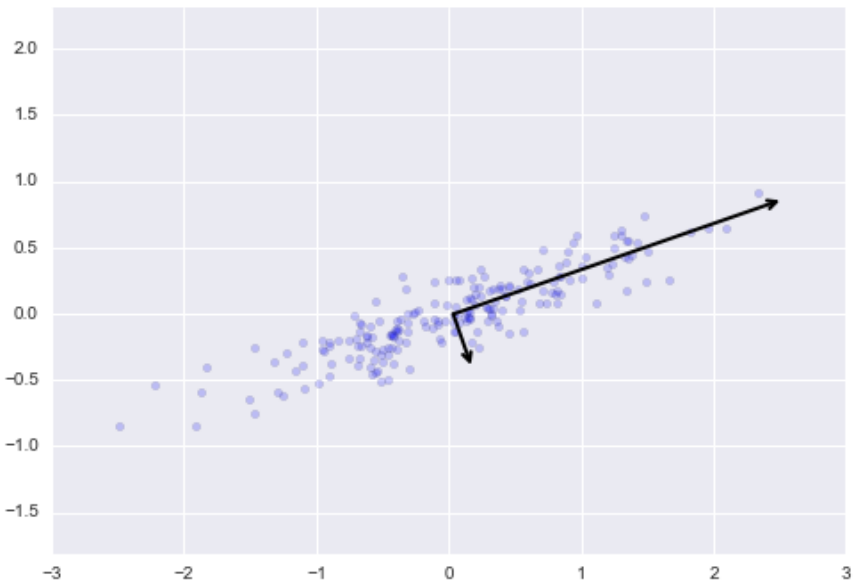
Principal Component Analysis Example

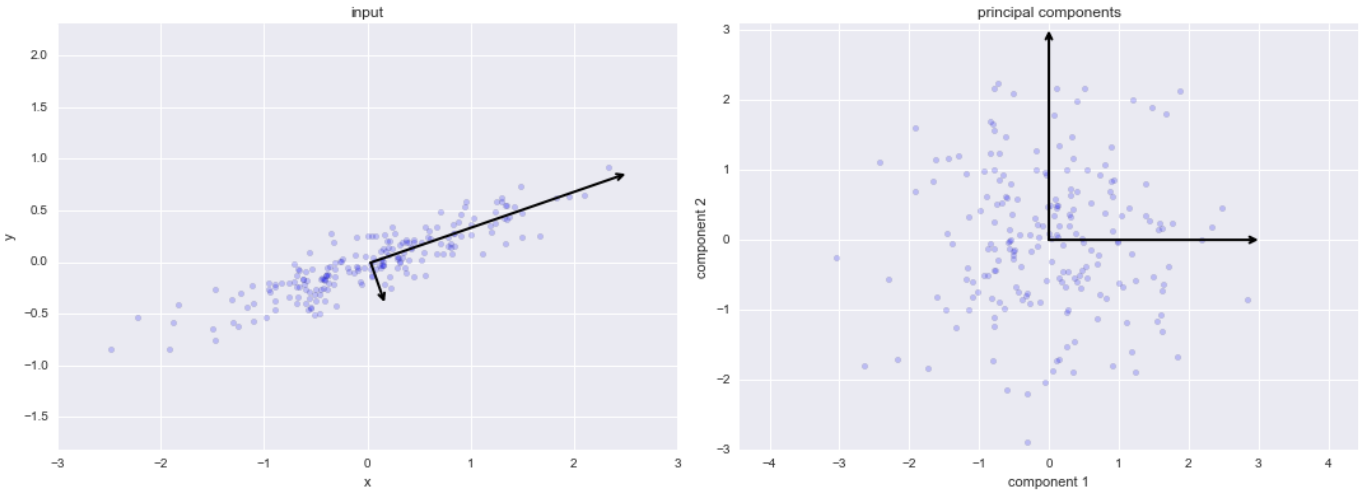
```
In [1]: import numpy as np
   ...: import matplotlib.pyplot as plt

In [2]: rng = np.random.RandomState(1)
  ...: X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
  ...: plt.scatter(X[:, 0], X[:, 1])
  ...: plt.axis('equal');
```



```
In [3]: from sklearn.decomposition import PCA
   ...: pca = PCA(n_components=2)
   ...: pca.fit(X)
   ...:
Out[3]:
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)

In [4]: pca.components_
Out[4]:
array([[-0.94446029, -0.32862557],
       [-0.32862557,  0.94446029]])

In [5]: pca.explained_variance_
Out[5]: array([ 0.75871884,  0.01838551])
```

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```




input
x
y
principal components
component 1
component 2

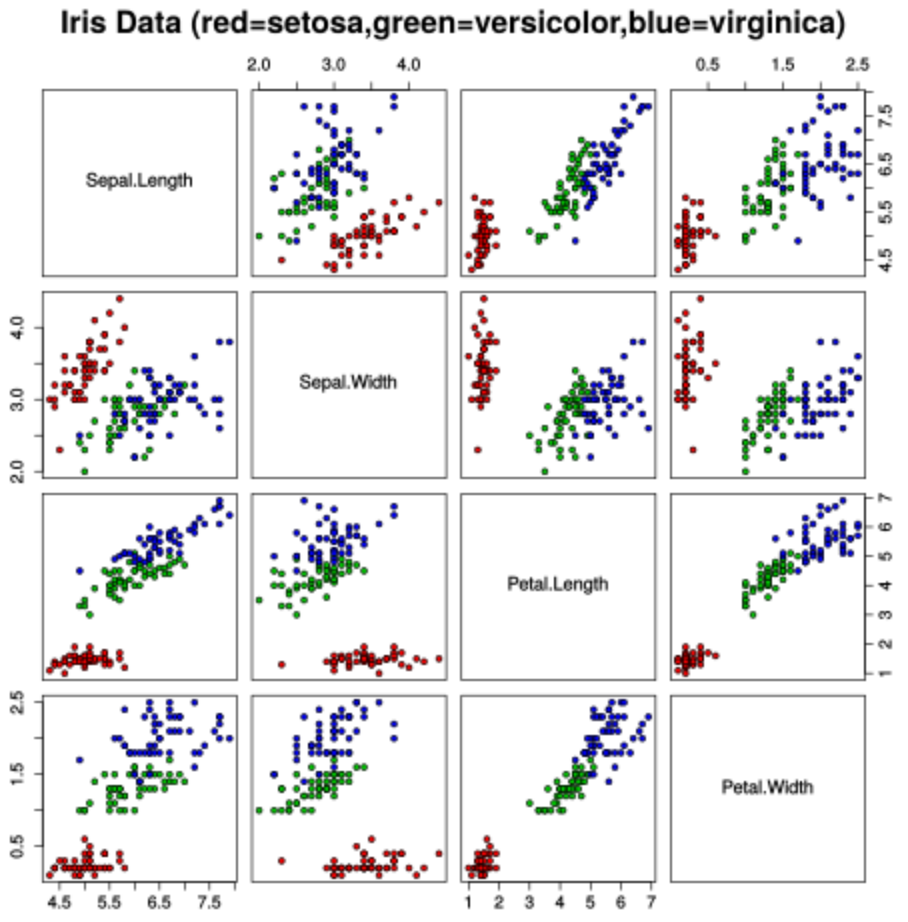
```
In [7]: pca = PCA(n_components=1)
   ...: pca.fit(X)
   ...: X_pca = pca.transform(X)
   ...:

In [8]: X.shape
Out[8]: (200, 2)

In [9]: X_pca.shape
Out[9]: (200, 1)
```

```
X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal');
```

PCA Example - Iris Data Set

```
# Code source: Gaël Varoquaux
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


from sklearn import decomposition
from sklearn import datasets

np.random.seed(5)

centers = [[1, 1], [-1, -1], [1, -1]]
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
In [18]: iris.data
Out[18]:
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4],
       [ 4.6,  3.4,  1.4,  0.3],
       [ 5. ,  3.4,  1.5,  0.2],
       [ 4.4,  2.9,  1.4,  0.2],
       [ 4.9,  3.1,  1.5,  0.1],
       [ 5.4,  3.7,  1.5,  0.2],
       [ 4.8,  3.4,  1.6,  0.2],
       ...
```



Transform the Data

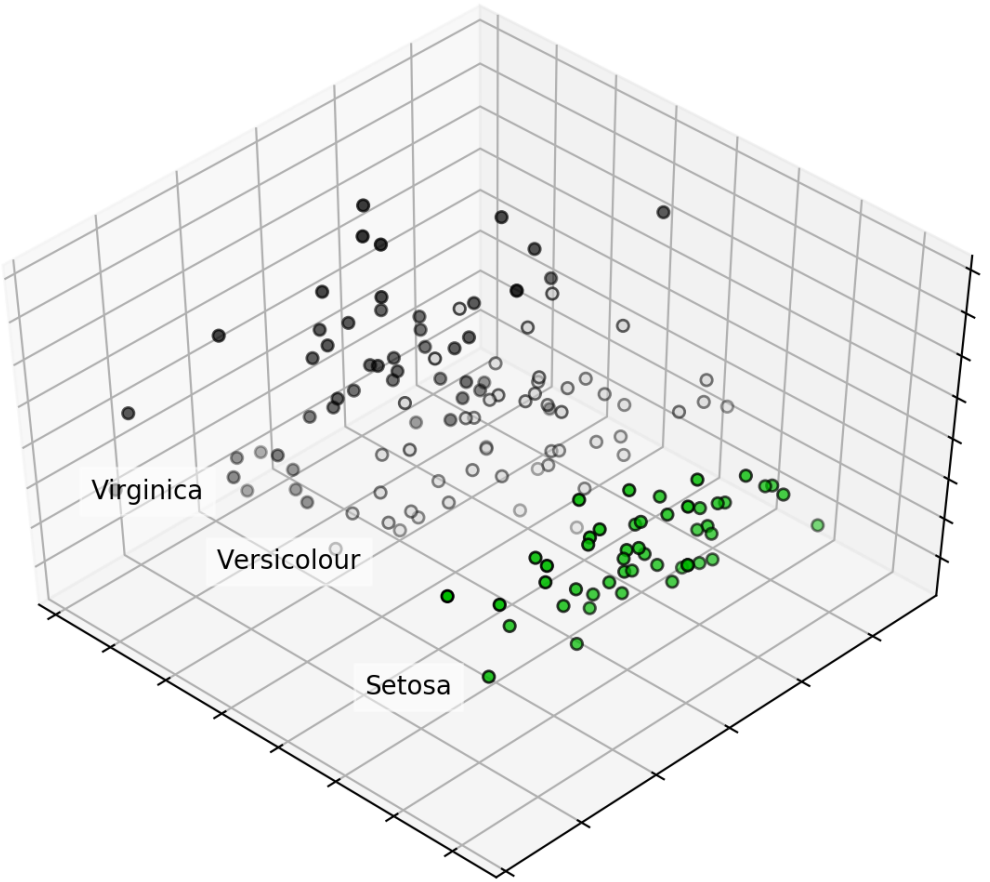
```
fig = plt.figure(1, figsize=(4, 3))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()
pca = decomposition.PCA(n_components=3)
pca.fit(X)
X = pca.transform(X)
```

```
In [11]: pca.explained_variance_ratio_
Out[11]: array([ 0.92461621,  0.05301557,  0.01718514])
```

```
In [12]: sum(pca.explained_variance_ratio_)
Out[12]: 0.99481691454981003
```

Plot the Data

```
for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:
    ax.text3D(X[y == label, 0].mean(),
              X[y == label, 1].mean() + 1.5,
              X[y == label, 2].mean(), name,
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y, cmap=plt.cm.spectral,
           edgecolor='k')

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])

plt.show()
```


Virginica
Versicolour
Setosa

Number of Dimensions to Capture Variance

```
In [13]: cumsum = np.cumsum(pca.explained_variance_ratio_)

In [14]: d = np.argmax(cumsum >= 0.95) + 1

In [15]: d
Out[15]: 2
```

PCA Example - Digits

```
In [1]: from sklearn.datasets import load_digits
   ...: digits = load_digits()
   ...: digits.data.shape
Out[1]: (1797, 64)
```

```
In [3]: import numpy as np
   ...: import matplotlib.pyplot as plt
   ...: from sklearn.decomposition import PCA
```

```
In [6]: pca = PCA(2)  # project from 64 to 2 dimensions
   ...: projected = pca.fit_transform(digits.data)
```

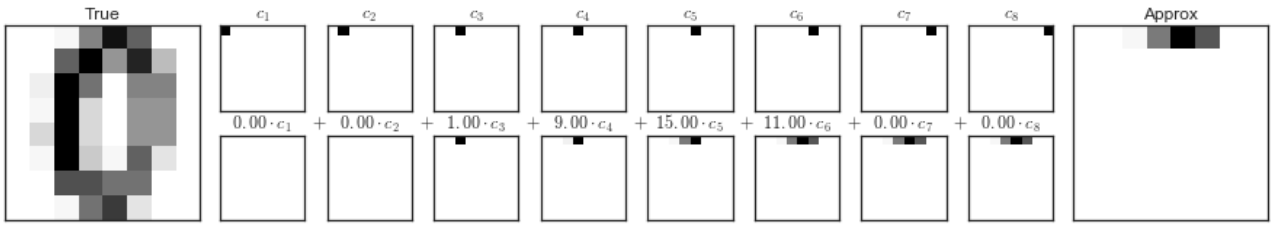
```
In [10]: digits.data.shape
Out[10]: (1797, 64)

In [11]: projected.shape
Out[11]: (1797, 2)
```

Plot the Data

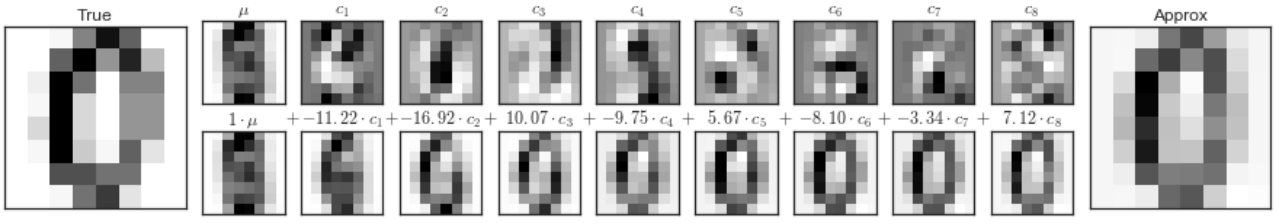
```
In [12]: plt.scatter(projected[:, 0], projected[:, 1],
    ...:             c=digits.target, edgecolor='none', alpha=0.5,
    ...:             cmap=plt.cm.get_cmap('spectral', 10))
    ...: plt.xlabel('component 1')
    ...: plt.ylabel('component 2')
    ...: plt.colorbar();
```

$$x = [x_1, x_2, x_3, \ldots, x_{64}]$$

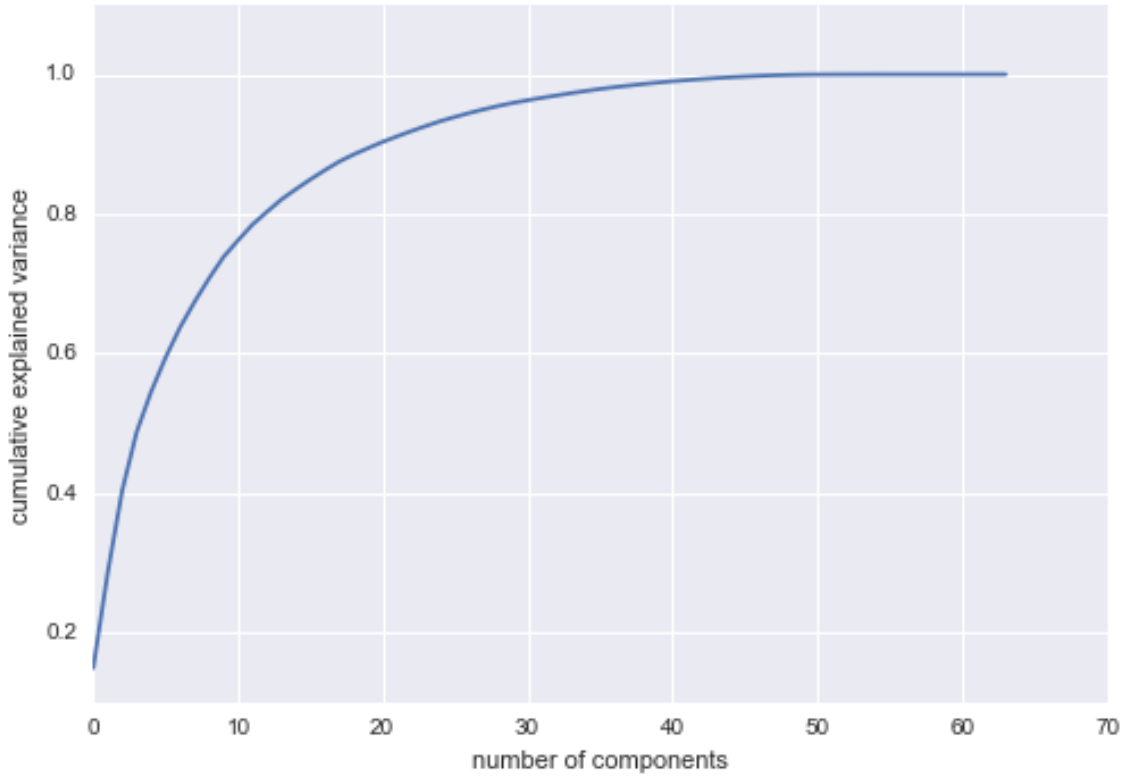
$$image(x) = x_1 \cdot p_1 + x_2 \cdot p_2 + \ldots + x_{64} \cdot p_{64}$$



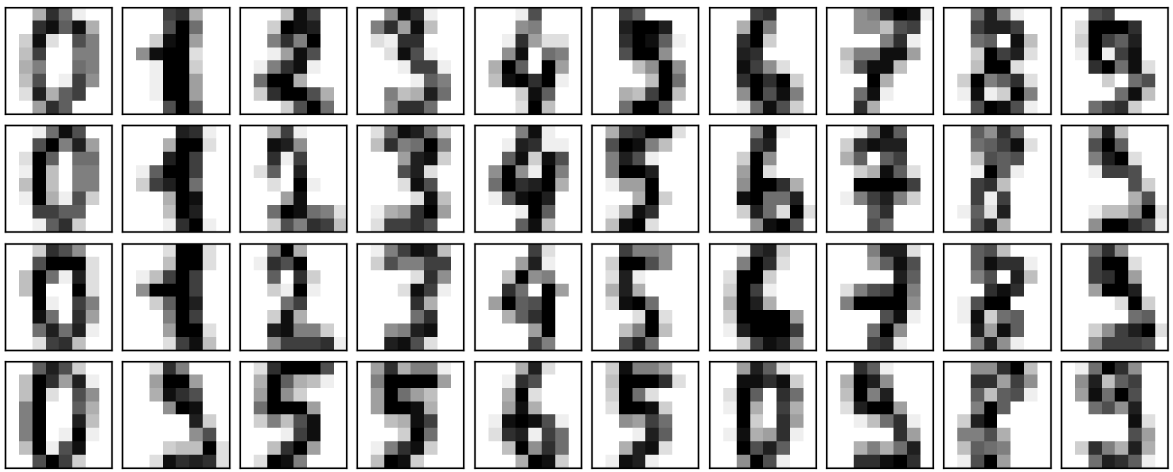
$$image(x) = mean + x_1 \cdot b_1 + x_2 \cdot b_2 + \ldots + x_{64} \cdot b_{64}$$



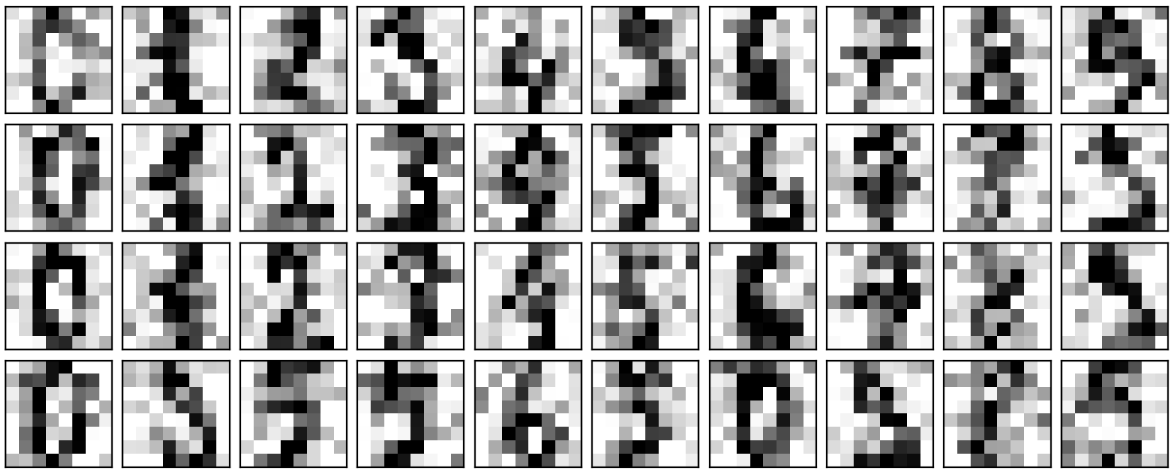
```
In [13]: pca = PCA().fit(digits.data)
    ...: plt.plot(np.cumsum(pca.explained_variance_ratio_))
    ...: plt.xlabel('number of components')
    ...: plt.ylabel('cumulative explained variance');
```



```
In [14]: def plot_digits(data):
    ...:     fig, axes = plt.subplots(4, 10, figsize=(10, 4),
    ...:                              subplot_kw={'xticks':[], 'yticks':[]},
    ...:                              gridspec_kw=dict(hspace=0.1, wspace=0.1))
    ...:     for i, ax in enumerate(axes.flat):
    ...:         ax.imshow(data[i].reshape(8, 8),
    ...:                   cmap='binary', interpolation='nearest',
    ...:                   clim=(0, 16))
    ...: plot_digits(digits.data)
```

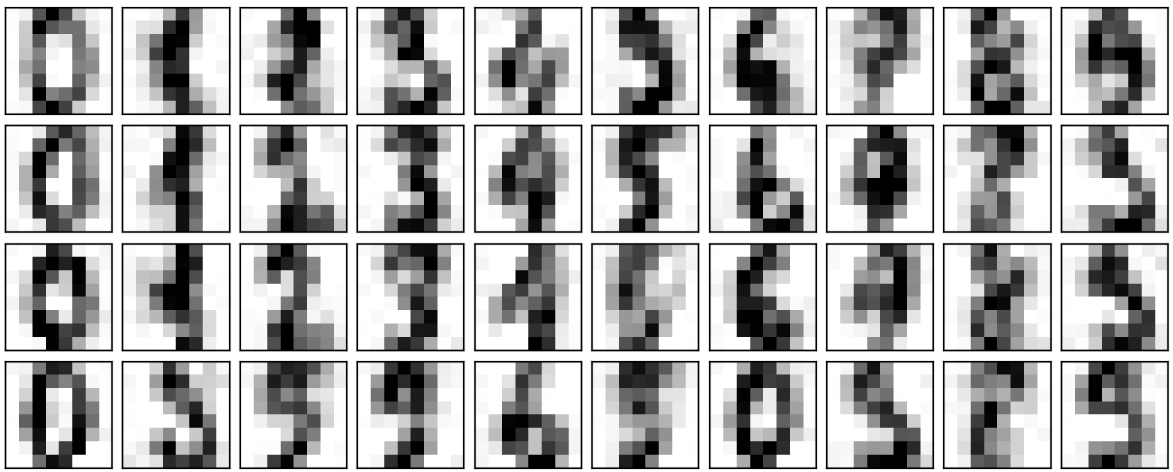


```
In [15]: np.random.seed(42)
    ...: noisy = np.random.normal(digits.data, 4)
    ...: plot_digits(noisy)
```



```
In [16]: pca = PCA(0.50).fit(noisy)
    ...: pca.n_components_
    ...:
Out[16]: 12
```

```
In [17]: components = pca.transform(noisy)
    ...: filtered = pca.inverse_transform(components)
    ...: plot_digits(filtered)
```



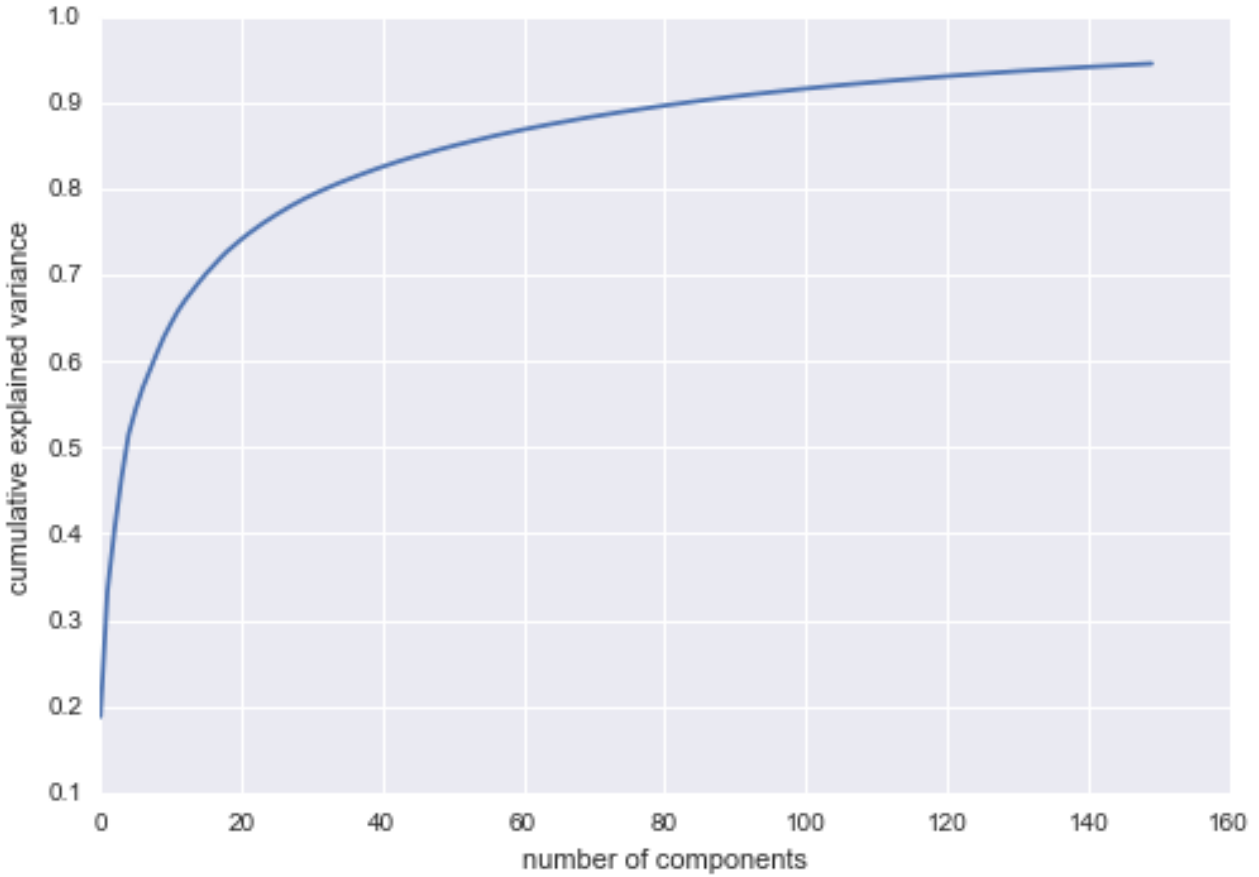
PCA Example - Faces

```
In [1]: from sklearn.datasets import fetch_lfw_people
In [2]: faces = fetch_lfw_people(min_faces_per_person=60)
In [3]: faces.target_names
Out[3]:
array(['Ariel Sharon', 'Colin Powell', 'Donald Rumsfeld', 'George W Bush',
       'Gerhard Schroeder', 'Hugo Chavez', 'Junichiro Koizumi',
       'Tony Blair'],
      dtype='|S17')
In [5]: faces.images.shape
Out[5]: (1348, 62, 47)
```

```
In [6]: from sklearn.decomposition import RandomizedPCA
   ...: pca = RandomizedPCA(150)
   ...: pca.fit(faces.data)
Out[6]:
   RandomizedPCA(copy=True, iterated_power=2, n_components=150,
     random_state=None, whiten=False)
```

```
In [8]: import matplotlib.pyplot as plt
In [9]: fig, axes = plt.subplots(3, 8, figsize=(9, 4),
   ...:                          subplot_kw={'xticks':[], 'yticks':[]},
   ...:                          gridspec_kw=dict(hspace=0.1, wspace=0.1))
   ...: for i, ax in enumerate(axes.flat):
   ...:     ax.imshow(pca.components_[i].reshape(62, 47), cmap='bone')
```

```
In [11]: import numpy as np
In [12]: plt.plot(np.cumsum(pca.explained_variance_ratio_))
    ...: plt.xlabel('number of components')
    ...: plt.ylabel('cumulative explained variance');
```



```
In [13]: pca = RandomizedPCA(150).fit(faces.data)
    ...: components = pca.transform(faces.data)
    ...: projected = pca.inverse_transform(components)
In [14]: fig, ax = plt.subplots(2, 10, figsize=(10, 2.5),
    ...:                 subplot_kw={'xticks':[], 'yticks':[]},
    ...:                 gridspec_kw=dict(hspace=0.1, wspace=0.1))
    ...: for i in range(10):
    ...:     ax[0, i].imshow(faces.data[i].reshape(62, 47), cmap='binary_r')
    ...:     ax[1, i].imshow(projected[i].reshape(62, 47), cmap='binary_r')
    ...:
    ...: ax[0, 0].set_ylabel('full-dim\ninput')
    ...: ax[1, 0].set_ylabel('150-dim\nreconstruction');
```

Exercise: Principal Component Analysis

(open the notebook named **`Exercise 14 - Principal Component Analysis.ipynb`**)

Demo: Date and Time Handling

Date and Time Handling

- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **`Demo - Date and Time Handling.ipynb`** and go through it together
- when done, click here to skip screenshots

Type	Description
date	Store calendar date (year, month, day) using the Gregorian calendar
time	Store time of day as hours, minutes, seconds and microseconds
datetime	Store date and time
timedelta	Store the difference between two datetimes
tzinfo	Store time zone information

Datetime and timedelta

```
>>> from datetime import datetime
>>> now = datetime.now()
>>> now
datetime.datetime(2018, 2, 18, 20, 33, 4, 108848)
```

```
>>> now.year, now.month, now.day
(2018, 2, 18)
```

```
>>> delta = datetime(2018, 6, 14) - datetime(2003, 6, 14)
>>> delta
datetime.timedelta(5479)
```

```
>>> delta.days, delta.seconds
(5479, 0)
```

```
>>> from datetime import timedelta
>>> datetime.now() + timedelta(12)
datetime.datetime(2018, 3, 20, 20, 39, 7, 552783)
```

Converting to/from strings

```
>>> str(datetime(2018, 2, 18))
'2018-02-18 00:00:00'
```

```
>>> datetime(2018, 2, 18).strftime('%Y-%m-%d')
'2018-02-18'
```

```
>>> datetime.strptime('2018-02-18', '%Y-%m-%d')
datetime.datetime(2018, 2, 18, 0, 0)
```

Type	Description
%Y	Four-digit year
%y	Two-digit year
%m	Two-digit month
%d	Two-digit day
%H	Hour (24-hour)
%I	Hour (12-hour)
%M	Two-digit minute
%S	Seconds
%w	Weekday
%z	UTC TZ offset
%F	%Y-%m-%d
%D	%m/%d/%y

Parser

```
>>> from dateutil.parser import parse
>>> parse('2018-02-18')
datetime.datetime(2018, 2, 18, 0, 0)
```

```
>>> parse('2/12/2018')
datetime.datetime(2018, 2, 12, 0, 0)
```

```
>>> parse('2/12/2018', dayfirst=True)
datetime.datetime(2018, 12, 2, 0, 0)
```

Pandas Date Handling

```
>>> import pandas as pd
>>> dates = ['2018-02-18 12:00:00', '2018-02-05 14:30:00', '2018-03-09 17:35:00']
>>> pd.to_datetime(dates)
DatetimeIndex(['2018-02-18 12:00:00', '2018-02-05 14:30:00',
               '2018-03-09 17:35:00'],
              dtype='datetime64[ns]', freq=None)
```

```
>>> date_idx = pd.to_datetime(dates + [None])
>>> date_idx
DatetimeIndex(['2018-02-18 12:00:00', '2018-02-05 14:30:00',
               '2018-03-09 17:35:00', 'NaT'],
              dtype='datetime64[ns]', freq=None)
```

```
>>> pd.isnull(date_idx)
array([False, False, False,  True], dtype=bool)
```

Date ranges

```
>>> index=pd.date_range('2/18/2018', periods=1000)
>>> index
DatetimeIndex(['2018-02-18', '2018-02-19', '2018-02-20', '2018-02-21',
               '2018-02-22', '2018-02-23', '2018-02-24', '2018-02-25',
               '2018-02-26', '2018-02-27',
               ...
               '2020-11-04', '2020-11-05', '2020-11-06', '2020-11-07',
               '2020-11-08', '2020-11-09', '2020-11-10', '2020-11-11',
               '2020-11-12', '2020-11-13'],
              dtype='datetime64[ns]', length=1000, freq='D')
```

```
>>> len(index)
1000
```

```
>>> index[:20:2]
DatetimeIndex(['2018-02-18', '2018-02-20', '2018-02-22', '2018-02-24',
               '2018-02-26', '2018-02-28', '2018-03-02', '2018-03-04',
               '2018-03-06', '2018-03-08'],
              dtype='datetime64[ns]', freq='2D')
```

```
>>> import numpy as np
>>> dates = pd.Series(np.random.random(1000), index=index)
```

```
>>> dates['2018/03/15':'2018/03/20']
2018-03-15    0.599067
2018-03-16    0.808155
2018-03-17    0.988099
2018-03-18    0.982365
2018-03-19    0.755831
2018-03-20    0.035025
Freq: D, dtype: float64
```

Offsets

```
>>> from pandas.tseries.offsets import Day, MonthEnd
>>> now
datetime.datetime(2018, 2, 18, 0, 0)
```

```
>>> now + 6 * Day()
Timestamp('2018-02-24 00:00:00')
```

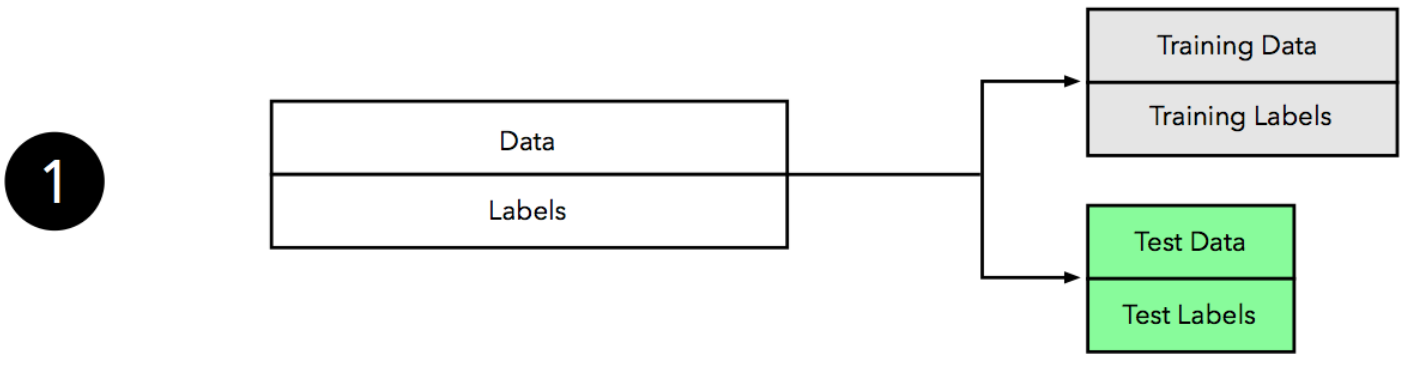
```
>>> now + MonthEnd()
Timestamp('2018-02-28 00:00:00')
```

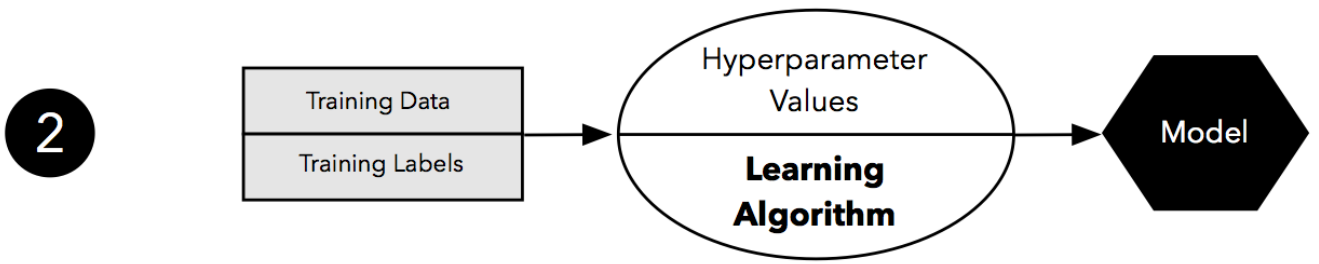
```
>>> now + MonthEnd(3)
Timestamp('2018-04-30 00:00:00')
```

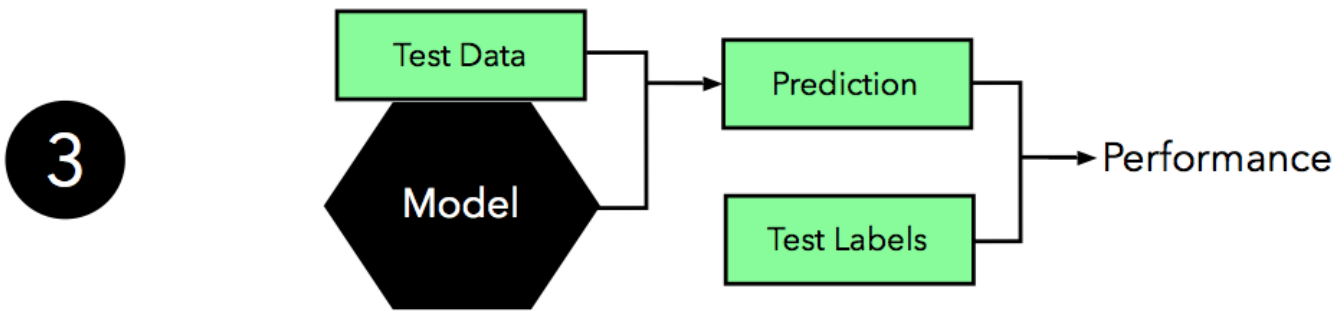
Exercise: Feature Engineering Dates

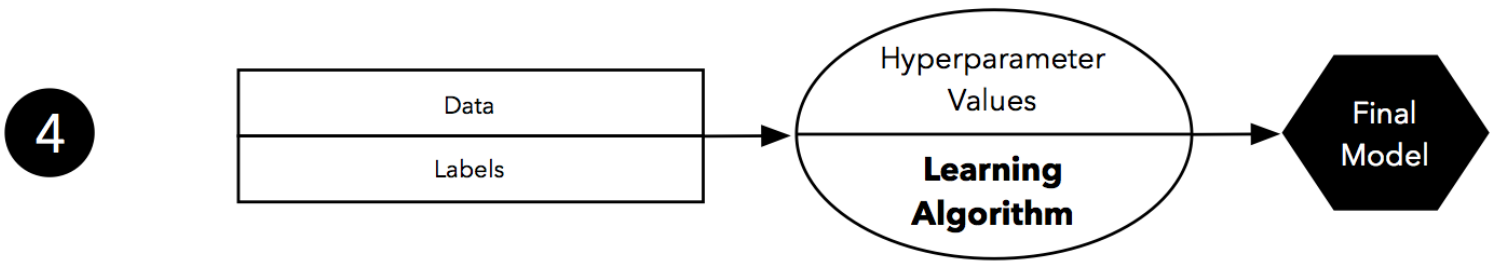
(open the notebook named **`Exercise 15 - Feature Engineering Dates.ipynb`**)

Model Evaluation

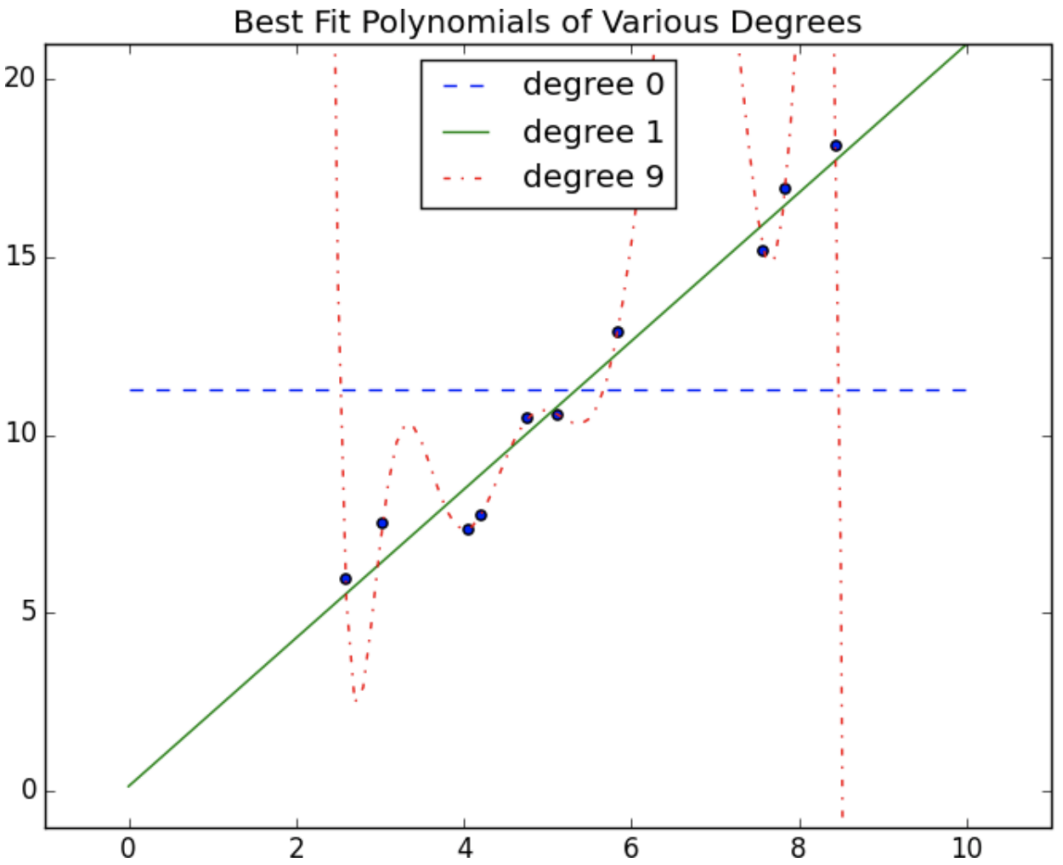






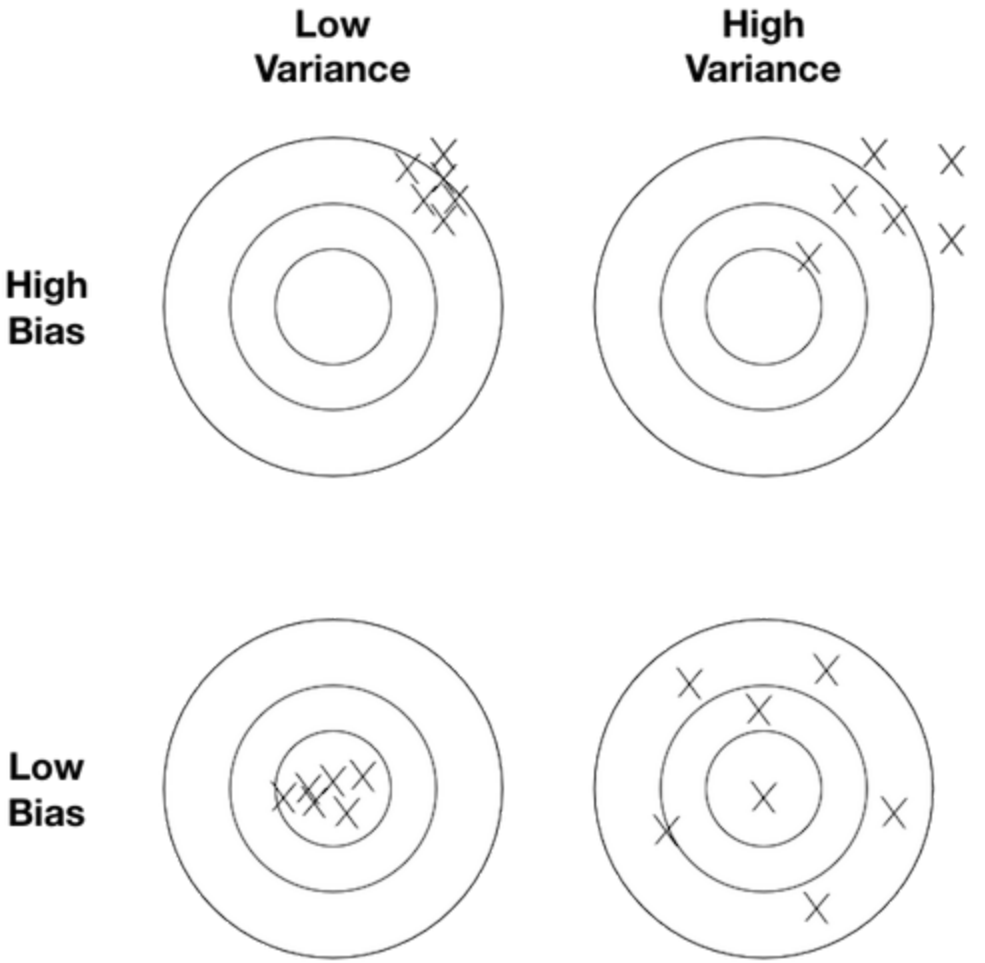


Model fit



"Data Science from Scratch" by Joel Grus

Variance Bias Tradeoff



Demo: Model Evaluation

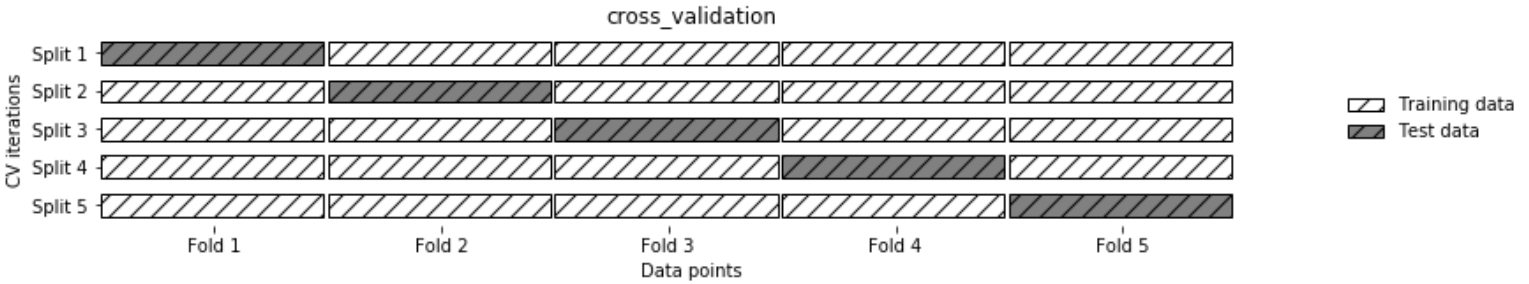
Model Evaluation

- there are screenshots in this presentation to maintain continuity, but let's open the notebook named **`Demo - Model Evaluation.ipynb`** and go through it together
- when done, click here to skip screenshots

Logistic Regression

```
# "Introduction to Machine Learning with Python", Müller and Guido

>>> from sklearn.datasets import make_blobs
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.model_selection import train_test_split

>>> # create a synthetic dataset
>>> X, y = make_blobs(random_state=0)
>>> # split data and labels into a training and a test set
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
>>> # instantiate a model and fit it to the training set
>>> logreg = LogisticRegression().fit(X_train, y_train)
>>> # evaluate the model on the test set
>>> print("Test set score: {:.2f}".format(logreg.score(X_test, y_test)))

Test set score: 0.88
```

Cross Validation



"Introduction to Machine Learning with Python", Müller and Guido

```
>>> from sklearn.model_selection import cross_val_score
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression

>>> iris = load_iris()
>>> logreg = LogisticRegression()

>>> scores = cross_val_score(logreg, iris.data, iris.target)
>>> print("Cross-validation scores: {}".format(scores))

Cross-validation scores: [0.961 0.922 0.958]
```

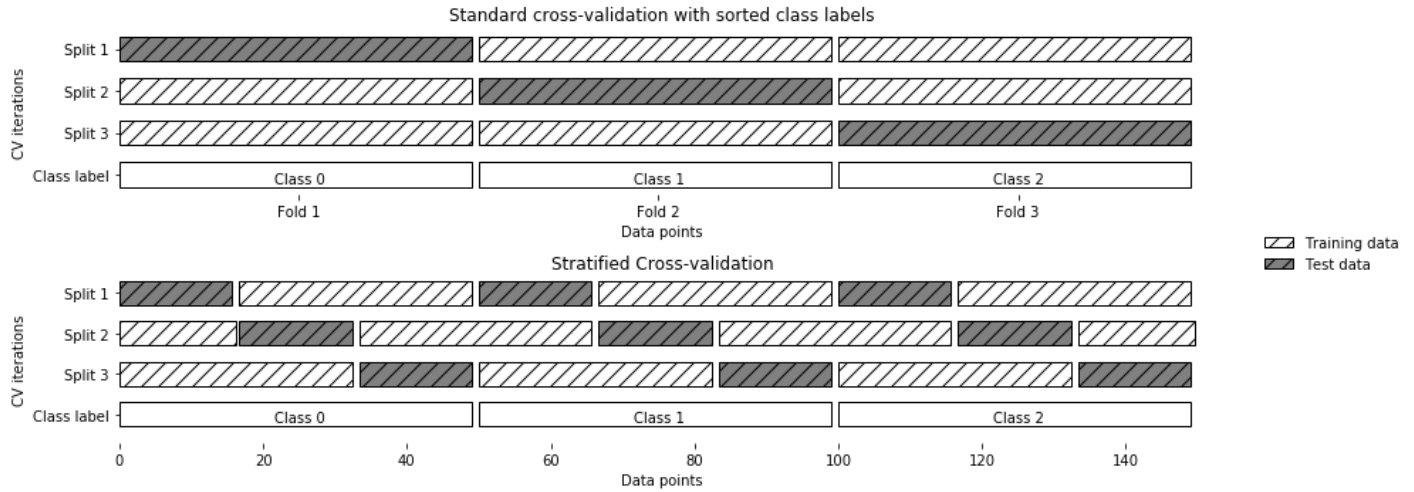
```
>>> scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
>>> print("Cross-validation scores: {}".format(scores))

Cross-validation scores: [ 1.     0.967  0.933  0.9    1.   ]
```

```
>>> print("Average cross-validation score: {:.2f}".format(scores.mean()))

Average cross-validation score: 0.96
```

Uh-oh

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> print("Iris labels:\n{}".format(iris.target))

Iris labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Stratified Cross Validation



"Introduction to Machine Learning with Python", Müller and Guido

Specifying KFold Validation

```
>>> from sklearn.model_selection import KFold
>>> kfold = KFold(n_splits=5)
```

```
>>> print("Cross-validation scores:\n{}".format(
>>>   cross_val_score(logreg, iris.data, iris.target, cv=kfold)))

Cross-validation scores:
[1.    0.933 0.433 0.967 0.433]
```

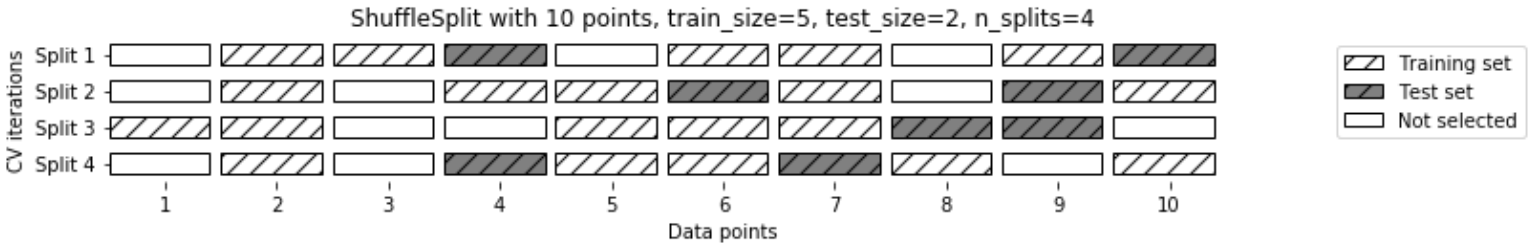
```
>>> kfold = KFold(n_splits=3)
>>> print("Cross-validation scores:\n{}".format(
>>>   cross_val_score(logreg, iris.data, iris.target, cv=kfold)))

Cross-validation scores:
[0. 0. 0.]
```

```
>>> kfold = KFold(n_splits=3, shuffle=True, random_state=0)
>>> print("Cross-validation scores:\n{}".format(
>>>    cross_val_score(logreg, iris.data, iris.target, cv=kfold)))

Cross-validation scores:
[0.9  0.96 0.96]
```

Leave-out One Validation

```
>>> from sklearn.model_selection import LeaveOneOut
>>> loo = LeaveOneOut()
>>> scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
>>> print("Number of cv iterations: ", len(scores))
>>> print("Mean accuracy: {:.2f}".format(scores.mean()))

Number of cv iterations:  150
Mean accuracy: 0.95
```

Shuffle-split Cross Validation

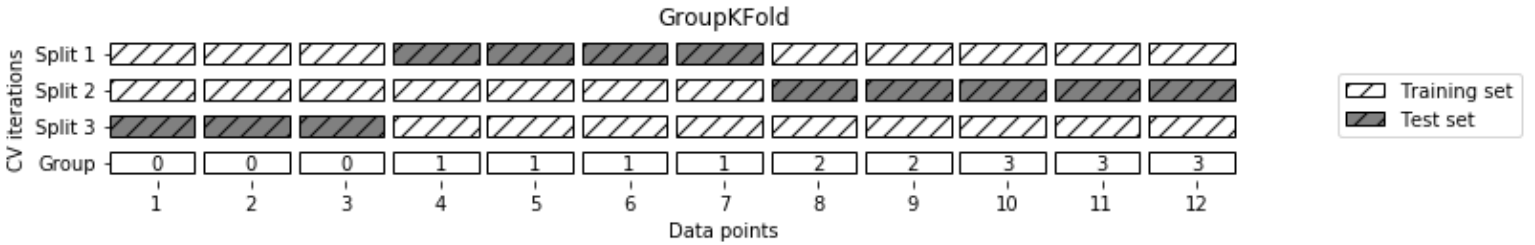


"Introduction to Machine Learning with Python", Müller and Guido

Shuffle-split Validation

```
>>> from sklearn.model_selection import ShuffleSplit
>>> shuffle_split = ShuffleSplit(test_size=.5, train_size=.5, n_splits=10)
>>> scores = cross_val_score(logreg, iris.data, iris.target, cv=shuffle_split)
>>> print("Cross-validation scores:\n{}".format(scores))

Cross-validation scores:
[ 0.92   0.987  0.92   0.96   0.987  0.947  0.92   0.853  0.88   0.96 ]
```

Group Cross Validation

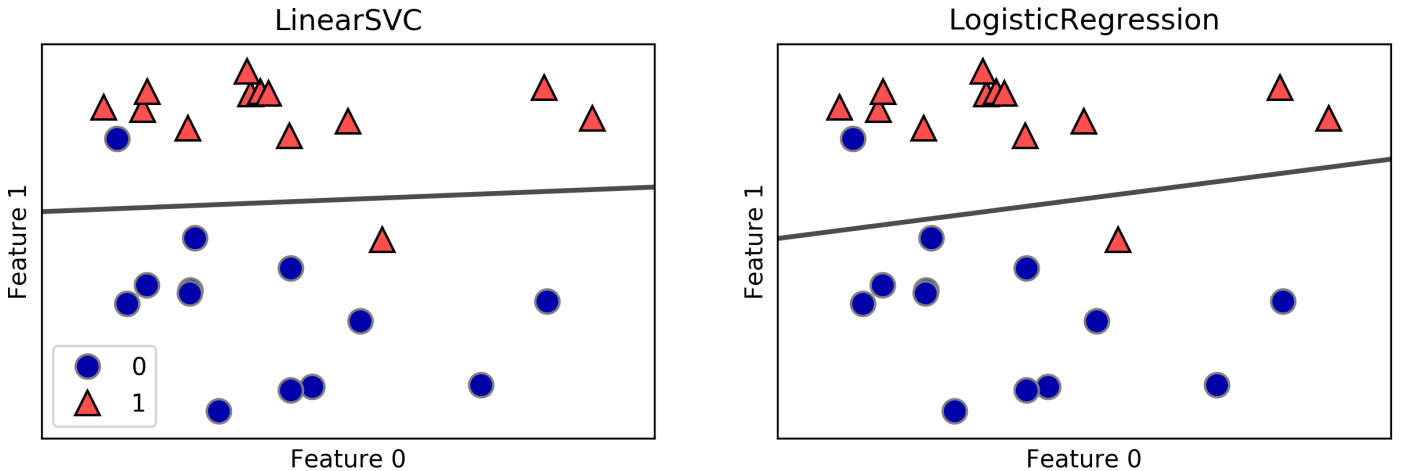


"Introduction to Machine Learning with Python", Müller and Guido

GroupKFold Validation

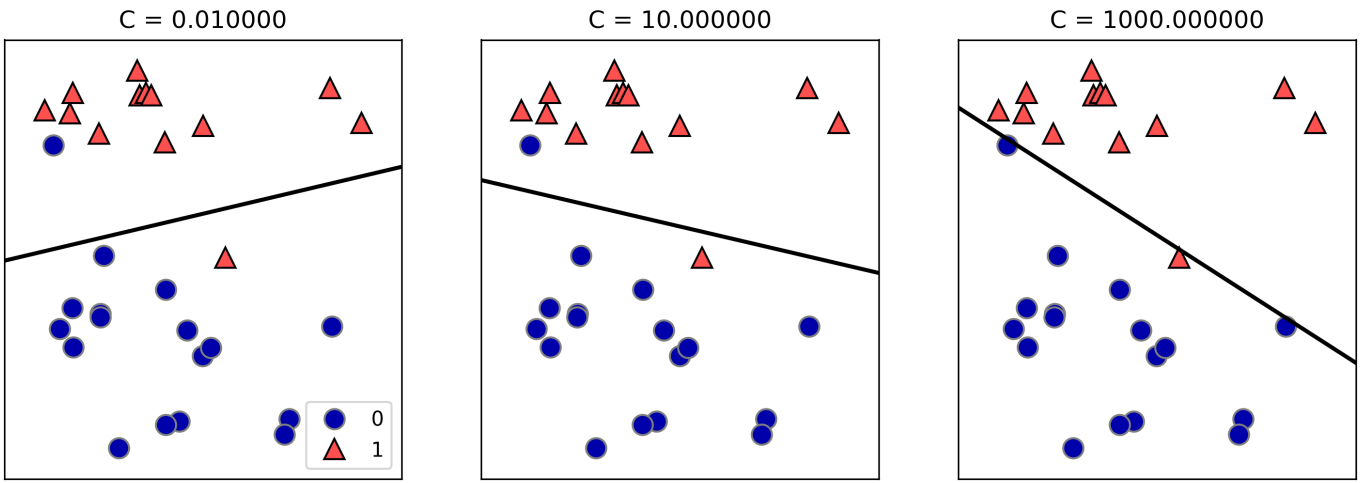
```
>>> from sklearn.model_selection import GroupKFold
>>> # create synthetic dataset
>>> X, y = make_blobs(n_samples=12, random_state=0)
>>> # assume the first three samples belong to the same group,
>>> # then the next four, etc
>>> groups = [0, 0, 0, 1, 1, 1, 1, 2, 2, 3, 3, 3]
>>> scores = cross_val_score(logreg, X, y, groups, cv=GroupKFold(n_splits=3))
>>> print("Cross-validation scores:\n{}".format(scores))

Cross-validation scores:
[ 0.75   0.8    0.667]
```

Model Performance - Linear Classifiers



"Introduction to Machine Learning with Python", Müller and Guido

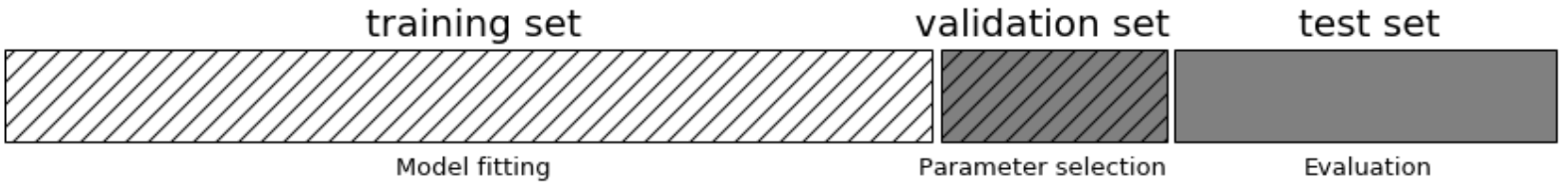
Regularization Parameters



"Introduction to Machine Learning with Python", Müller and Guido

```
>>> # naive grid search implementation
>>> from sklearn.svm import SVC
>>> X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
>>>                                                     random_state=0)
>>> print("Size of training set: {}   size of test set: {}".format(
>>>       X_train.shape[0], X_test.shape[0]))
>>>
>>> best_score = 0
>>>
>>> for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
>>>     for C in [0.001, 0.01, 0.1, 1, 10, 100]:
>>>         # for each combination of parameters, train an SVC
>>>         svm = SVC(gamma=gamma, C=C)
>>>         svm.fit(X_train, y_train)
>>>         # evaluate the SVC on the test set
>>>         score = svm.score(X_test, y_test)
>>>         # if we got a better score, store the score and parameters
>>>         if score > best_score:
>>>             best_score = score
>>>             best_parameters = {'C': C, 'gamma': gamma}
>>>
>>> print("Best score: {:.2f}".format(best_score))
>>> print("Best parameters: {}".format(best_parameters))

Size of training set: 112   size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```

Train-Validate-Test



"Introduction to Machine Learning with Python", Müller and Guido

```
from sklearn.svm import SVC
# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
```

```
# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(
    X_trainval, y_trainval, random_state=1)
```

```
best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the validation set
        score = svm.score(X_valid, y_valid)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
```

```
# rebuild a model on the combined training and validation set,
# and evaluate it on the test set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

```
Size of training set: 84   size of validation set: 28   size of test set: 38

Best score on validation set: 0.96
Best parameters:  {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92
```

Cross Validation and Grid Search

```
# reference: manual_grid_search_cv
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters,
        # train an SVC
        svm = SVC(gamma=gamma, C=C)
        # perform cross-validation
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # compute mean cross-validation accuracy
        score = np.mean(scores)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
# rebuild a model on the combined training and validation set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
```

GridSearchCV

```
>>> from sklearn.model_selection import GridSearchCV
>>> from sklearn.svm import SVC
>>>
>>> param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
>>>               'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
>>>
>>> grid_search = GridSearchCV(SVC(), param_grid, cv=5,
>>>                            return_train_score=True)
```

```
>>> X_train, X_test, y_train, y_test = train_test_split(
>>>     iris.data, iris.target, random_state=0)
>>> grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise',
    estimator=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),
    fit_params=None, iid=True, n_jobs=1,
    param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100],
                'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
    pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
    scoring=None, verbose=0)
```

```
>>> print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
```

```
Test set score: 0.97
```

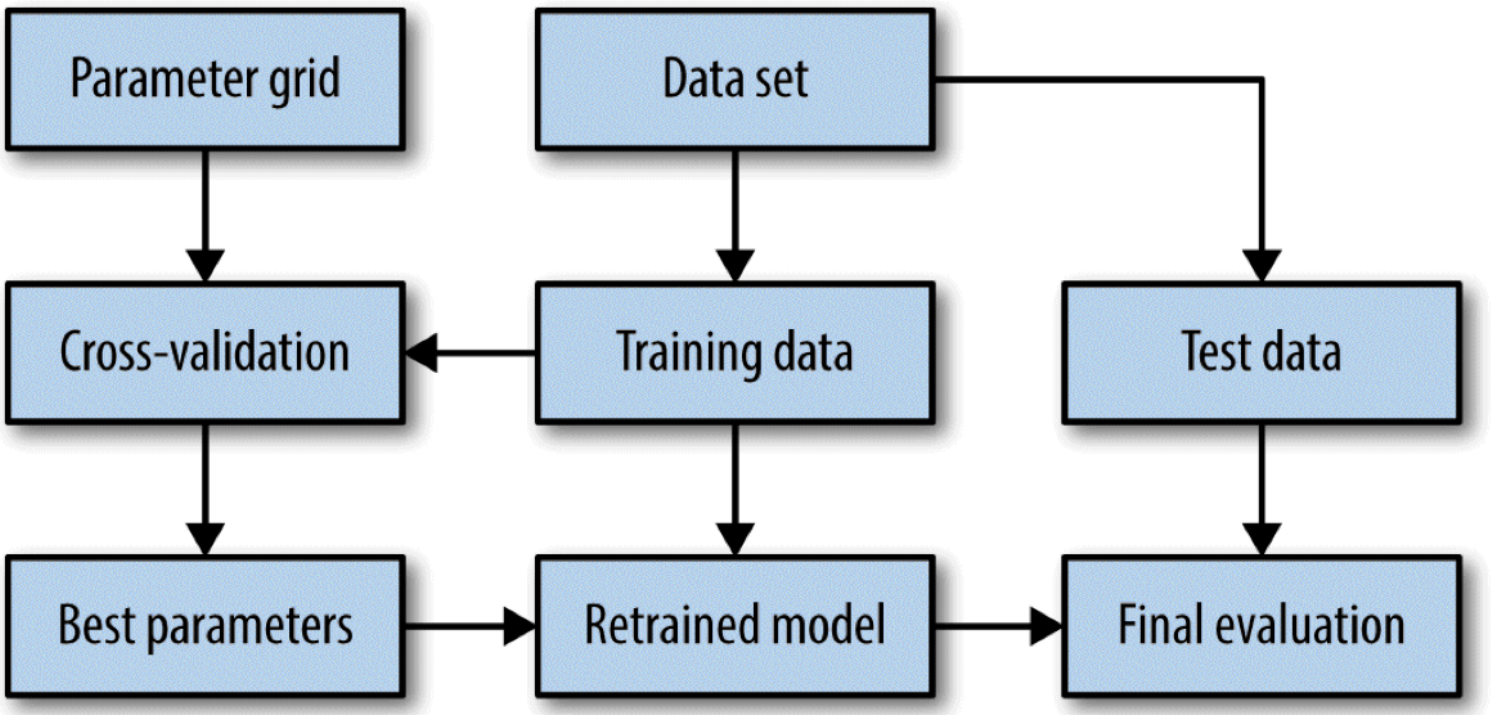
```
>>> print("Best parameters: {}".format(grid_search.best_params_))
>>> print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'C': 100, 'gamma': 0.01}
Best cross-validation score: 0.97
```

```
>>> print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

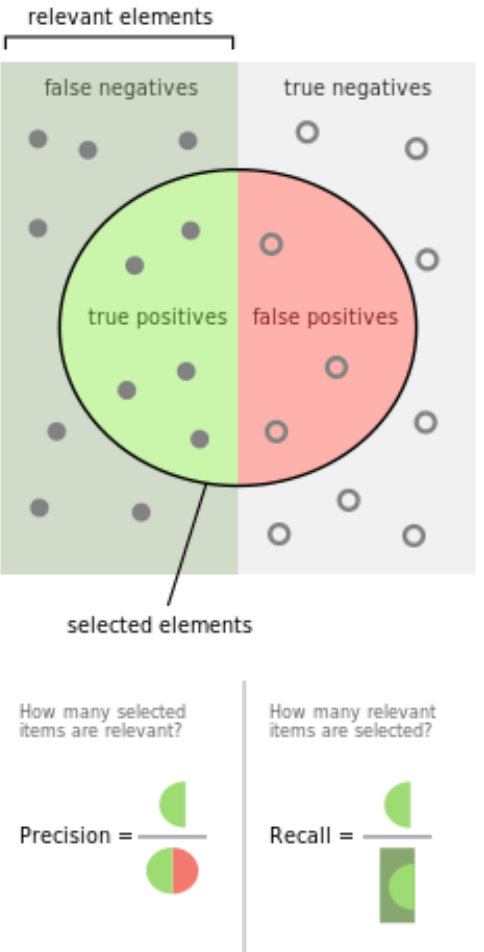
```
Best estimator:
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Model Evaluation



"Introduction to Machine Learning with Python", Müller and Guido

Precision, Recall and Accuracy

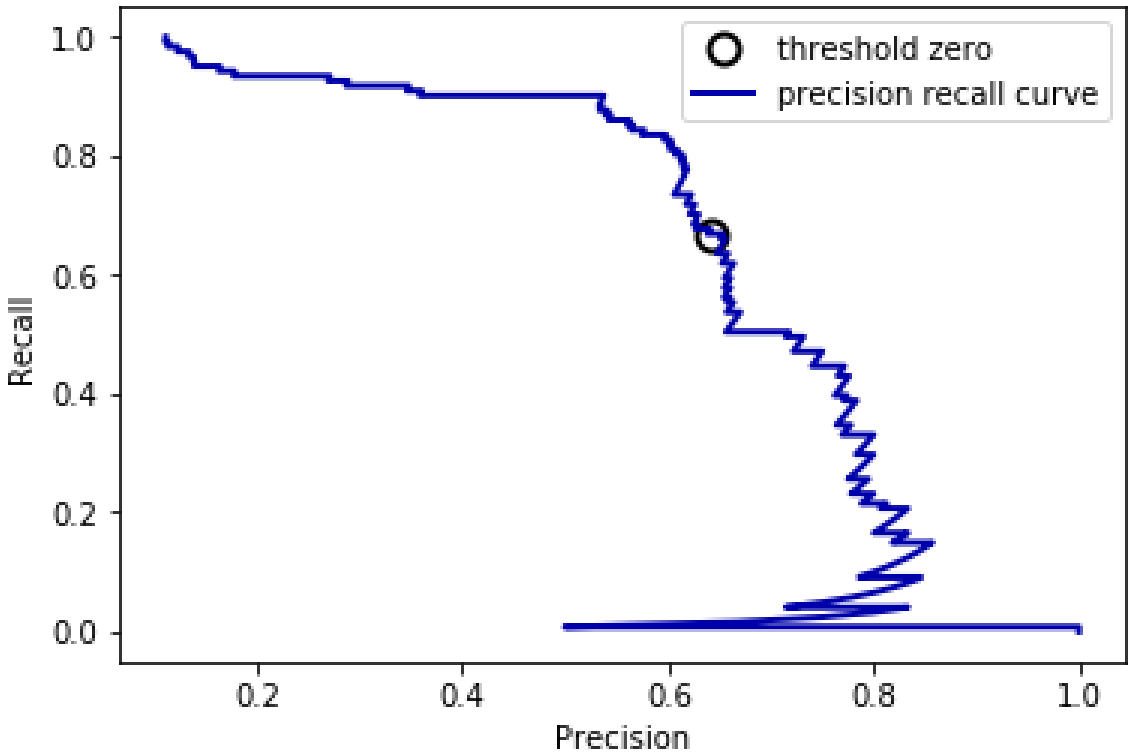


$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

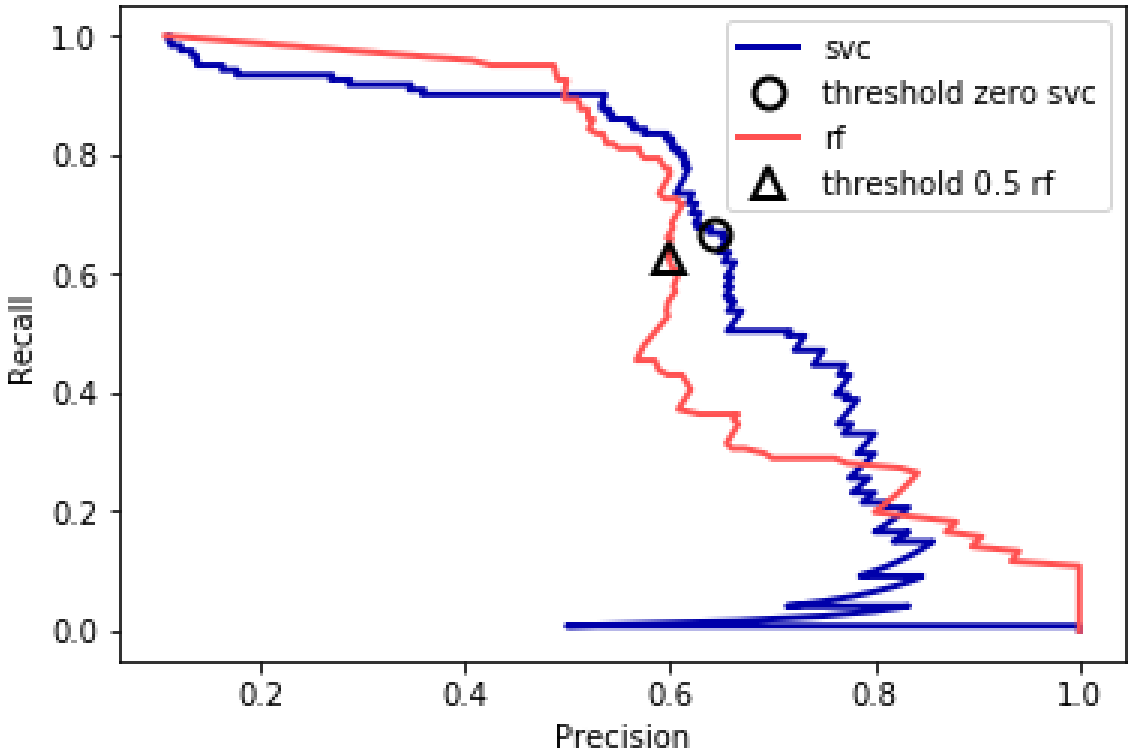
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Comparing Precision and Recall (SVM)



"Introduction to Machine Learning with Python", Müller and Guido

Comparing Precision and Recall (SVM vs RF)



"Introduction to Machine Learning with Python", Müller and Guido

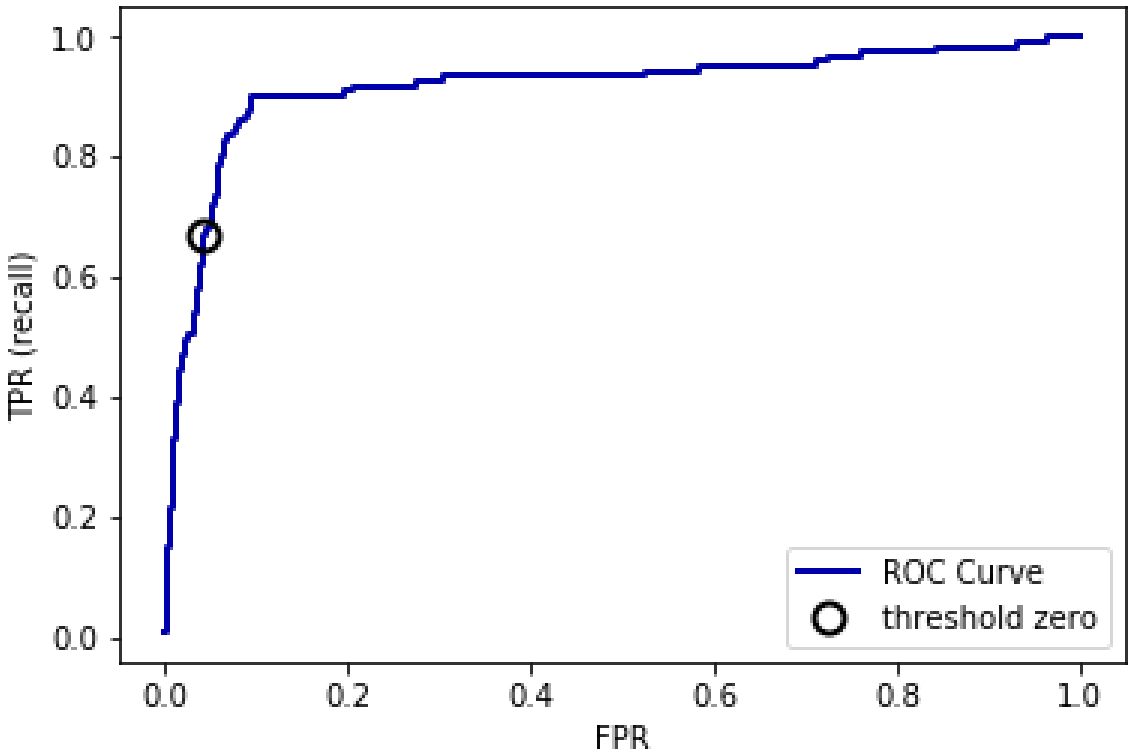
True Positive Rate and False Positive Rate

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

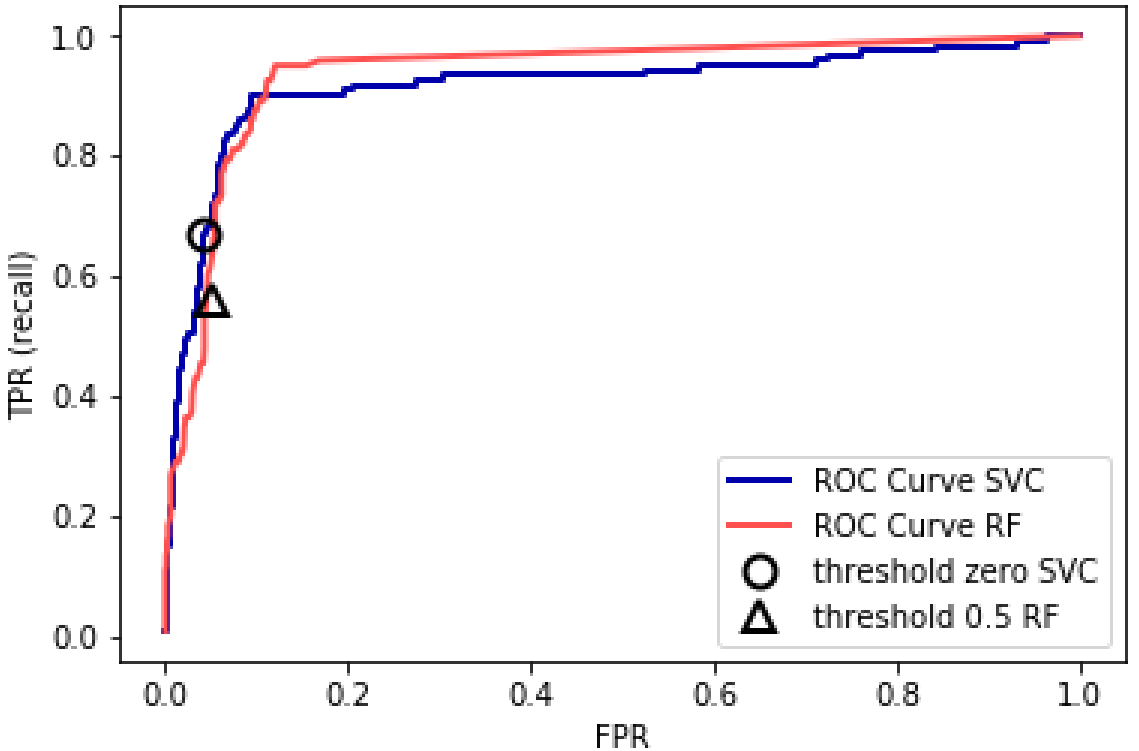
(What is another name for the TPR?)

ROC Curve (SVM)



"Introduction to Machine Learning with Python", Müller and Guido

ROC Curve (SVM vs RF)



"Introduction to Machine Learning with Python", Müller and Guido

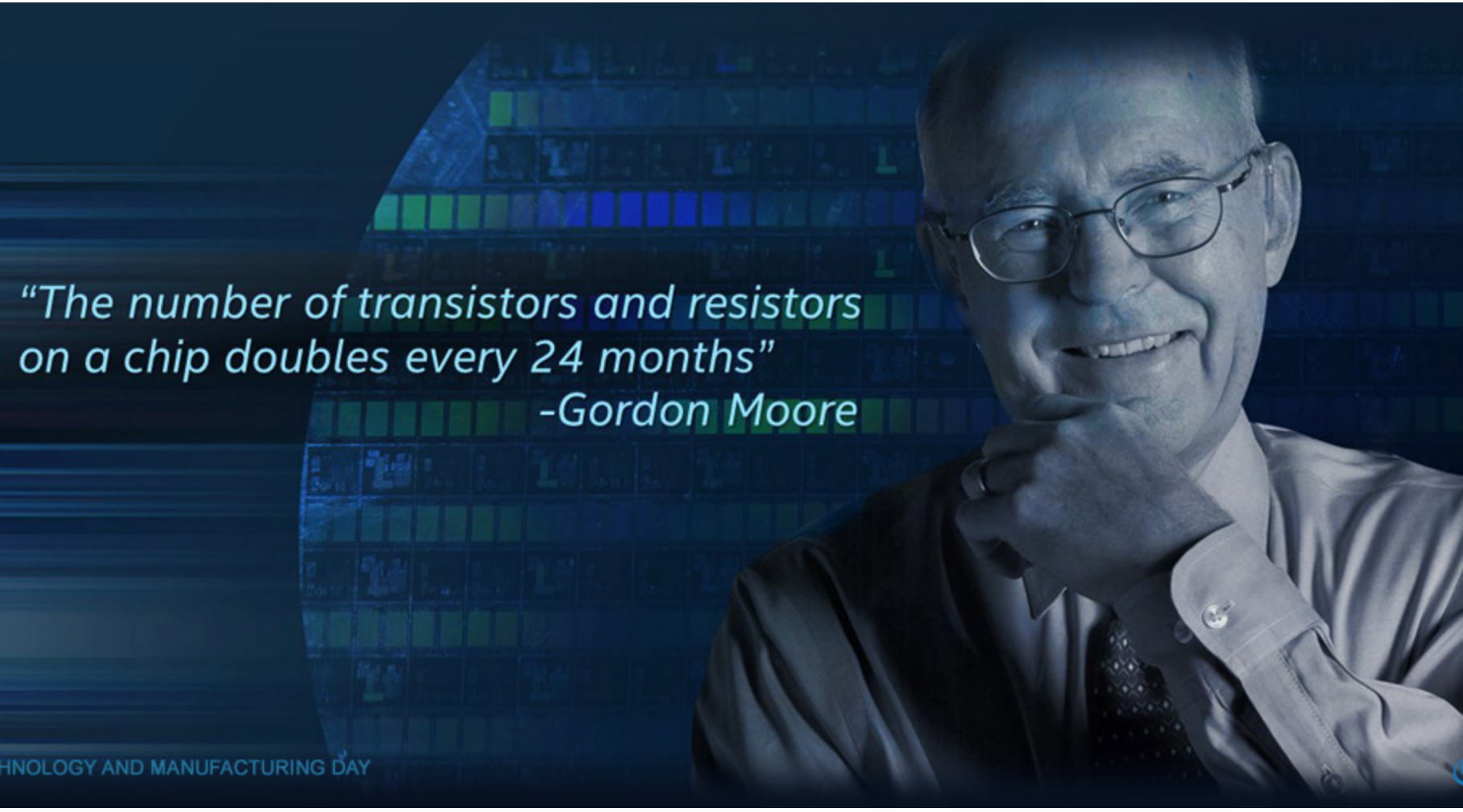
Exercise: Model Evaluation

(open the notebook named **`Exercise 16 - Model Evaluation.ipynb`**)

Surveys: Check your email for a survey link

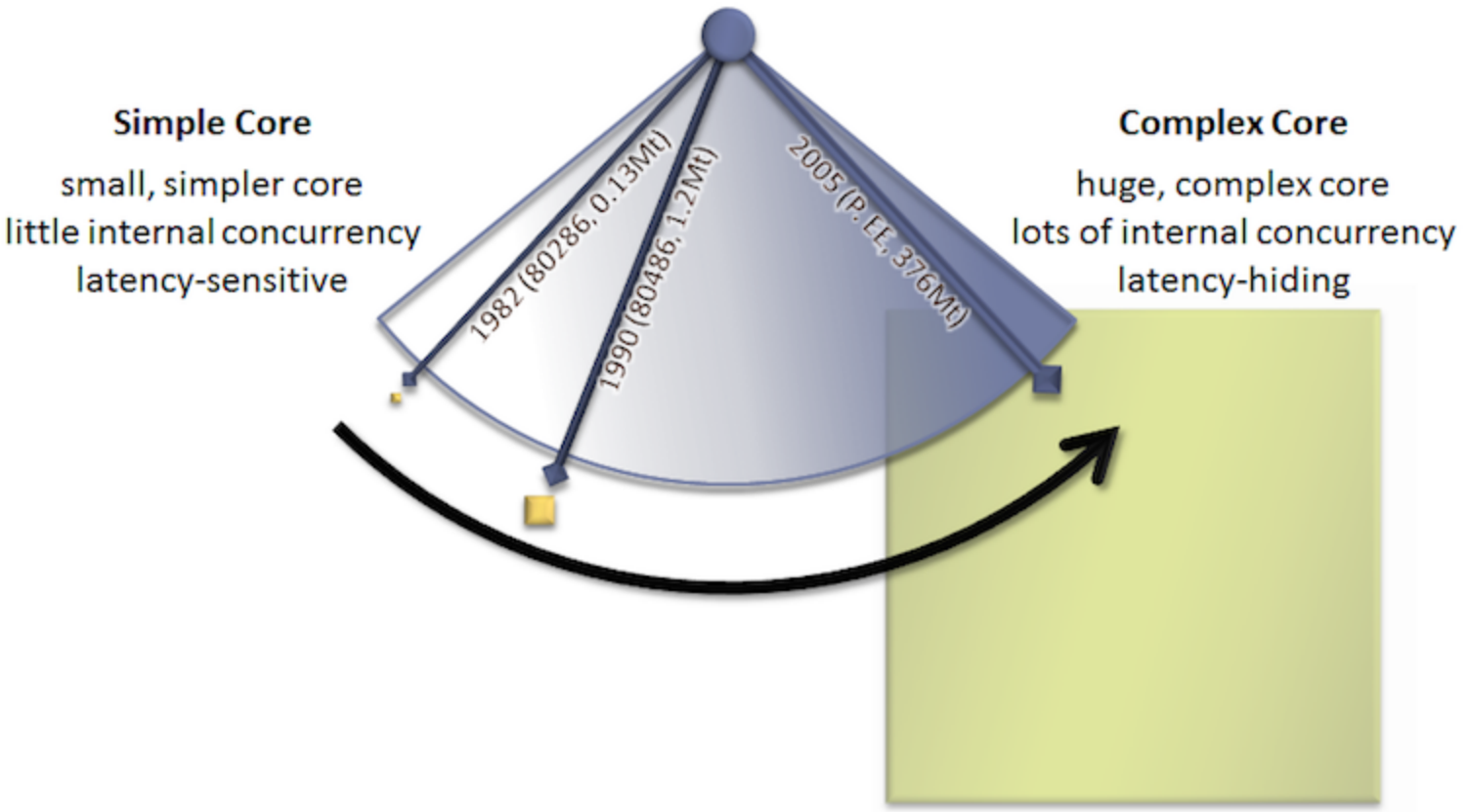
(Complete the survey now - takes 2 minutes)

Performance

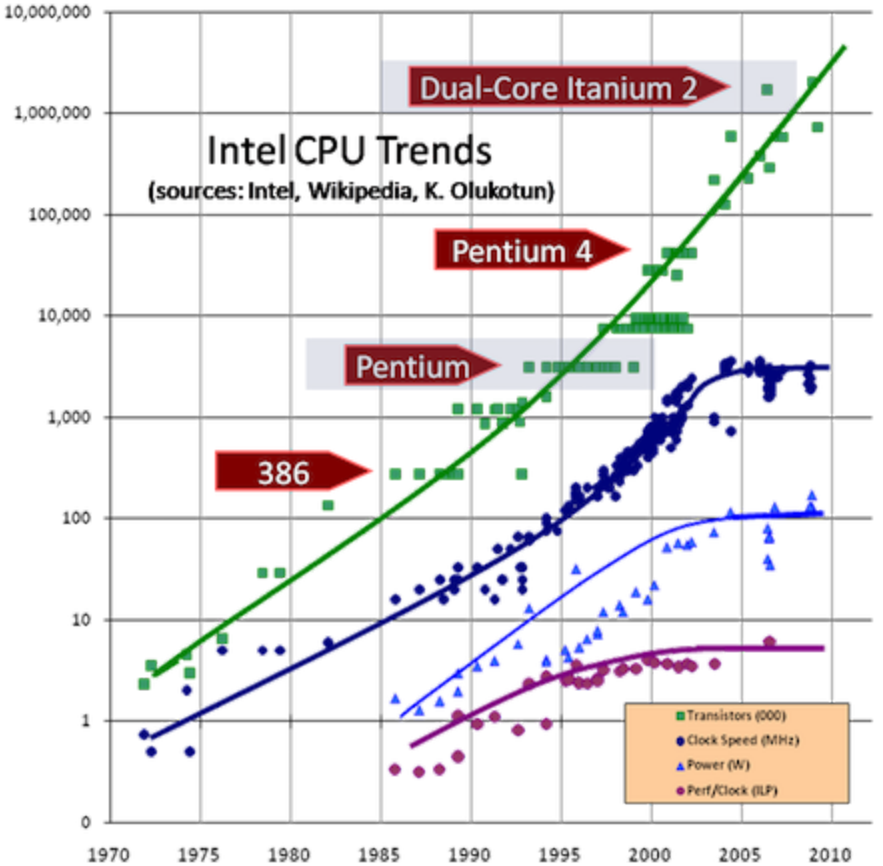


https://newsroom.intel.com/editorials/moores-law-setting-the-record-straight/

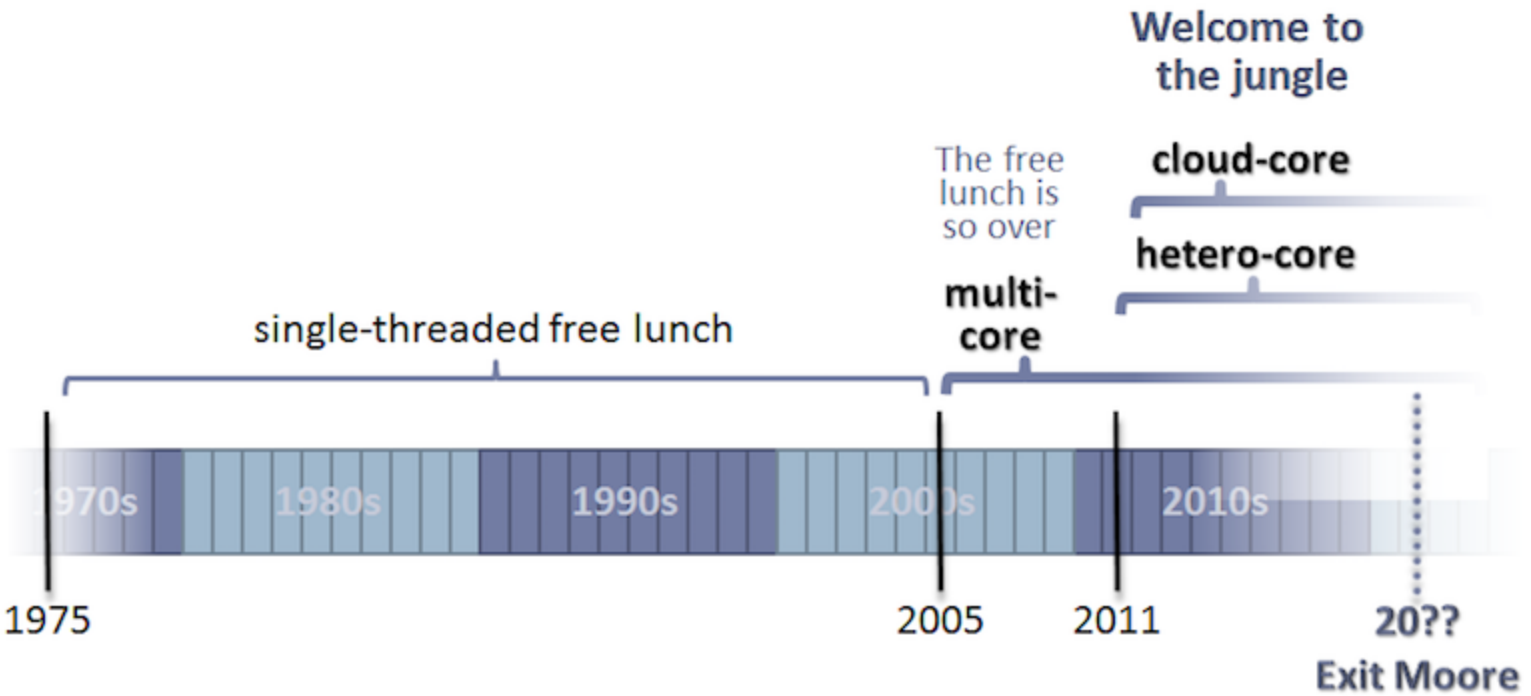
"What Andy giveth, Bill taketh away..."



https://herbsutter.com/welcome-to-the-jungle/

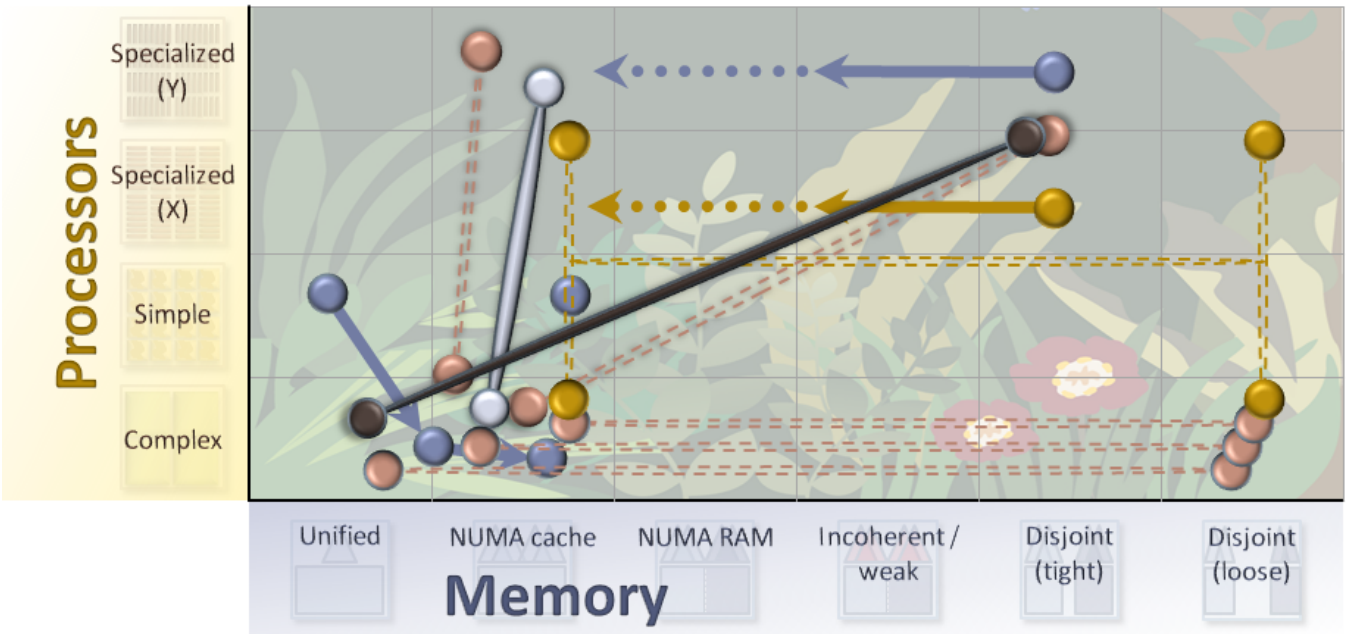


http://www.gotw.ca/publications/concurrency-ddj.htm



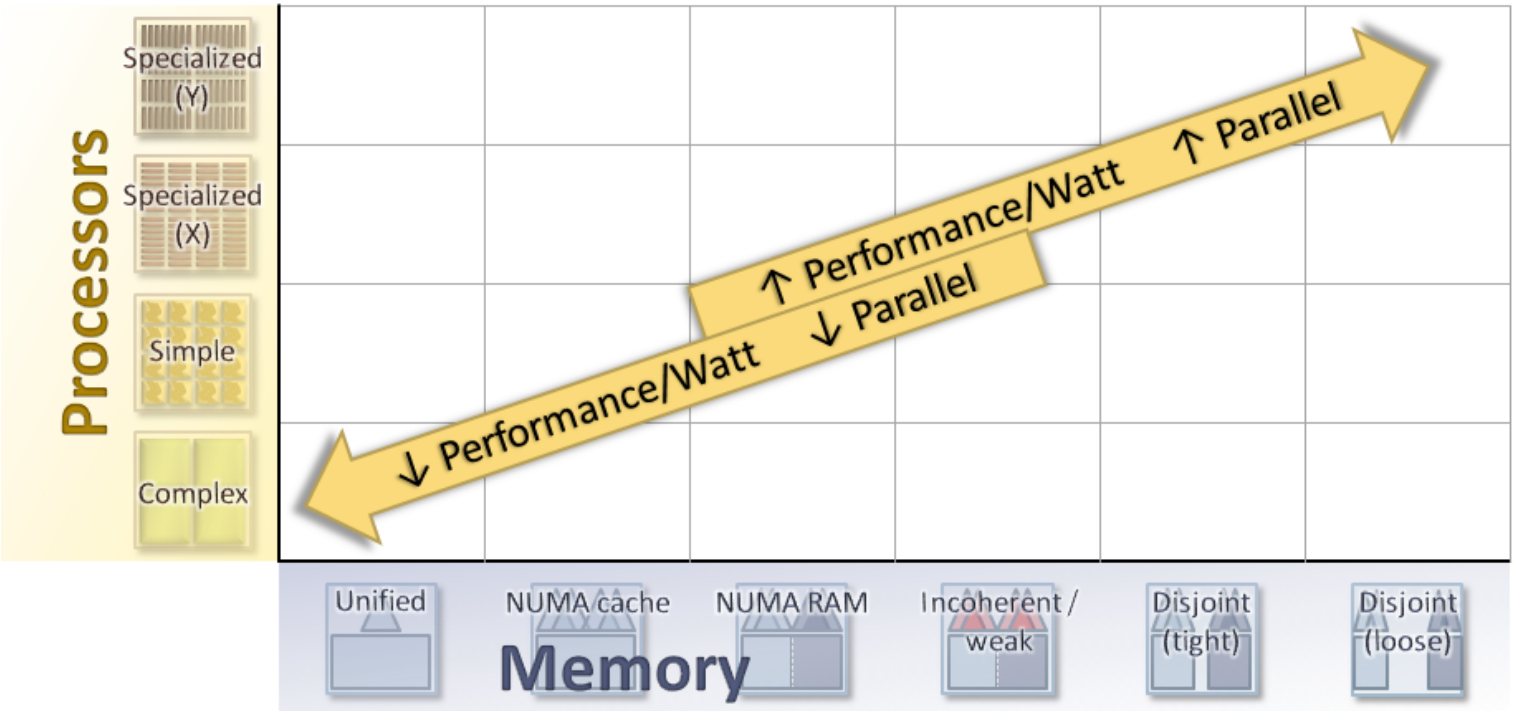
https://herbsutter.com/welcome-to-the-jungle/

The Jungle

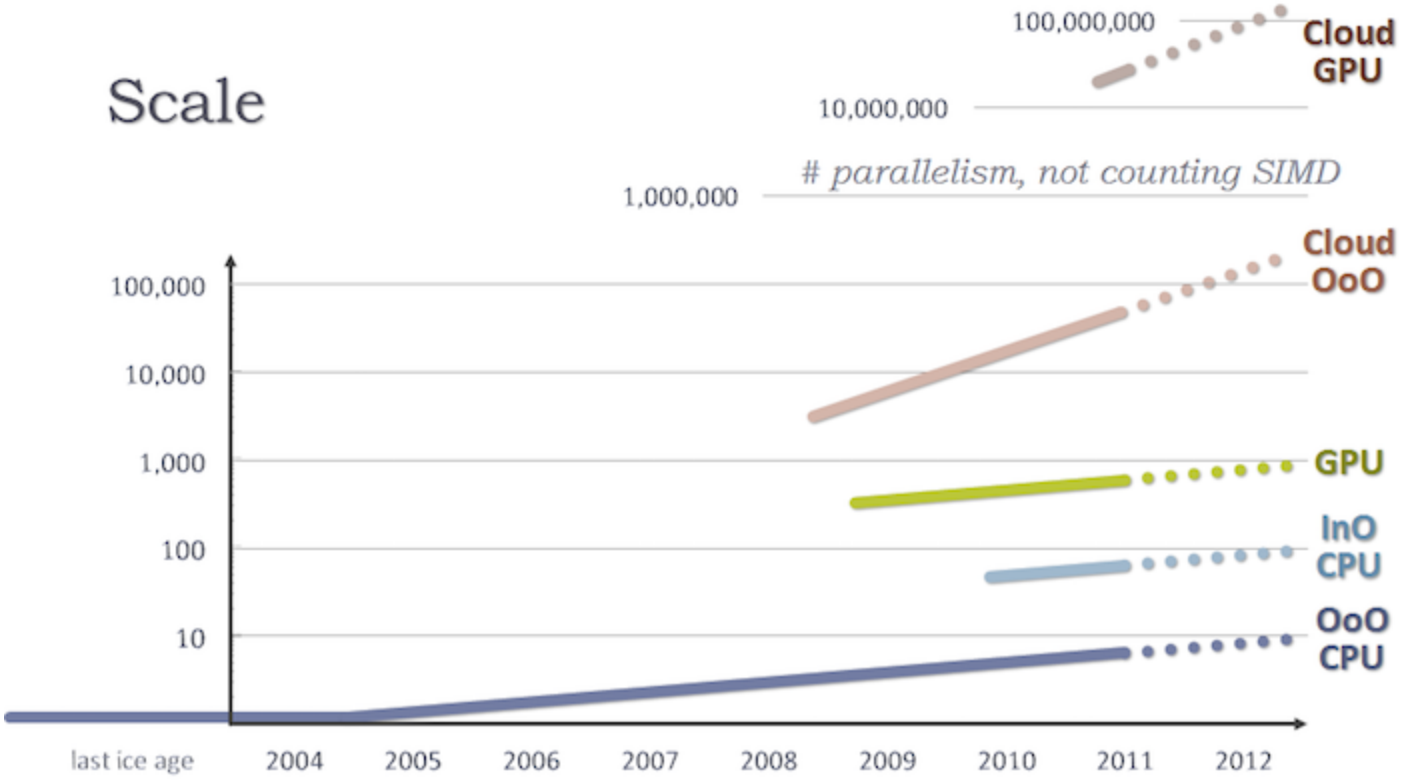


https://herbsutter.com/welcome-to-the-jungle/

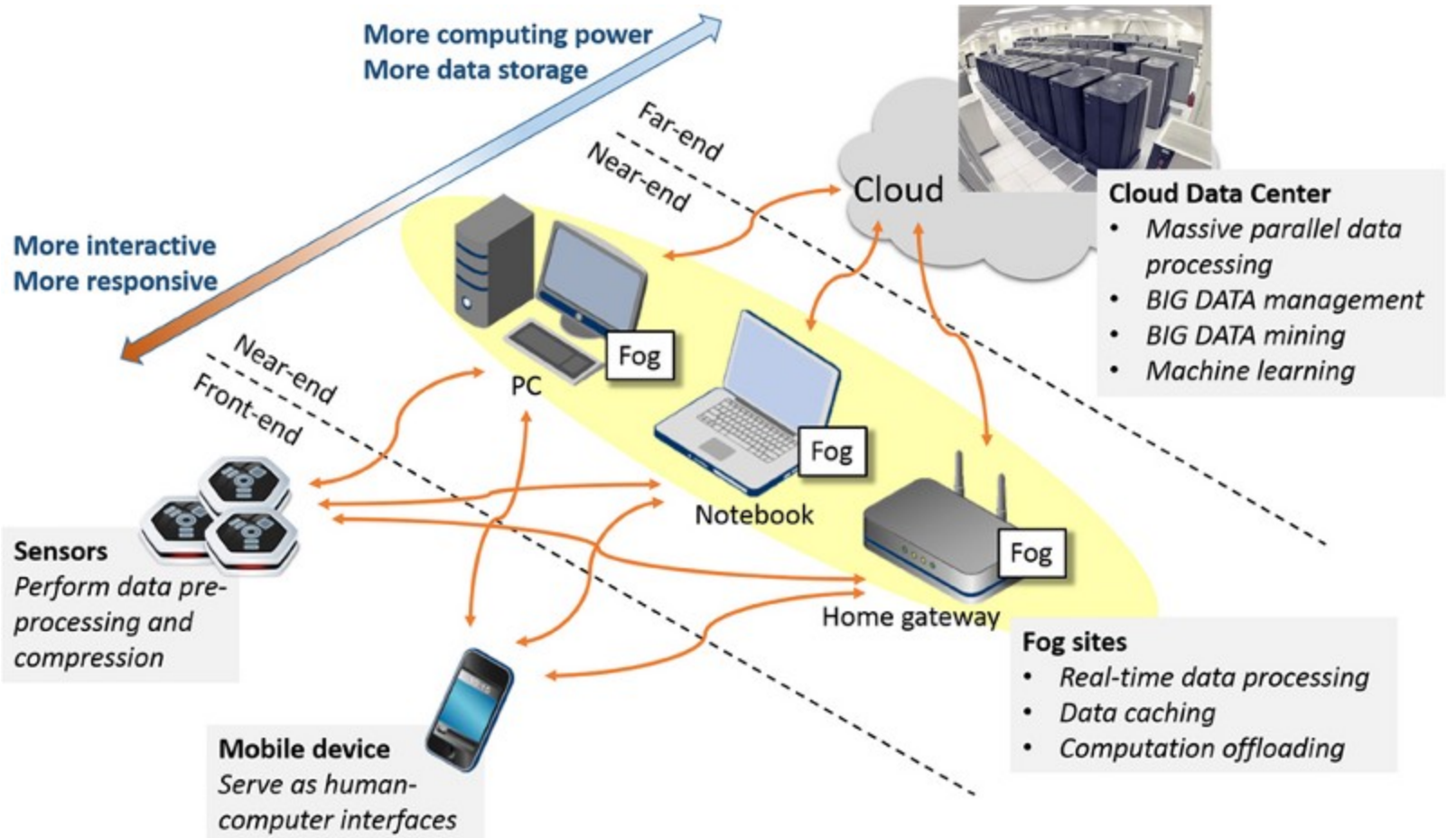
Charting the Landscape



https://herbsutter.com/welcome-to-the-jungle/



https://herbsutter.com/welcome-to-the-jungle/



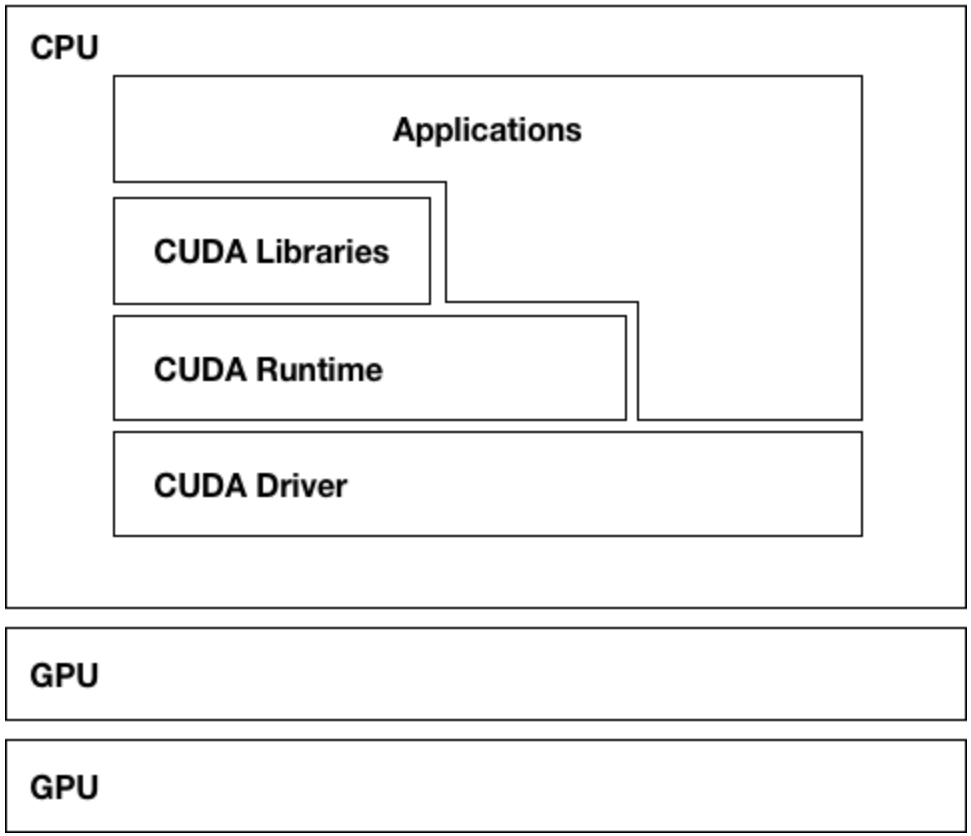
https://www.energomonitor.com/insight/edge-computing-fog-computing-benefits-of-fogging/

"Mainstream hardware is becoming permanently parallel, heterogeneous, and distributed. These changes are permanent, and so will permanently affect the way we have to write performance-intensive code on mainstream architectures."

https://herbsutter.com/welcome-to-the-jungle/

CUDA

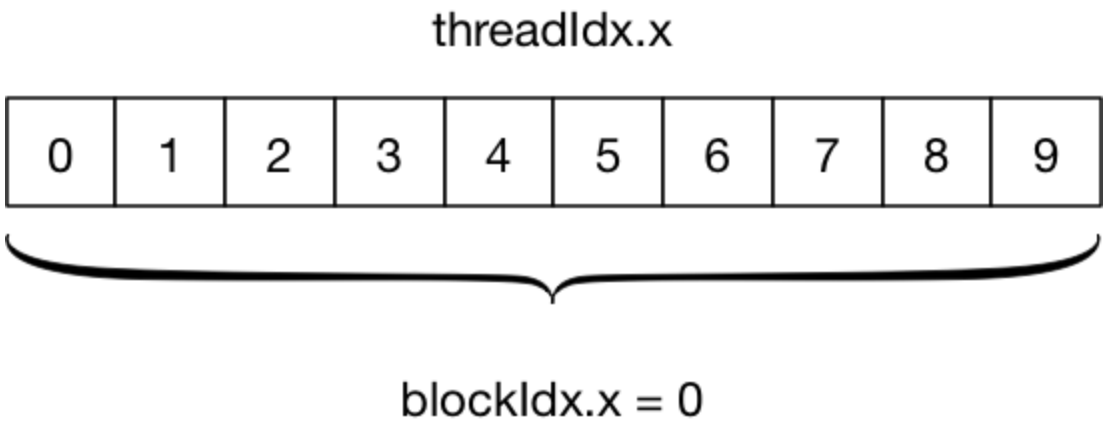
CUDA Architecture



"Professional CUDA C Programming"

```
#include <stdio.h>

__global__ void helloFromGPU(void)
{
    printf("Hello, World from the GPU!\n");
}

int main(void)
{
  printf("Hello, World from the CPU!\n");

  helloFromGPU <<<1, 10>>>();
  cudaDeviceReset();
  return 0;
}
```

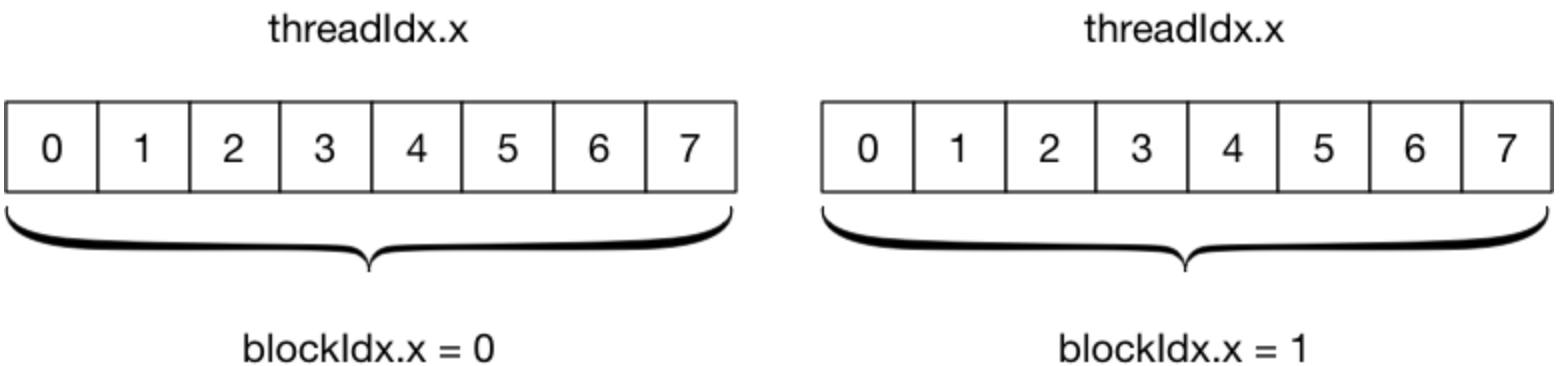
```
helloFromGPU <<<1, 10>>>();
```



```
$ nvcc helloworld.cu -o helloworld
```

```
$ ./helloworld
Hello, World from the CPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
Hello, World from the GPU!
```

```
helloFromGPU <<<2, 8>>>();
```



CUDA Program Structure

1. Allocate GPU memory
2. Copy data from CPU memory to GPU memory
3. Invoke the CUDA kernel
4. Copy data from GPU memory to CPU memory
5. Free GPU memory

Kernel Rules

1. Only access device memory
2. void return type
3. Fixed argument list
4. No static variables
5. No function pointers
6. Asychronous

```
void sumArraysOnHost(float *A,
                     float *B,
                     float *C,
                     const int N)
{
  for( int idx = 0; idx < N; idx++ ) {
    C[idx] = A[idx] + B[idx];
  }
}
```

```
__global__ void sumArraysOnGPU(float *A,
                               float *B,
                               float *C)
{
*  int i = threadIdx.x;
  C[i] = A[i] + B[i];
}
```

```
int main(int argc, char **argv) {
   int dev = 0;
   cudaSetDevice(dev);

   int nElem = 32;

   size_t nBytes = nElem * sizeof(float);

   float *h_A, *h_b, *hostRef, *gpuRef;
   h_A = (float *) malloc(nBytes);
   h_B = (float *) malloc(nBytes);
   gpuRef = (float *) malloc(nBytes);
   ...
}
```

```
...
initialData(h_A, nElem);
initialData(h_B, nElem);

memset(gpuRef, 0, nBytes);

float *d_A, *d_B, *d_c;
cudaMalloc((float **) &d_A, nBytes);
cudaMalloc((float **) &d_B, nBytes);
cudaMalloc((float **) &d_C, nBytes);

cudaMemcpy(d_A, h_A, nBytes,
  cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, nBytes,
  cudaMemcpyHostToDevice);

...
```

```
    ...
    dim3 block (nElem);
    dim3 grid (nElem/block.x);

    sumArraysOnGPU<<< grid, block >>>
       (d_A, d_B, d_C);

    cudaMemcpy(gpuRef, d_C, nBytes,
      cudaMemcpyDeviceToHost);

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    free(h_A);
    free(h_B);
    free(gpuRef);

    return(0);
}
```

"Traditionally, when evaluating hardware platforms for acceleration, one must inevitably consider the trade-off between flexibility and performance."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

"On one end of the spectrum, general purpose processors (GPP) provide a high degree of flexibility and ease of use, but perform relatively inefficiently. These platforms tend to be more readily accessible, can be produced cheaply, and are appropriate for a wide variety of uses and reuses."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

"On the other end of the spectrum, application specific integrated circuits (ASIC) provide high performance at the cost of being inflexible and more difficult to produce. These circuits are dedicated to a specific application, and are expensive and time consuming to produce."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

"FPGAs serve as a compromise between these two extremes. They belong to a more general class of programmable logic devices (PLD) and are, in the most simple sense, a reconfigurable integrated circuit. As such, they provide the performance benefits of integrated circuits, with the reconfigurable flexibility of GPPs."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

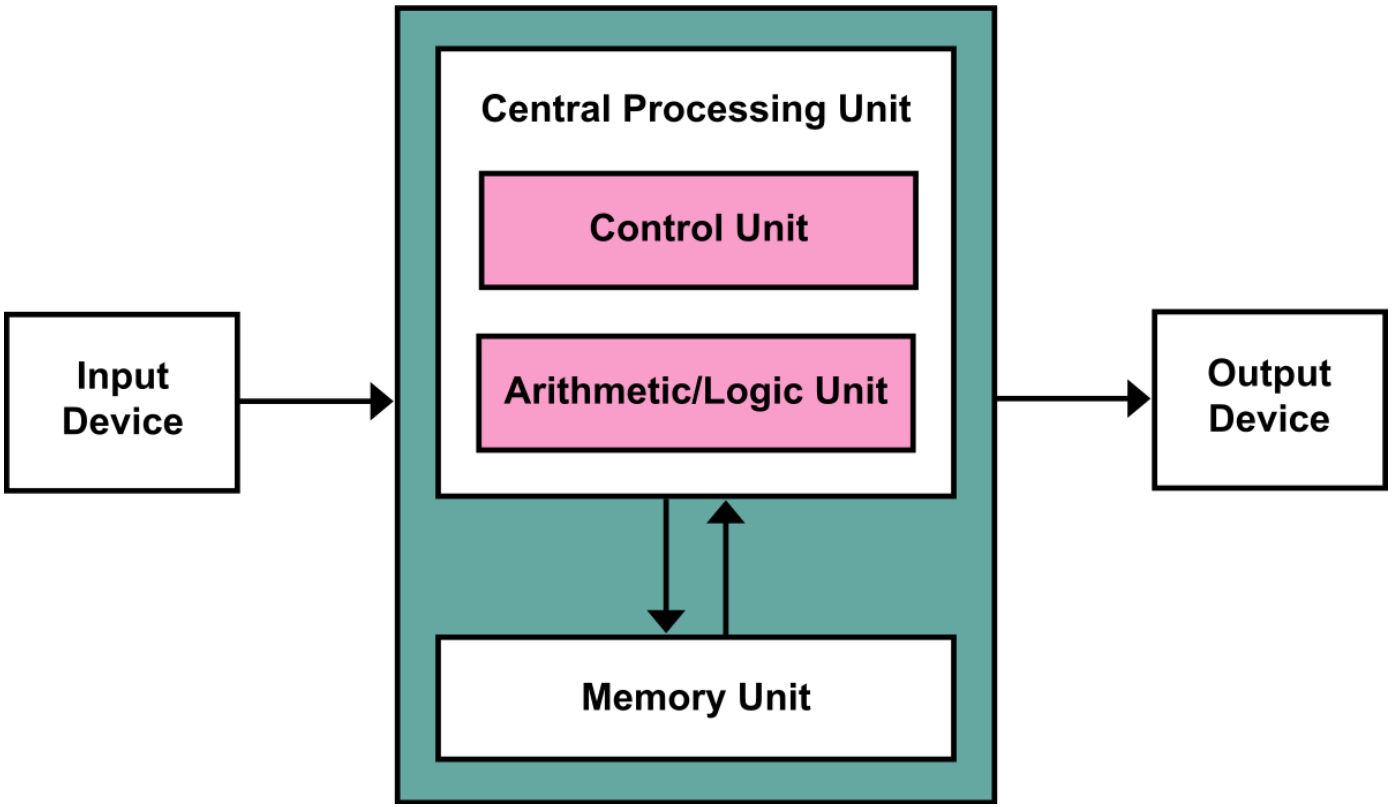
CPUs vs GPUs vs FPGAs vs ASICs

Bitcoin mining

- High-end CPU might get you 20,000,000 H/s
  - Several hundred thousand years to find a solution
- GPUs can get you 200,000,000 H/s
  - Hundreds of years to find a solution
- Run them in parallel, but they generate heat and are more optimized for floating point numbers
- Availability of a Verilog design for Bitcoin mining offered FPGA solutions - Better use of transistors, ran cooler, good at bit twiddling
  - Offered 1,000,000,000 H/s
  - Chaining 100 of these would allow you to find a solution… in 100 years
- Application-specific integrated circuits (ASICs)

Marco Krohn

Von Neumann Architecture



https://en.wikipedia.org/wiki/Von_Neumann_architecture

"In comparison, the programmable logic cells on FPGAs can be used to implement the data and control path found in common logic functions, which do not rely on the Von Neumann architecture. They are also capable of exploiting distributed on-chip memory, as well as large degrees of pipeline parallelism, which fit naturally with the feed-forward nature deep learning methods."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

"Modern FPGAs also support partial dynamic reconfiguration, where part of the FPGA can be reprogrammed while another part of the FPGA is being used. This can have implications for large deep learning models, where individual layers could be reconfigured on the FPGA while not disrupting ongoing computation in other layers."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

"With GPUs and other fixed architectures, a software execution model is followed, and structured around executing tasks in parallel on independent compute units. As such, the goal in developing deep learning techniques for GPUs is to adapt algorithms to follow this model, where computation is done in parallel, and data interdependence is ensured."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

"In contrast, FPGA architecture is tailored for the application. When developing deep learning techniques for FPGAs, there is less emphasis on adapting algorithms for a fixed computational structure, allowing more freedom to explore algorithm level optimizations."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

"Techniques which require many complex low-level hardware control operations which cannot be easily implemented in high-level software languages are especially attractive for FPGA implementations."

"Deep Learning on FPGAs: Past, Present, and Future", Lacey et al.

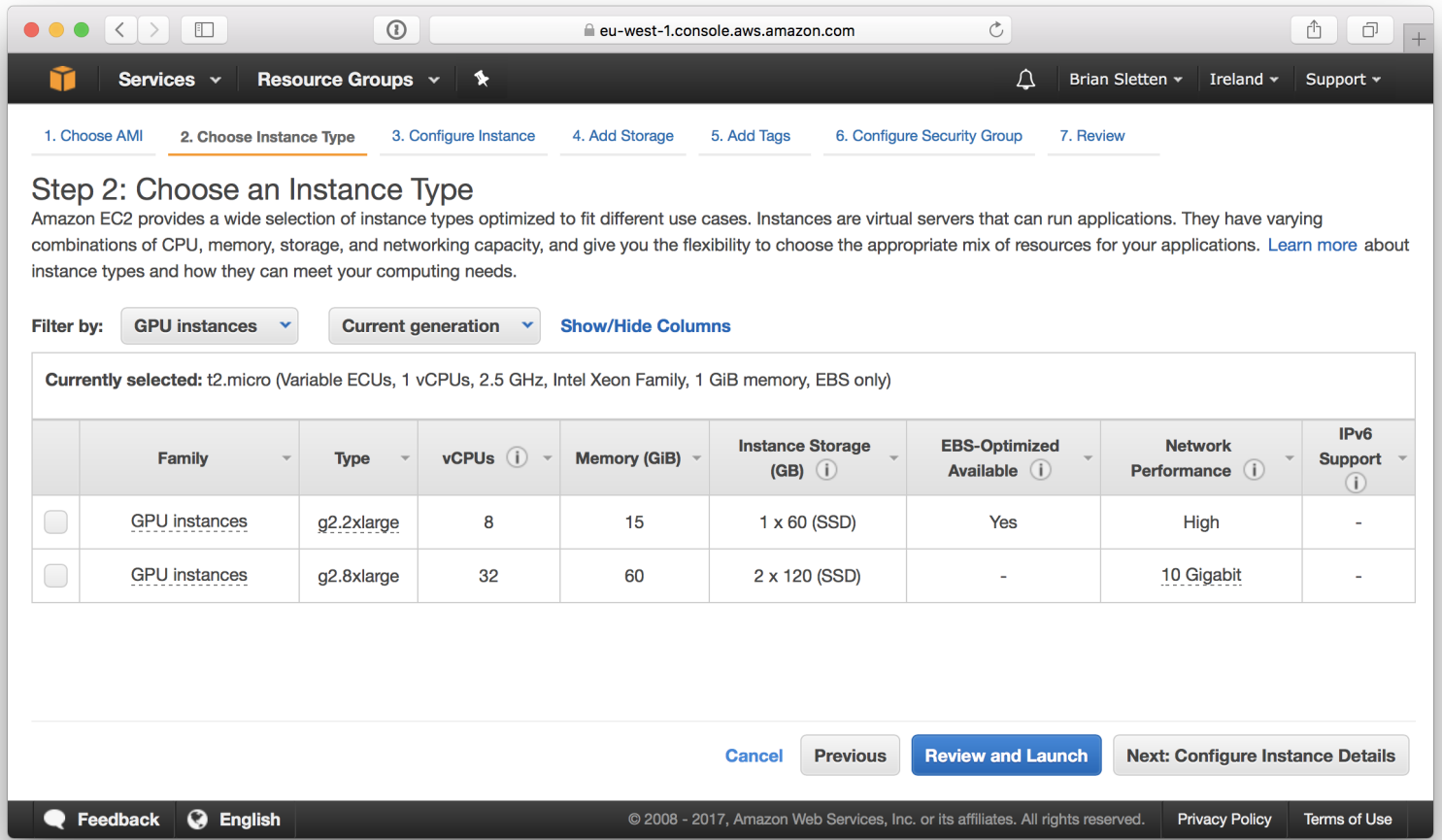

OpenCL

OpenCL Support

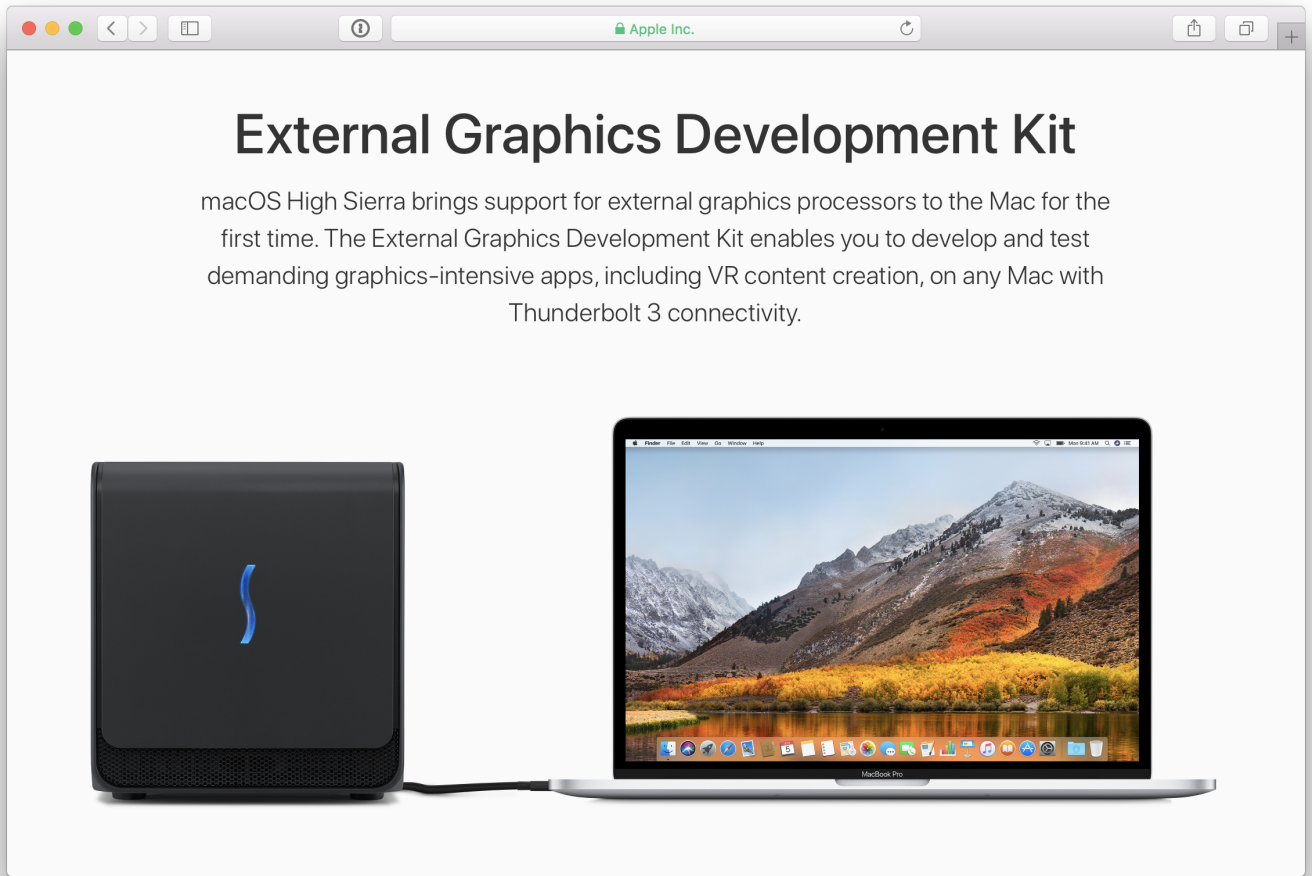
- GPPs
- GPUs
- DSPs
- FPGAs

The Beast

AWS GPU Instances



Apple eGPU Developer Kit



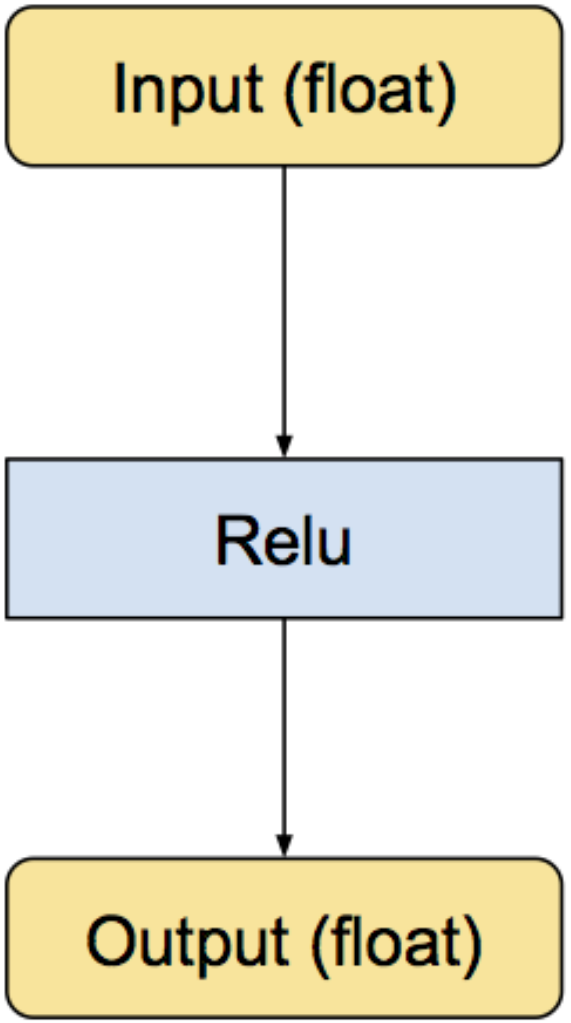
https://mathema.tician.de/software/pycuda/

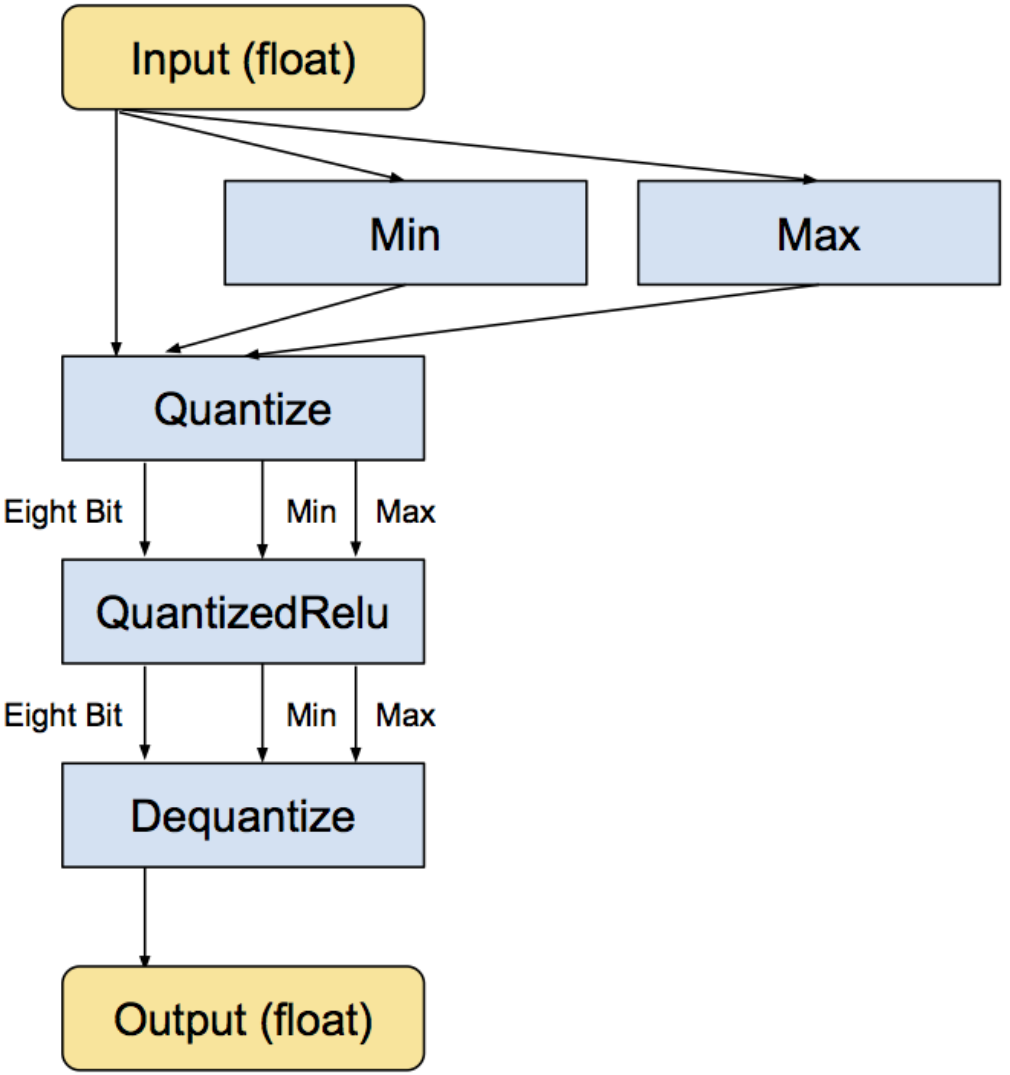
Quantizing Models

- Models tend to get large
- Trained off of floating point precision
- Compression of floating point numbers
- Can gain storage and inference performance through quantization

Quantizing Process

- Find min and max values
- Map min to 0, max to 255
- TensorFlow provides quantized-aware versions of ops


Input (float)
Relu
Output (float)


Input (float)
Min
Max
Quantize
Eight Bit
Min
Max
QuantizedRelu
Eight Bit
Min
Max
Dequantize
Output (float)

Product-based Exercise

Resources

Brian Sletten's ML Gist

Salesforce Internal Links

"POD Risk Assessment - Working Group" Gus Chatter Group

Infra Data Science blog

Horizon Pod Risk assessment dashboard

Horizon Pod scorecards

Capacity Optimization Gus Chatter Group