

Motivation Contd.



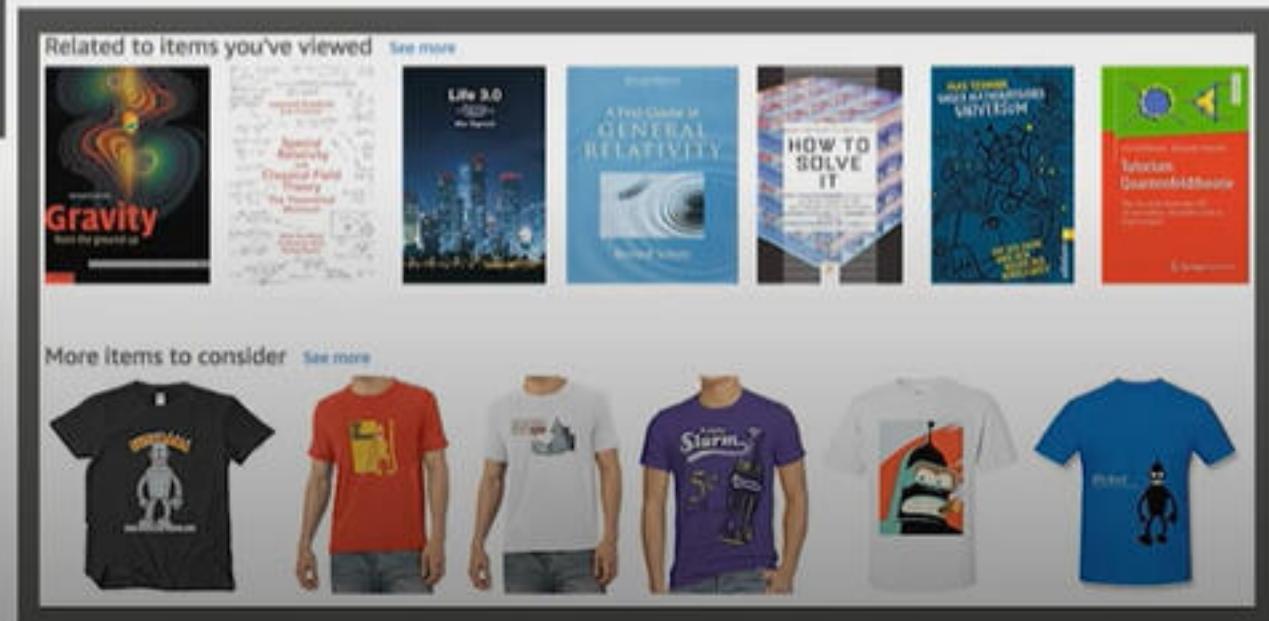
New for you



Inspired by your shopping trends



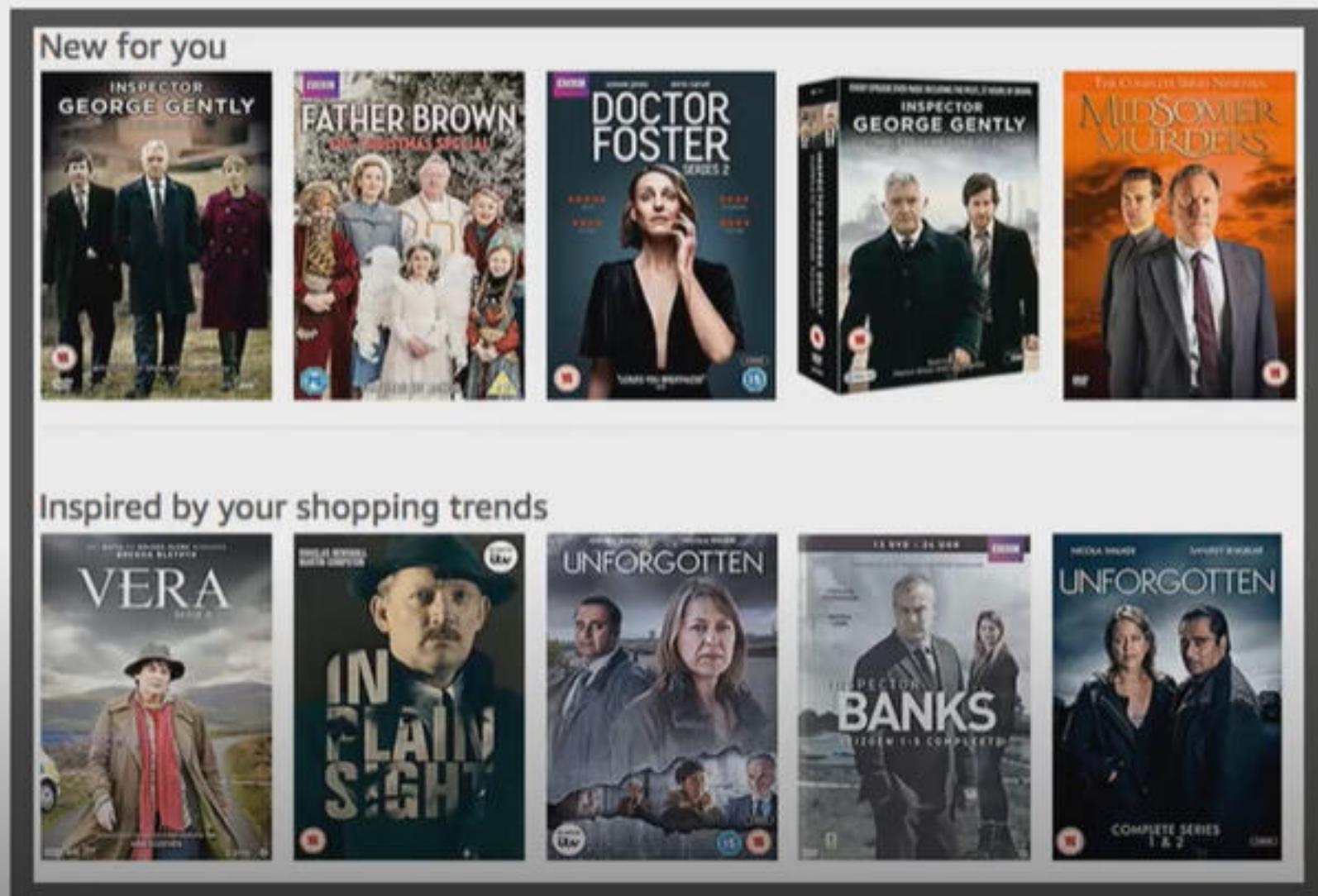
My Profile – amazon.co.uk



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Collaborative Filtering Contd.

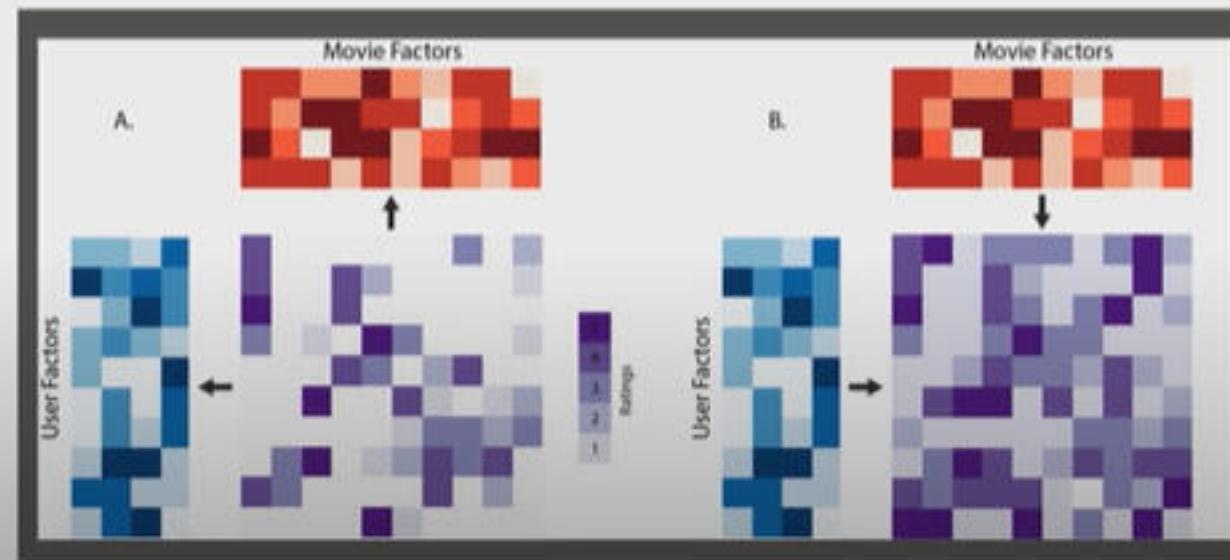
aws training and certification



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Matrix Factorization

- **Matrix Factorization** factorizes a matrix to separate matrices, that when multiplied approximate to the completed matrix.
- For the sake of efficiency we would like to factorize the matrix to a long and a wide matrices.



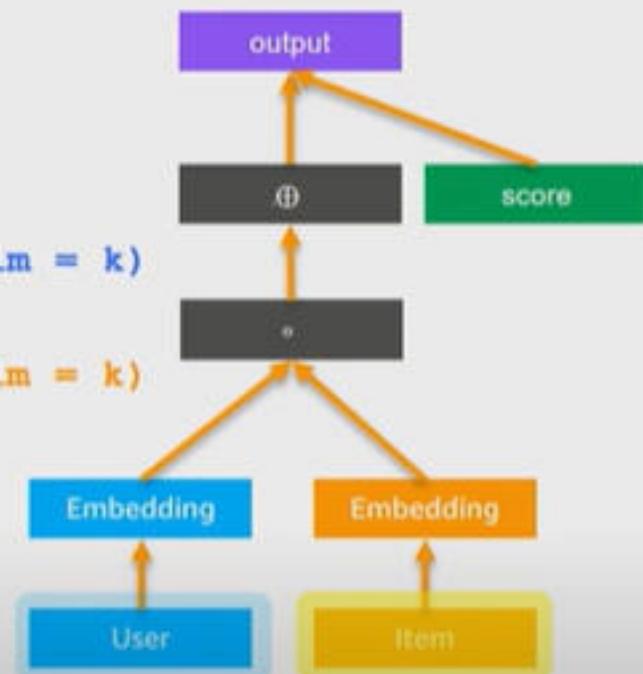
<https://www.udacity.com/course/deep-learning-linear-algebra-machine-learning-classic-data-science-principles-engineering-introduction-ud730>

Linear Model

```
def plain_net(k):
    # input
    user = mx.symbol.Variable('user')
    item = mx.symbol.Variable('item')
    score = mx.symbol.Variable('score')
    # user feature lookup
    user = mx.symbol.Embedding(data = user, input_dim = max_user, output_dim = k)
    # item feature lookup
    item = mx.symbol.Embedding(data = item, input_dim = max_item, output_dim = k)
    # predict by the inner product, which is elementwise product and then sum
    pred = user * item
    pred = mx.symbol.sum(data = pred, axis = 1)
    pred = mx.symbol.Flatten(data = pred)
    # loss layer
    pred = mx.symbol.LinearRegressionOutput(data = pred, label = score)
    return pred
```

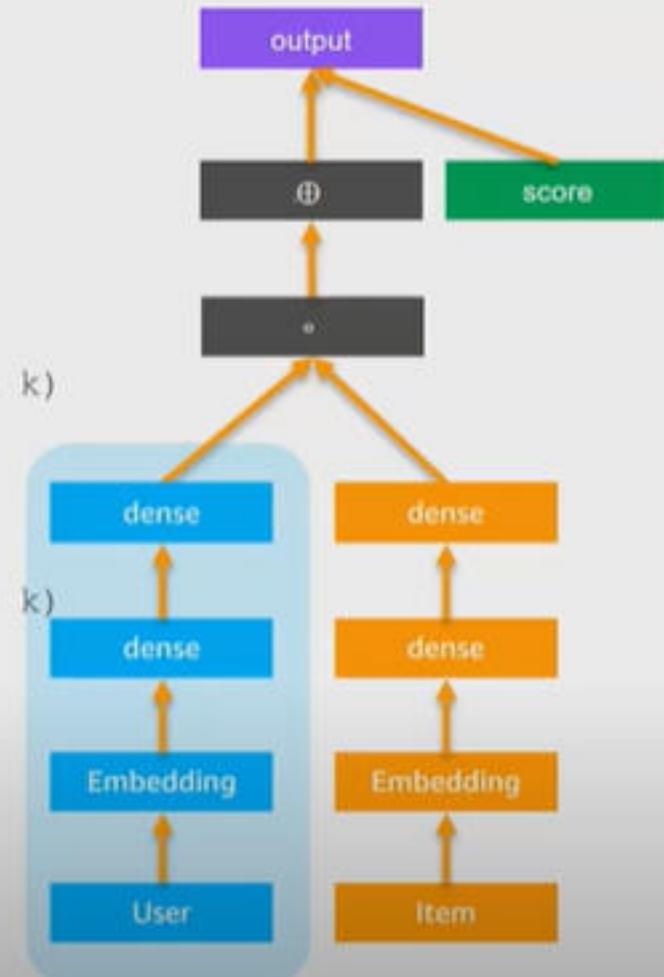
```
net1 = plain_net(64)
mx.viz.plot_network(net1)
```

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



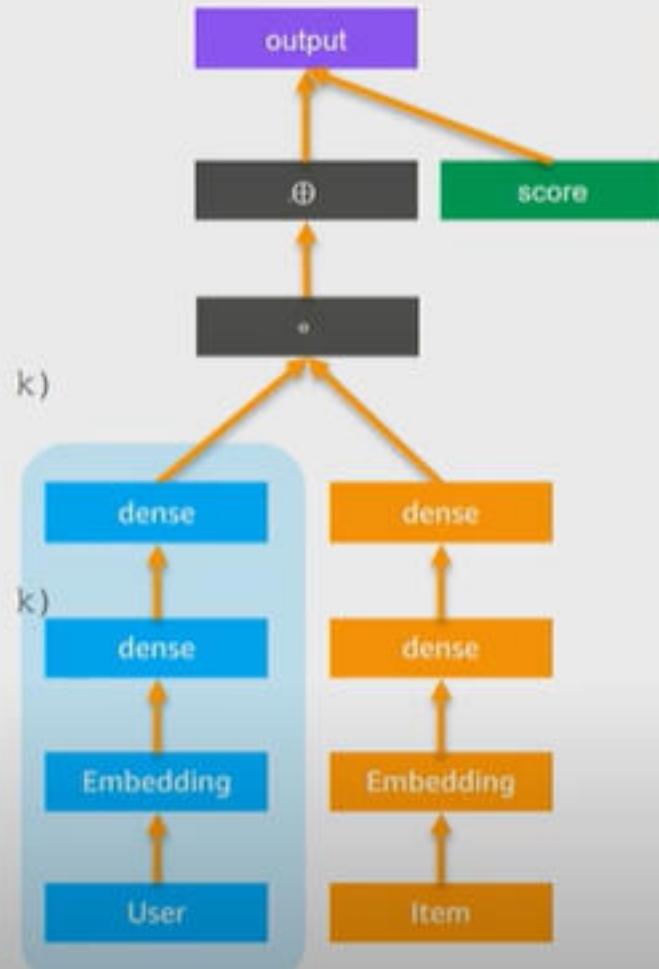
Adding Non-Linearity

```
def get_one_layer_mlp(hidden, k):  
    # input  
    user = mx.symbol.Variable('user')  
    item = mx.symbol.Variable('item')  
    score = mx.symbol.Variable('score')  
  
    # user latent features  
    user = mx.symbol.Embedding(data = user, input_dim = max_user, output_dim = k)  
    user = mx.symbol.Activation(data = user, act_type='relu')  
    user = mx.symbol.FullyConnected(data = user, num_hidden = hidden)  
  
    # item latent features  
    item = mx.symbol.Embedding(data = item, input_dim = max_item, output_dim = k)  
  
    # predict by the inner product  
    pred = user * item  
    pred = mx.symbol.sum(data = pred, axis = 1)  
    pred = mx.symbol.Flatten(data = pred)  
  
    # loss layer  
    pred = mx.symbol.LinearRegressionOutput(data = pred, label = score)  
    return pred
```



Adding Non-Linearity

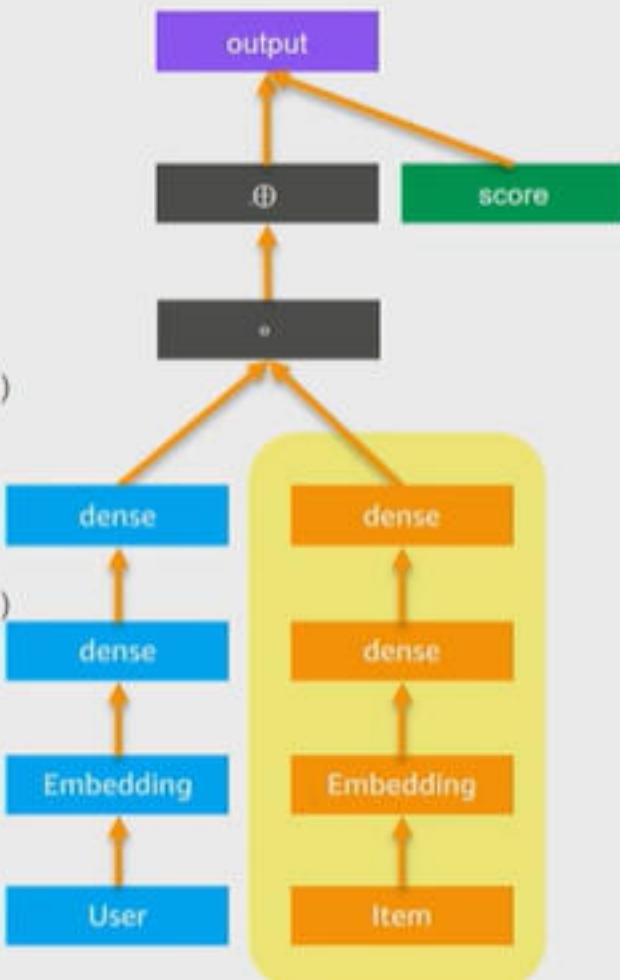
```
def get_one_layer_mlp(hidden, k):  
    # input  
    user = mx.symbol.Variable('user')  
    item = mx.symbol.Variable('item')  
    score = mx.symbol.Variable('score')  
  
    # user latent features  
    user = mx.symbol.Embedding(data = user, input_dim = max_user, output_dim = k)  
    user = mx.symbol.Activation(data = user, act_type='relu')  
    user = mx.symbol.FullyConnected(data = user, num_hidden = hidden)  
  
    # item latent features  
    item = mx.symbol.Embedding(data = item, input_dim = max_item, output_dim = k)  
  
    # predict by the inner product  
    pred = user * item  
    pred = mx.symbol.sum(data = pred, axis = 1)  
    pred = mx.symbol.Flatten(data = pred)  
  
    # loss layer  
    pred = mx.symbol.LinearRegressionOutput(data = pred, label = score)  
    return pred
```



Adding Non-Linearity Contd.



```
def get_one_layer_mlp(hidden, k):  
    # input  
    user = mx.symbol.Variable('user')  
    item = mx.symbol.Variable('item')  
    score = mx.symbol.Variable('score')  
  
    # user latent features  
    user = mx.symbol.Embedding(data = user, input_dim = max_user, output_dim = k)  
    user = mx.symbol.Activation(data = user, act_type='relu')  
    user = mx.symbol.FullyConnected(data = user, num_hidden = hidden)  
  
    # item latent features  
    item = mx.symbol.Embedding(data = item, input_dim = max_item, output_dim = k)  
    item = mx.symbol.Activation(data = item, act_type='relu')  
    item = mx.symbol.FullyConnected(data = item, num_hidden = hidden)  
  
    # predict by the inner product  
    pred = user * item  
    pred = mx.symbol.sum(data = pred, axis = 1)  
    pred = mx.symbol.Flatten(data = pred)  
  
    # loss layer  
    pred = mx.symbol.LinearRegressionOutput(data = pred, label = score)  
    return pred
```



Embedding Layer



- An Embedding Layer is where a network extracts the importance of features from data.
- Embedding is frequently used in NLP. For instance in sentiment analysis, embedding distills sentiment information from words.



https://www.oreilly.com/ideas/deep-matrix-factorization-using-apache-mxnet?cmp=tw-data-na-article-engagement_sponsored+kibird

Loading Data into an Array



```
train_data_iter = gluon.data.DataLoader(SparseMatrixDataset(train_data, train_label),
                                         shuffle=True, batch_size=batch_size)

test_data_iter = gluon.data.DataLoader(SparseMatrixDataset(test_data, test_label),
                                         shuffle=True, batch_size=batch_size)

class SparseMatrixDataset(gluon.data.Dataset):
    def __init__(self, data, label):
        assert data.shape[0] == len(label)
        self.data = data
        self.label = label
        if isinstance(label, ndarray.NDArray) and len(label.shape) == 1:
            self._label = label.asnumpy()
        else:
            self._label = label

    def __getitem__(self, idx):
        return self.data[idx, 0], self.data[idx, 1], self.label[idx]

    def __len__(self):
        return self.data.shape[0]
```

Defining the Network



```
class MFBlock(gluon.Block):
    def __init__(self, max_users, max_items, num_emb, dropout_p=0.5):
        super(MFBlock, self).__init__()
        self.max_users = max_users
        self.max_items = max_items
        self.dropout_p = dropout_p
        self.num_emb = num_emb
        with self.name_scope():
            self.user_embeddings = gluon.nn.Embedding(max_users, num_emb)
            self.item_embeddings = gluon.nn.Embedding(max_items, num_emb)

    def forward(self, users, items):
        a = self.user_embeddings(users)
        b = self.item_embeddings(items)
        predictions = a * b
        predictions = nd.sum(predictions, axis=1)
        return predictions
```

Choosing the Optimizer



```
trainer = gluon.Trainer(net.collect_params(), 'sgd',
{'learning_rate': lr, 'wd': wd, 'momentum': 0.9})
```

Training the Model



```
epochs = 10
def train(data_iter, net):
    for e in range(epochs):
        print("epoch: {}".format(e))
        for i, (user, item, label) in enumerate(train_data_iter):
            user = user.as_in_context(ctx).reshape((batch_size,))
            item = item.as_in_context(ctx).reshape((batch_size,))
            label = label.as_in_context(ctx).reshape((batch_size,))
            with mx.autograd.record():
                output = net(user, item)
                loss = loss_function(output, label)
                loss.backward()
                net.collect_params().values()
            trainer.step(batch_size)
        print("EPOCH {}: RMSE ON TRAINING and TEST: {}, {}".format(e,
                                                                    eval_net(train_data_iter, net),
                                                                    eval_net(test_data_iter, net)))
    return output
```

Adding Non-Linearity



```
class MFBlock(gluon.Block):
    def __init__(self, max_users, max_items, num_emb, dropout_p=0.5):
        super(MFBlock, self).__init__()
        self.max_users = max_users
        self.max_items = max_items
        self.dropout_p = dropout_p
        self.num_emb = num_emb
        with self.name_scope():
            self.user_embeddings = gluon.nn.Embedding(max_users, num_emb)
            self.item_embeddings = gluon.nn.Embedding(max_items, num_emb)
            self.dense = gluon.nn.Dense(num_emb, activation='relu')

    def forward(self, users, items):
        a = self.user_embeddings(users)
        a = self.dense(a)

        b = self.item_embeddings(items)
        b = self.dense(b)

        predictions = a * b
        predictions = nd.sum(predictions, axis=1)
        return predictions
```

Limitations



- Matrix Factorization is ideal for small catalogues and can perform based on small amounts of data.
- As the catalogues get larger, memory becomes a challenge.

Limitations Contd.

- For instance MovieLens 20M has 27278 items and 138493 users. User x Item matrix will have a dimension of $138,493 \times 27,278 = 3,777,812,054$. From all possible ratings the dataset includes only 20000263. This means only 0.05 percent of the dataset contains data and 99.95 percent is just sparsity.

Storing the Matrix

Dense

3.7B entries

Each entry:

- Rating: 1 byte

3.7 GB

Sparse

20M non-zero entries

Each entry:

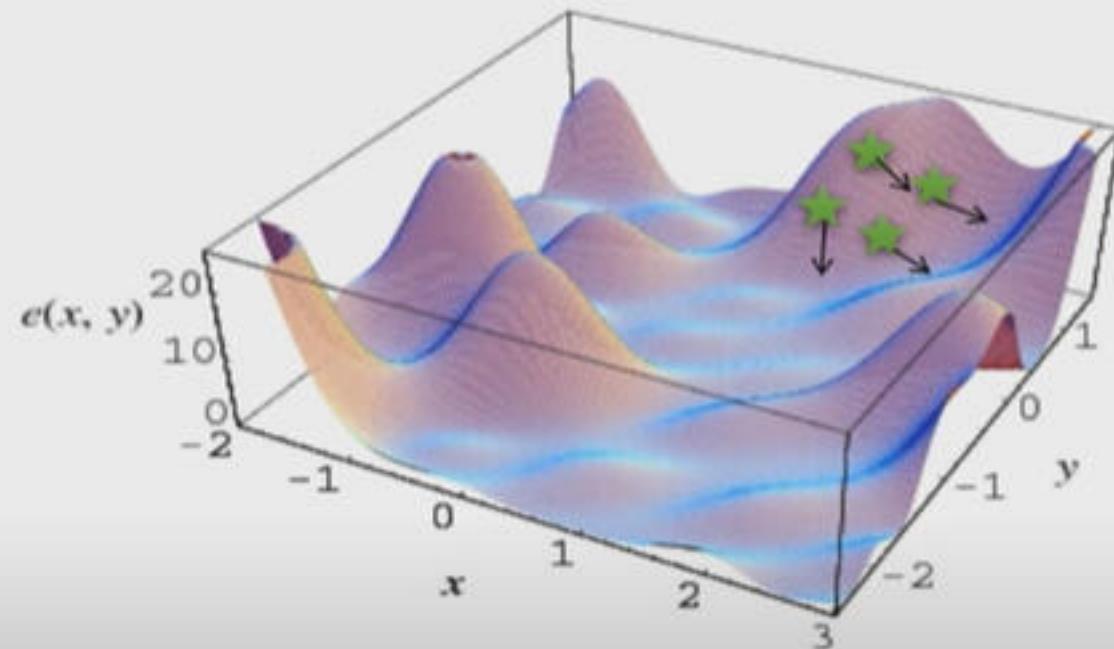
- Rating: 1 byte
- Movie_id: 32-bit integer
- User_id: 32-bit integer

180 MB

Sparse is 20x smaller!

The Scaling Issue Contd.

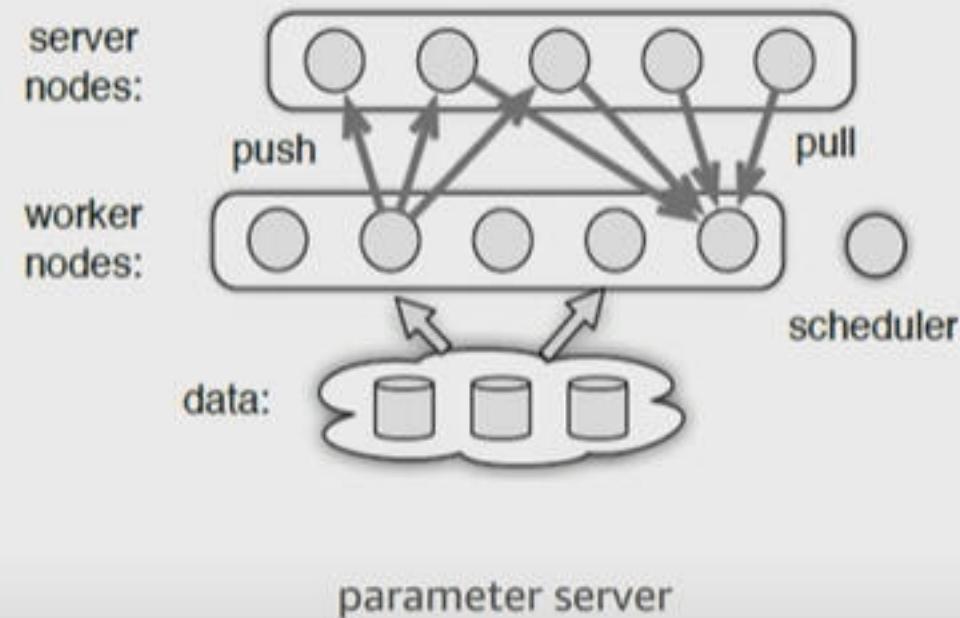
- DiFacto or Distributed Factorization Machines are capable of distributing computation of sub-gradients on mini-batches asynchronously and can thus distribute load to several machines.



asynch SGD

The Scaling Issue

- Factorization Machines take advantage of combining Support Vector Machines and MF in order to scale and deal with sparsity.
- The problem is that FM runs on a single machine and has huge memory requirements.



(Alex Smola et al. - 2016)

asynch SGD

Content (Feature)-Based



- We might not have user data, but there is often a wealth of product information available. We can use this data in order to recommend similar products to a user.

code	category	sub category	weight	price	colour	dimentions
itm1	1	1	20	123.1	blue	23x12x19
itm2	2	1	20	900	white	23x12x20
itm3	1	1	22	123.1	green	20x10x18
itm4	3	1	20	600	blue	23x12x22
itm5	1	2	19	200	yellow	23x12x23
itm6	1	1	1	12	red	2x1x3
itm7	1	1	900	2000	blue	100x80x99
itm8	9	8	20	6000	grey	99x99x99
itm9	7	5	1000	123.1	blue	123x5x8
itm10	9	8	20	5000	brown	99x99x99

Content Based

aws training and certification

Related to items you've viewed [See more](#)

The image shows six book covers arranged horizontally. From left to right: 1. 'Gravity from the ground up' by Brian Cox. 2. 'Special Relativity and Classical Field Theory: The Theoretical Minimum' by Leonard Susskind & Art Friedman. 3. 'Life 3.0: Being Human in the Age of Artificial Intelligence' by Max Tegmark. 4. 'A First Course in GENERAL RELATIVITY' by Bernard Schutz, Second Edition. 5. 'HOW TO SOLVE IT: A NEW METHOD OF MATHEMATICAL THINKING' by George Polya. 6. 'MAX TEGMARK: UNDERRÄTTEN OM MATEMATISCHES UNIVERSUM' (German edition) by Max Tegmark.

More items to consider [See more](#)

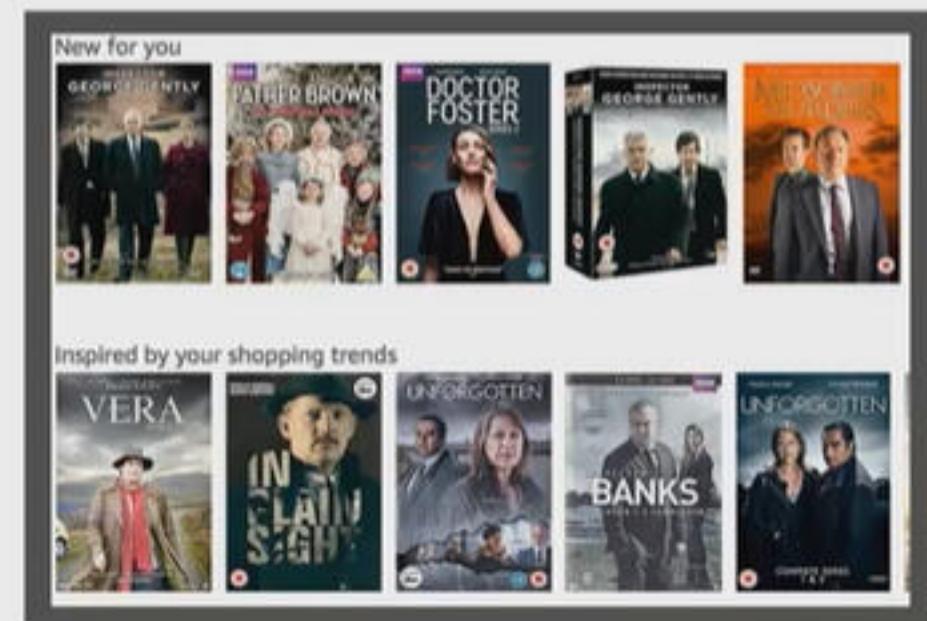
The image shows six t-shirts arranged horizontally. From left to right: 1. Black t-shirt with a cartoon robot wearing a crown. 2. Red t-shirt with a yellow graphic design. 3. White t-shirt with a small blue graphic. 4. Purple t-shirt with a graphic of a beer bottle and the word 'Slurm'. 5. White t-shirt with a colorful toucan graphic. 6. Blue t-shirt with a black cartoon robot graphic.

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Untapped Data

- There is still a wealth of information we have not tapped into.
 - Movies have images.
 - Images can be captioned.
 - Products have names, while we have so far reduced them to item numbers.
 - TV series have episode names.
 - Products have verbose reviews.
- We can harvest all of this information and enrich our recommendation system.

There is a strong similarity in the ambience and composition of these images:

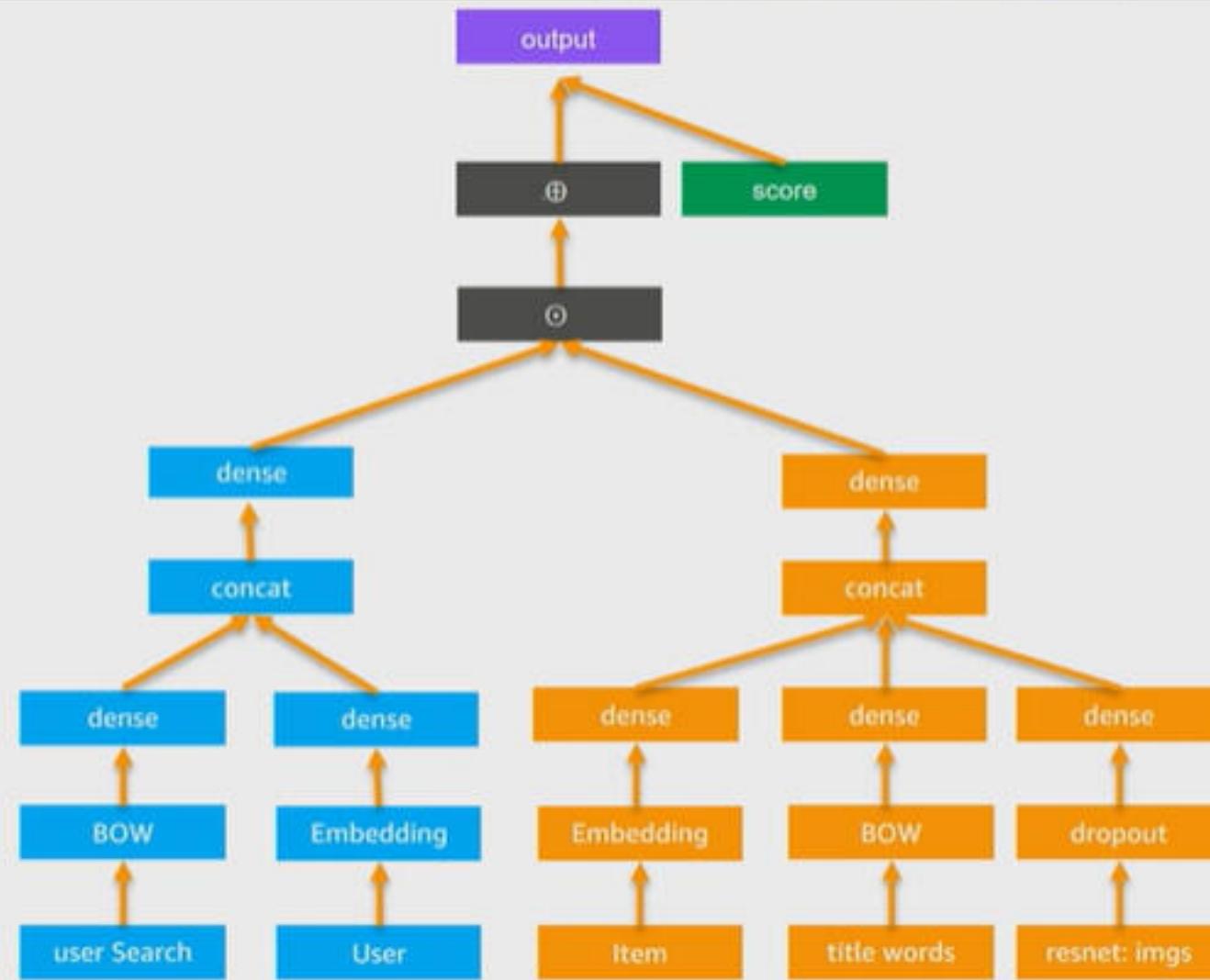


Deep Structure Semantic Models



- A Matrix Factorization solution in its core is multiplication of two matrices.
- Neural Networks are good at picking up semantic intent at phrase/sentence level.
- Neural Networks are great at image captioning.
- The output of a network is a tensor.
- So we can use the output of several networks as our embedding layer for an enriched recommendation system.

DSSM Contd.



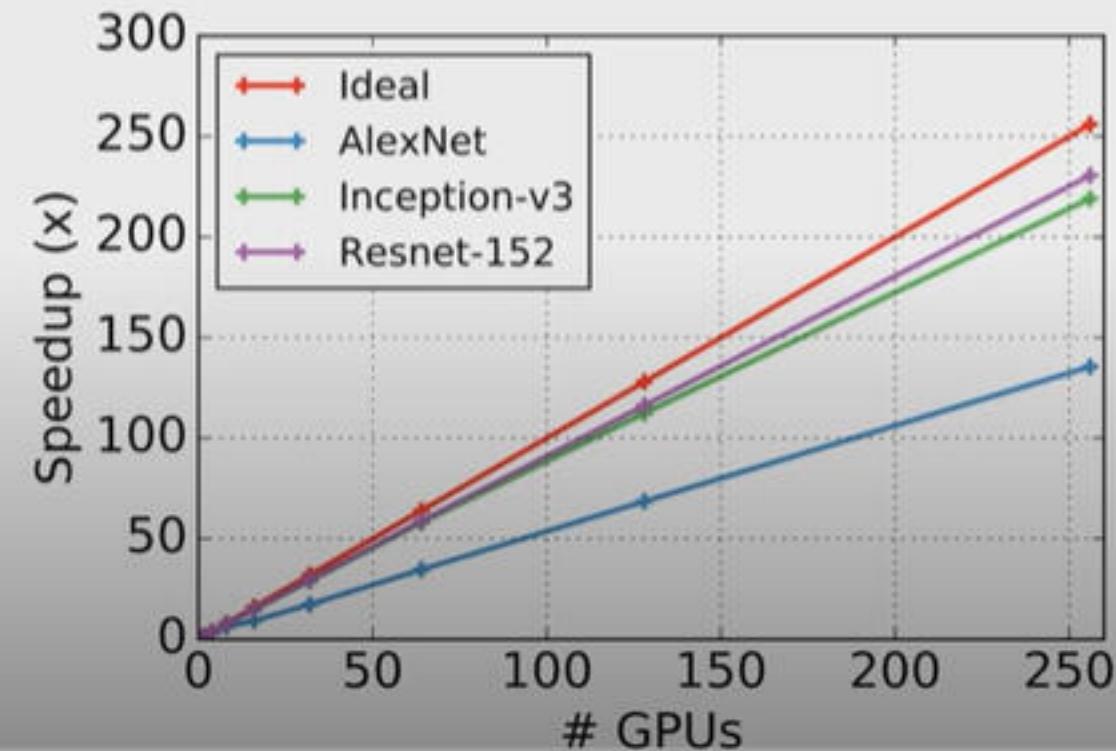
Demo - Matrix Factorization Using Your Own Code and the Amazon SageMaker Factorization Machine

DataSet



User Id	Movie Id	Rating	Timestamp
1	1	5	874965758
1	2	3	876893171
1	3	4	878542960
1	4	3	876893119
1	5	3	889751712

- We are using MXNet and Gluon for coding.
- MXNet benchmark shows near linear performance across multiple machines.
- Gluon is a Pytorch-like imperative API for MXNet.



Amazon SageMaker



- A **fully managed** service that enables **data scientists** and **developers** to quickly and easily **build** machine-learning based models into **production** smart applications.
- Amazon SageMaker includes several built-in state of the art algorithms that are fine-tuned to run on distributed environments.
<https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html>
- Amongst built-in algorithms there is a Factorization Machine algorithm implemented using MXNet.
- All you need to do it to tune model's hyperparameters and passing your data to the model.



A set of small, semi-transparent icons used for navigating and interacting with the Jupyter notebook interface.

code in part inspired by: <https://github.com/EthanRosenthal/torchmf>

```
In [19]: import os
import mxnet as mx
from mxnet import gluon, nd, ndarray

import pandas as pd
import numpy as np
```

```
In [20]: data_path = 'ml-100k/'
num_emb = 64
opt = 'Adam'
lr = 0.02
mnmtm = 0.
wd = 0.
batch_size = 50
ctx = mx.gpu(4)
```

```
In [3]: def download_ml_data(prefix):
    if not os.path.exists("%s.zip" % prefix):
        print("Downloading MovieLens data: %s" % prefix)
        os.system("wget http://files.grouplens.org/datasets/movielens/%s.zip" % prefix)
        os.system("unzip %s.zip" % prefix)
```

```
In [4]: download_ml_data('data')

Downloading MovieLens data: data
```

```
In [5]: def max_id(fname):
    mu = 0
    mi = 0
    with open(fname) as f:
        for line in f:
            tks = line.strip().split('\t')
            if len(tks) != 4:
                continue
            mu = max(mu, int(tks[0]))
            mi = max(mi, int(tks[1]))
    return mu + 1, mi + 1
max_users, max_items = max_id(data_path + 'u.data')
```

```
In [6]: train_df = pd.read_csv(data_path+'ul.base', header=None, sep='\t')
test_df = pd.read_csv(data_path+'ul.test', header=None, sep='\t')

train_data = nd.array(train_df[[0,1]].values, dtype=np.float32)
```

```
In [19]: import os
import mxnet as mx
from mxnet import gluon, nd, ndarray

import pandas as pd
import numpy as np
```

```
In [20]: data_path = 'ml-100k/'
num_emb = 64
opt = 'Adam'
lr = 0.02
mmntm = 0.
wd = 0.
batch_size = 50
ctx = mx.gpu(4)
```

```
In [3]: def download_ml_data(prefix):
    if not os.path.exists("%s.zip" % prefix):
        print("Downloading MovieLens data: %s" % prefix)
        os.system("wget http://files.grouplens.org/datasets/movielens/%s.zip" % prefix)
        os.system("unzip %s.zip" % prefix)
```

```
In [4]: download_ml_data('data')

Downloading MovieLens data: data
```

```
In [5]: def max_id(fname):
    mu = 0
    mi = 0
    with open(fname) as f:
        for line in f:
            tks = line.strip().split('\t')
            if len(tks) != 4:
                continue
            mu = max(mu, int(tks[0]))
            mi = max(mi, int(tks[1]))
    return mu + 1, mi + 1
max_users, max_items = max_id(data_path + 'u.data')
```

```
    return mu + 1, mi + 1
max_users, max_items = max_id(data_path + 'u.data')

In [6]: train_df = pd.read_csv(data_path+'ul.base', header=None, sep='\t')
test_df = pd.read_csv(data_path+'ul.test', header=None, sep='\t')

train_data = nd.array(train_df[[0,1]].values, dtype=np.float32)
train_label = nd.array(train_df[2].values, dtype=np.float32)

test_data = nd.array(test_df[[0,1]].values, dtype=np.float32)
test_label = nd.array(test_df[2].values, dtype=np.float32)
```

```
In [7]: class SparseMatrixDataset(gluon.data.Dataset):
    def __init__(self, data, label):
        assert data.shape[0] == len(label)
        self.data = data
        self.label = label
        if isinstance(label, ndarray.NDArray) and len(label.shape) == 1:
            self._label = label.asnumpy()
        else:
            self._label = label

    def __getitem__(self, idx):
        return self.data[idx, 0], self.data[idx, 1], self.label[idx]

    def __len__(self):
        return self.data.shape[0]
```

```
In [8]: class MFBlock(gluon.Block):
    def __init__(self, max_users, max_items, num_emb, dropout_p=0.5):
        super(MFBlock, self).__init__()

        self.max_users = max_users
        self.max_items = max_items
        self.dropout_p = dropout_p
        self.num_emb = num_emb

        with self.name_scope():
            self.user_embeddings = gluon.nn.Embedding(max_users, num_emb)
            self.item_embeddings = gluon.nn.Embedding(max_items, num_emb)
```

```
    return mu + 1, mi + 1
max_users, max_items = max_id(data_path + 'u.data')

In [6]: train_df = pd.read_csv(data_path+'ul.base', header=None, sep='\t')
test_df = pd.read_csv(data_path+'ul.test', header=None, sep='\t')

train_data = nd.array(train_df[[0,1]].values, dtype=np.float32)
train_label = nd.array(train_df[2].values, dtype=np.float32)

test_data = nd.array(test_df[[0,1]].values, dtype=np.float32)
test_label = nd.array(test_df[2].values, dtype=np.float32)
```

```
In [7]: class SparseMatrixDataset(gluon.data.Dataset):
    def __init__(self, data, label):
        assert data.shape[0] == len(label)
        self.data = data
        self.label = label
        if isinstance(label, ndarray.NDArray) and len(label.shape) == 1:
            self._label = label.asnumpy()
        else:
            self._label = label

    def __getitem__(self, idx):
        return self.data[idx, 0], self.data[idx, 1], self.label[idx]

    def __len__(self):
        return self.data.shape[0]
```

```
In [8]: class MFBlock(gluon.Block):
    def __init__(self, max_users, max_items, num_emb, dropout_p=0.5):
        super(MFBlock, self).__init__()

        self.max_users = max_users
        self.max_items = max_items
        self.dropout_p = dropout_p
        self.num_emb = num_emb

        with self.name_scope():
            self.user_embeddings = gluon.nn.Embedding(max_users, num_emb)
            self.item_embeddings = gluon.nn.Embedding(max_items, num_emb)
```

```
        assert data.shape[0] == len(label)
        self.data = data
        self.label = label
        if isinstance(label, ndarray.NDArray) and len(label.shape) == 1:
            self._label = label.asnumpy()
        else:
            self._label = label

    def __getitem__(self, idx):
        return self.data[idx, 0], self.data[idx, 1], self.label[idx]

    def __len__(self):
        return self.data.shape[0]
```

```
In [8]: class MFBlock(gluon.Block):
    def __init__(self, max_users, max_items, num_emb, dropout_p=0.5):
        super(MFBlock, self).__init__()

        self.max_users = max_users
        self.max_items = max_items
        self.dropout_p = dropout_p
        self.num_emb = num_emb

        with self.name_scope():
            self.user_embeddings = gluon.nn.Embedding(max_users, num_emb)
            self.item_embeddings = gluon.nn.Embedding(max_items, num_emb)
            self.dropout = gluon.nn.Dropout(dropout_p)

    def forward(self, users, items):
        a = self.user_embeddings(users)
        b = self.item_embeddings(items)
        predictions = self.dropout(a) * self.dropout(b)
        predictions = nd.sum(predictions, axis=1)
        return predictions
```

```
In [9]: net = MFBlock(max_users=max_users, max_items=max_items, num_emb=num_emb, dropout_p=0.)
net.collect_params()
```



32:17 / 54:31



```
In [11]: net.collect_params().initialize(mx.init.Xavier(magnitude=2.24), ctx=ctx, force_reinit=True)
```

```
In [12]: trainer = gluon.Trainer(net.collect_params(), 'sgd',
                             {'learning_rate': lr, 'wd': wd, 'momentum': 0.9})
```

```
In [13]: train_data_iter = gluon.data.DataLoader(SparseMatrixDataset(train_data, train_label),
                                             shuffle=True, batch_size=batch_size)
test_data_iter = gluon.data.DataLoader(SparseMatrixDataset(test_data, test_label),
                                       shuffle=True, batch_size=batch_size)
```

```
In [14]: def eval_net(data, net):
    acc = mx.metric.RMSE()
    for i, (user, item, label) in enumerate(data):
        user = user.as_in_context(ctx).reshape((batch_size,))
        item = item.as_in_context(ctx).reshape((batch_size,))
        label = label.as_in_context(ctx).reshape((batch_size,))
        predictions = net(user, item)
        loss = loss_function(predictions, label)
        acc.update(preds=predictions, labels=label)
    return acc.get()[1]
```

```
In [15]: eval_net(test_data_iter, net)
```

```
Out[15]: 3.5358702216744424
```

```
In [16]: epochs = 10
#smoothing_constant = 10

def train(data_iter, net):
    a = []
    b = []
    c = []
    d = []
    for e in range(epochs):
        print("epoch: {}".format(e))
        for i, (user, item, label) in enumerate(train_data_iter):
            user = user.as_in_context(ctx).reshape((batch_size,))
```



35:47 / 54:31



```
    loss = loss_function(predictions, label)
    acc.update(preds=predictions, labels=label)
    return acc.get()[1]
```

```
In [15]: eval_net(test_data_iter, net)
```

```
Out[15]: 3.5358702216744424
```

```
In [16]: epochs = 10
#smoothing_constant = 10

def train(data_iter, net):
    a = []
    b = []
    c = []
    d = []
    for e in range(epochs):
        print("epoch: {}".format(e))
        for i, (user, item, label) in enumerate(train_data_iter):
            user = user.as_in_context(ctx).reshape((batch_size,))
            item = item.as_in_context(ctx).reshape((batch_size,))
            label = label.as_in_context(ctx).reshape((batch_size,))
            with mx.autograd.record():
                output = net(user, item)
                loss = loss_function(output, label)
                loss.backward()
            net.collect_params().values()
            trainer.step(batch_size)
            a = eval_net(test_data_iter, net)
            b = eval_net(train_data_iter, net)
            print("EPOCH {}: RMSE ON TRAINING and TEST: {}, {}".format(e,a,b))

    return a, b
```

```
In [17]: (a,b) = train(train_data_iter, net)
```

```
epoch: 0
EPOCH 0: RMSE ON TRAINING and TEST: 3.533845988896489. 3.523215442742407
epoch: 1
EPOCH 1: RMSE ON TRAINING and TEST: 3.398607304608822. 3.328483776437491
```



38:04 / 54:31



```
        loss.backward()
    net.collect_params().values()
    trainer.step(batch_size)
    a = eval_net(test_data_iter, net)
    b = eval_net(train_data_iter, net)
    print("EPOCH {}: RMSE ON TRAINING and TEST: {}.\n {}".format(e,a,b))

    return a, b
```

```
In [17]: (a,b) = train(train_data_iter, net)
```

```
epoch: 0
EPOCH 0: RMSE ON TRAINING and TEST: 3.533845988896489. 3.523215442742407
epoch: 1
EPOCH 1: RMSE ON TRAINING and TEST: 3.398607304608822. 3.328483776437491
epoch: 2
EPOCH 2: RMSE ON TRAINING and TEST: 2.032065240380168. 1.797713072796911
epoch: 3
EPOCH 3: RMSE ON TRAINING and TEST: 1.3171076374143362. 1.1745245898365975
epoch: 4
EPOCH 4: RMSE ON TRAINING and TEST: 1.0678859624534844. 0.9613828420139849
epoch: 5
EPOCH 5: RMSE ON TRAINING and TEST: 0.9528403462797403. 0.866311743491143
epoch: 6
EPOCH 6: RMSE ON TRAINING and TEST: 0.8905037337005138. 0.8156895110182464
epoch: 7
EPOCH 7: RMSE ON TRAINING and TEST: 0.8528216697186232. 0.785909748146683
epoch: 8
EPOCH 8: RMSE ON TRAINING and TEST: 0.8289129304856062. 0.7666749547488988
epoch: 9
EPOCH 9: RMSE ON TRAINING and TEST: 0.8108262846082449. 0.7518034620203078
```

```
In [18]: (a,b)
```

```
Out[18]: (0.8108262846082449, 0.7518034620203078)
```

```
In [ ]:
```



39:22 / 54:31



```
In [7]: class MFBlock(gluon.Block):
    def __init__(self, max_users, max_items, num_emb, dropout_p=0.5):
        super(MFBlock, self).__init__()

        self.max_users = max_users
        self.max_items = max_items
        self.dropout_p = dropout_p
        self.num_emb = num_emb

        with self.name_scope():
            self.user_embeddings = gluon.nn.Embedding(max_users, num_emb)
            self.item_embeddings = gluon.nn.Embedding(max_items, num_emb)
            self.dropout = gluon.nn.Dropout(dropout_p)
            self.dense = gluon.nn.Dense(num_emb, activation='relu')

    def forward(self, users, items):
        a = self.user_embeddings(users)
        a = self.dense(a)

        b = self.item_embeddings(items)
        b = self.dense(b)

        predictions = self.dropout(a) * self.dropout(b)
        predictions = nd.sum(predictions, axis=1)
        return predictions
```

```
In [8]: net = MFBlock(max_users=max_users, max_items=max_items, num_emb=num_emb, dropout_p=0.)
net.collect_params()
```

```
Out[8]: mfblock0_
Parameter mfblock0_embedding0_weight (shape=(944, 64), dtype=<class 'numpy.float32'>)
Parameter mfblock0_embedding1_weight (shape=(1683, 64), dtype=<class 'numpy.float32'>)
Parameter mfblock0_dense0_weight (shape=(64, 0), dtype=<class 'numpy.float32'>)
```

```
        loss = loss_function(output, label)
        loss.backward()
        net.collect_params().values()
        trainer.step(batch_size)
        print("EPOCH {}: RMSE ON TRAINING and TEST: {}.".format(e,
                                                                eval_net(train_data_iter, net),
                                                                eval_net(test_data_iter, net)))
    return "end of training"
```

In [16]: train(train_data_iter, net)

```
epoch: 0
EPOCH 0: RMSE ON TRAINING and TEST: 0.7461072485804557. 0.7763755543172359
epoch: 1
EPOCH 1: RMSE ON TRAINING and TEST: 0.7369058181449771. 0.7680148655653
epoch: 2
EPOCH 2: RMSE ON TRAINING and TEST: 0.7472432709142566. 0.7772404563993216
epoch: 3
EPOCH 3: RMSE ON TRAINING and TEST: 0.7370284162349999. 0.7691778198421001
epoch: 4
EPOCH 4: RMSE ON TRAINING and TEST: 0.7406699358060956. 0.7754190125107765
epoch: 5
EPOCH 5: RMSE ON TRAINING and TEST: 0.7273183228254319. 0.7669575016319752
epoch: 6
EPOCH 6: RMSE ON TRAINING and TEST: 0.7240261309757828. 0.7765023551046848
epoch: 7
EPOCH 7: RMSE ON TRAINING and TEST: 0.693246350326389. 0.7645233050823211
epoch: 8
EPOCH 8: RMSE ON TRAINING and TEST: 0.6648808738991618. 0.7582760076761246
epoch: 9
EPOCH 9: RMSE ON TRAINING and TEST: 0.6372081472031772. 0.7497557436436415
```

Out[16]: 'end of training'

In [17]: net1 = gluon.nn.Sequential()
with net1.name_scope():
 net1.add(gluon.nn.Embedding(max_users, num_emb))
 net1.add(gluon.nn.Dense(64))

```
from scipy.sparse import lil_matrix

BUCKET = 'cyrusmv-sagemaker-demos'
s3 = boto3.client('s3')


def download_file(s3_source, dest):
    if not os.path.exists(dest):
        os.makedirs(dest)

    url = urlparse(s3_source)
    bucket, key = url.netloc, url.path.lstrip('/')
    file_name = key.split('/')[-1]
    with open('%s/%s' % (dest, file_name), 'wb') as data:
        s3.download_fileobj(bucket, key, data)


def loadDataset(filename, lines, columns):
    # Features are one-hot encoded in a sparse matrix
    X = lil_matrix((lines, columns)).astype('float32')
    # Labels are stored in a vector
    Y = []
    line=0
    with open(filename, 'r') as f:
        samples=csv.reader(f,delimiter='\t')
        for userId,movieId,rating,timestamp in samples:
            X[line,int(userId)-1] = 1
            X[line,int(nbUsers)+int(movieId)-1] = 1
            Y.append(int(rating))
            line=line+1

    Y=np.array(Y).astype('float32')
    return X,Y

nbUsers=943
nbMovies=1682
```



41:34 / 54:31



```
<_io.BytesIO object at 0x7f9ac0385f50>
Wrote dataset: cyrusmv-sagemaker-demos/exercise4/fm-movielens100k/train/train.protobuf
<_io.BytesIO object at 0x7f9ac0343c50>
Wrote dataset: cyrusmv-sagemaker-demos/exercise4/fm-movielens100k/test/test.protobuf
Output: s3://cyrusmv-sagemaker-demos/exercise4/fm-movielens100k/output
```

```
In [3]: import sagemaker
from sagemaker import get_execution_role

train_data = 's3://%s/exercise4/fm-movielens100k/train/train.protobuf' % BUCKET
test_data = 's3://%s/exercise4/fm-movielens100k/test/test.protobuf' % BUCKET

containers = {'us-west-2': '174872318107.dkr.ecr.us-west-2.amazonaws.com/factorization-machines:latest',
              'us-east-1': '382416733822.dkr.ecr.us-east-1.amazonaws.com/factorization-machines:latest',
              'us-east-2': '404615174143.dkr.ecr.us-east-2.amazonaws.com/factorization-machines:latest',
              'eu-west-1': '438346466558.dkr.ecr.eu-west-1.amazonaws.com/factorization-machines:latest'}

fm = sagemaker.estimator.Estimator(containers[boto3.Session().region_name],
                                    get_execution_role(),
                                    train_instance_count=1,
                                    train_instance_type='ml.c4.xlarge',
                                    output_path=output_prefix,
                                    sagemaker_session=sagemaker.Session())

fm.set_hyperparameters(feature_dim=nbFeatures,
                       predictor_type='regressor',
                       mini_batch_size=1000,
                       num_factors=64,
                       _speedometer_period=10,
                       epochs=50)

fm.fit({'train': train_data, 'test': test_data})
# <AmazonSageMakerEstimator>: {"Training": {"TrainingJobName": "fm-movielens100k", "TrainingJobStatus": "Completed", "AlgorithmSpecification": {"TrainingImage": "174872318107.dkr.ecr.us-west-2.amazonaws.com/factorization-machines:latest", "TrainingInputMode": "File"}, "HyperParameters": {"feature_dim": "120", "predictor_type": "regressor", "mini_batch_size": "1000", "num_factors": "64", "epochs": "50", "speedometer_period": "10"}, "ResourceConfig": {"InstanceType": "ml.c4.xlarge", "VolumeKmsKeyId": null, "InstanceCount": 1}, "OutputConfig": {"S3OutputPath": "s3://cyrusmv-sagemaker-demos/exercise4/fm-movielens100k/output"}, "RoleArn": "arn:aws:iam::123456789012:role/SageMaker-Execution-Role", "TrainingDurationInSeconds": 10800, "CompletionTime": "2018-05-15T13:45:15.624Z", "LastModifiedTime": "2018-05-15T13:45:15.624Z", "AlgorithmArn": "arn:aws:sagemaker:us-west-2:123456789012:algorithm/fm-movielens100k"}, "Testing": {"TestJobName": "fm-movielens100k-test", "TestJobStatus": "Completed", "AlgorithmSpecification": {"TrainingImage": "174872318107.dkr.ecr.us-west-2.amazonaws.com/factorization-machines:latest", "TrainingInputMode": "File"}, "HyperParameters": {"feature_dim": "120", "predictor_type": "regressor", "mini_batch_size": "1000", "num_factors": "64", "epochs": "50", "speedometer_period": "10"}, "ResourceConfig": {"InstanceType": "ml.c4.xlarge", "VolumeKmsKeyId": null, "InstanceCount": 1}, "OutputConfig": {"S3OutputPath": "s3://cyrusmv-sagemaker-demos/exercise4/fm-movielens100k/output"}, "RoleArn": "arn:aws:iam::123456789012:role/SageMaker-Execution-Role", "TestingDurationInSeconds": 10800, "CompletionTime": "2018-05-15T13:45:15.624Z", "LastModifiedTime": "2018-05-15T13:45:15.624Z", "AlgorithmArn": "arn:aws:sagemaker:us-west-2:123456789012:algorithm/fm-movielens100k"}, "Deployment": null}
```

```
_speedometer_period=10,  
epochs=50)  
  
fm.fit({'train': train_data, 'test': test_data})  
  
[03/21/2018 23:46:18 INFO 139926754449216] Epoch[1] Batch [70]#011Speed: 152590.06 samples/sec#011rmse=0.963353  
#metrics {"Metrics": {"update.time": {"count": 1, "max": 500.90694427490234, "sum": 500.90694427490234, "min": 500.90694427490234}, "EndTime": 1521675978.700887, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "factorization-machines"}, "StartTime": 1521675978.19969}  
  
#metrics {"Metrics": {"Max Batches Seen Between Resets": {"count": 1, "max": 80, "sum": 80.0, "min": 80}, "Number of Batches Since Last Reset": {"count": 1, "max": 80, "sum": 80.0, "min": 80}, "Number of Records Since Last Reset": {"count": 1, "max": 80000, "sum": 80000.0, "min": 80000}, "Total Batches Seen": {"count": 1, "max": 3201, "sum": 3201.0, "min": 3201}, "Total Records Seen": {"count": 1, "max": 3201000, "sum": 3201000.0, "min": 3201000}, "Max Records Seen Between Resets": {"count": 1, "max": 80000, "sum": 80000.0, "min": 80000}, "Reset Count": {"count": 1, "max": 41, "sum": 41.0, "min": 41}}, "EndTime": 1521675978.701077, "Dimensions": {"Host": "algo-1", "Meta": "training_data_iter", "Operation": "training", "Algorithm": "factorization-machines", "epoch": 39}, "StartTime": 1521675978.701025}  
  
[03/21/2018 23:46:18 INFO 139926754449216] Epoch[1] Batch [10]#011Speed: 144500.13 samples/sec#011rmse=0.999548  
[03/21/2018 23:46:18 INFO 139926754449216] Epoch[1] Batch [20]#011Speed: 158260.69 samples/sec#011rmse=0.974219  
[03/21/2018 23:46:18 INFO 139926754449216] Epoch[1] Batch [30]#011Speed: 146147.58 samples/sec#011rmse=0.982746  
[03/21/2018 23:46:18 INFO 139926754449216] Epoch[1] Batch [40]#011Speed: 144542.45 samples/sec#011rmse=0.973204  
[03/21/2018 23:46:19 INFO 139926754449216] Epoch[1] Batch [50]#011Speed: 150276.38 samples/sec#011rmse=0.971157  
[03/21/2018 23:46:19 INFO 139926754449216] Epoch[1] Batch [60]#011Speed: 148270.97 samples/sec#011rmse=0.964169
```

In [4]: fm_predictor = fm.deploy(instance_type='ml.c4.xlarge', initial_instance_count=1)

```
INFO:sagemaker:Creating model with name: factorization-machines-2018-03-21-23-47-57-115  
INFO:sagemaker:Creating endpoint with name factorization-machines-2018-03-21-23-40-14-697
```

In [5]: import json
import numpy as np
from sagemaker.predictor import json_deserializer

```
_speedometer_period=10,  
epochs=50)  
  
fm.fit({'train': train_data, 'test': test_data})  
  
[03/21/2018 23:46:23 INFO 139926754449216] Saved checkpoint to "/tmp/tmpTlZMG/_state-0001.params"  
[03/21/2018 23:46:23 INFO 139926754449216] #test_score (algo-1) : rmse  
[03/21/2018 23:46:23 INFO 139926754449216] #test_score (algo-1) : 1.00061402047  
#metrics {"Metrics": {"Max Batches Seen Between Resets": {"count": 1, "max": 20, "sum": 20.0, "min": 20}, "Number of Batches Since Last Reset": {"count": 1, "max": 20, "sum": 20.0, "min": 20}, "Number of Records Since Last Reset": {"count": 1, "max": 20000, "sum": 20000.0, "min": 20000}, "Total Batches Seen": {"count": 1, "max": 20, "sum": 20.0, "min": 20}, "Total Records Seen": {"count": 1, "max": 20000, "sum": 20000.0, "min": 20000}, "Max Records Seen Between Resets": {"count": 1, "max": 20000, "sum": 20000.0, "min": 20000}, "Reset Count": {"count": 1, "max": 1, "sum": 1.0, "min": 1}}, "EndTime": 1521675983.980208, "Dimensions": {"Host": "algo-1", "Meta": "test_data_iter", "Operation": "training", "Algorithm": "factorization-machines"}, "StartTime": 1521675983.980173}  
  
#metrics {"Metrics": {"totaltime": {"count": 1, "max": 26234.050989151, "sum": 26234.050989151, "min": 26234.050989151}, "setuptime": {"count": 1, "max": 40.194034576416016, "sum": 40.194034576416016, "min": 40.194034576416016}, "EndTime": 1521675983.981656, "Dimensions": {"Host": "algo-1", "Operation": "training", "Algorithm": "factorization-machines"}, "StartTime": 1521675983.902766}}  
  
===== Job Complete =====  
Billable seconds: 217
```

```
In [4]: fm_predictor = fm.deploy(instance_type='ml.c4.xlarge', initial_instance_count=1)
```

```
INFO:sagemaker:Creating model with name: factorization-machines-2018-03-21-23-47-57-115  
INFO:sagemaker:Creating endpoint with name factorization-machines-2018-03-21-23-40-14-697
```

```
-----!  
  
In [5]: import json  
import numpy as np  
from sagemaker.predictor import json_deserializer
```

```
In [5]: import json
import numpy as np
from sagemaker.predictor import json_deserializer
```

```
nbUsers=943
nbMovies=1682
nbFeatures=nbUsers+nbMovies
```

```
def fm_serializer(data):
    js = {'instances': []}
    for row in data:
        keys = np.argwhere(row == np.amax(row)).flatten().tolist()
        js['instances'].append({
            'data':{
                'features': {
                    'keys': keys,
                    'shape': [nbFeatures],
                    'values': [1]*len(keys)
                }
            }
        })
    #print js
    return json.dumps(js)
```

```
fm_predictor.content_type = 'application/json'
fm_predictor.serializer = fm_serializer
fm_predictor.deserializer = json_deserializer
```

```
result = fm_predictor.predict(X_test[1000:1010].toarray())
print(result)
print()
print(X_test[1000:1010])
```

```
[{"label": 1, "probability": 0.3230037620034623}, {"label": 0, "probability": 0.6769962379965377}, {"label": 1, "probability": 0.225492639541626}, {"label": 0,
```

```
def fm_serializer(data):
    js = {'instances': []}
    for row in data:
        keys = np.argwhere(row == np.amax(row)).flatten().tolist()
        js['instances'].append({
            'data':{
                'features': {
                    'keys': keys,
                    'shape': [nbFeatures],
                    'values': [1]*len(keys)
                }
            }
        })
    #print js
    return json.dumps(js)

fm_predictor.content_type = 'application/json'
fm_predictor.serializer = fm_serializer
fm_predictor.deserializer = json_deserializer

result = fm_predictor.predict(X_test[1000:1010].toarray())
print(result)
print()
print(Y_test[1000:1010])

{u'predictions': [{u'score': 3.3320837020874023}, {u'score': 3.0627427101135254}, {u'score': 3.305492639541626}, {u'score': 2.9380016326904297}, {u'score': 2.8458235263824463}, {u'score': 3.073624849319458}, {u'score': 3.040721893310547}, {u'score': 3.3230855464935303}, {u'score': 3.044969081878662}, {u'score': 3.535712480545044}]}
()
[2. 1. 3. 3. 1. 3. 3. 1. 4.]
```

Amazon SageMaker

Developer Guide

Documentation - This Guide

Search

[What Is Amazon SageMaker?](#)[How It Works](#)[Getting Started](#)[Using Built-in Algorithms](#)[Common Information](#)[Linear Learner](#)[Factorization Machines](#)[How It Works](#)[Hyperparameters](#)[Inference Formats](#)[XGBoost Algorithm](#)[Image Classification Algorithm](#)[Sequence to Sequence \(seq2seq\)](#)[K-Means Algorithm](#)[Principal Component Analysis \(PCA\)](#)[Latent Dirichlet Allocation \(LDA\)](#)[Neural Topic Model \(NTM\)](#)[DeepAR Forecasting](#)[BlazingText](#)[Random Cut Forest](#)[Using Your Own Algorithms](#)

Factorization Machines Hyperparameters

Parameter Name	Description
feature_dim	Dimension of the input feature space. This could be very high with sparse input. Required. Valid values: Positive integer. Suggested value range: [10000,10000000] Default value: -
num_factors	Dimensionality of factorization. Required. Valid values: Positive integer. Suggested value range: [2,1000] Default value: -
predictor_type	Type of predictor. Required. Valid values: String: <code>binary_classifier</code> or <code>regressor</code> Default value: -
mini_batch_size	Size of mini-batch used for training. Valid values: positive integer Default value: 1000
epochs	Number of training epochs to run. Valid values: positive integer Default value: 1
clip_gradient	Optimizer parameter. Clip the gradient by projecting onto the box [-clip_gradient, +clip_gradient]. Valid values: float Default value: -
eps	Optimizer parameter. Small value to avoid division by 0. Valid values: float



45:10 / 54:31



Amazon SageMaker



Developer Guide

Documentation - This Guide

Search

[What Is Amazon SageMaker?](#)[How It Works](#)[Getting Started](#)[Using Built-in Algorithms](#)[Common Information](#)[Linear Learner](#)[Factorization Machines](#)[How It Works](#)[Hyperparameters](#)[Inference Formats](#)[XGBoost Algorithm](#)[Image Classification Algorithm](#)[Sequence to Sequence \(seq2seq\)](#)[K-Means Algorithm](#)[Principal Component Analysis \(PCA\)](#)[Latent Dirichlet Allocation \(LDA\)](#)[Neural Topic Model \(NTM\)](#)[DeepAR Forecasting](#)[BlazingText](#)[Random Cut Forest](#)[Using Your Own Algorithms](#)

predictor_type	Type of predictor. Required. Valid values: String: <code>binary_classifier</code> or <code>regressor</code> Default value: -
mini_batch_size	Size of mini-batch used for training. Valid values: positive integer Default value: 1000
epochs	Number of training epochs to run. Valid values: positive integer Default value: 1
clip_gradient	Optimizer parameter. Clip the gradient by projecting onto the box [-clip_gradient, +clip_gradient]. Valid values: float Default value: -
eps	Optimizer parameter. Small value to avoid division by 0. Valid values: float Default value: -
rescale_grad	Optimizer parameter. If set, multiplies the gradient with rescale_grad before updating. Often choose to be 1.0/batch_size. Valid values: float Default value: -
bias_lr	Learning rate for the bias term. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.1
linear_lr	Learning rate for linear terms. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.001

Amazon SageMaker



Developer Guide

Documentation - This Guide



Search

[What Is Amazon SageMaker?](#)[How It Works](#)[Getting Started](#)[Using Built-in Algorithms](#)[Common Information](#)[Linear Learner](#)[Factorization Machines](#)[How It Works](#)[Hyperparameters](#)[Inference Formats](#)[XGBoost Algorithm](#)[Image Classification Algorithm](#)[Sequence to Sequence \(seq2seq\)](#)[K-Means Algorithm](#)[Principal Component Analysis \(PCA\)](#)[Latent Dirichlet Allocation \(LDA\)](#)[Neural Topic Model \(NTM\)](#)[DeepAR Forecasting](#)[BlazingText](#)[Random Cut Forest](#)[Using Your Own Algorithms](#)

		Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.1
	linear_lr	Learning rate for linear terms. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.001
	factors_lr	Learning rate for factorization terms. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.0001
	bias_wd	Weight decay for the bias term. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01
	linear_wd	Weight decay for linear terms. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.001
	factors_wd	Weight decay for factorization terms. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.00001
	bias_init_method	Initialization method for the bias term. <ul style="list-style-type: none">• <i>normal</i>: initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>bias_init_sigma</code>.• <i>uniform</i>: initializes weights with random values uniformly sampled from a range specified by <code>[bias_init_scale, +bias_init_scale]</code>.• <i>constant</i>: initializes the weights to a scalar value specified by <code>bias_init_value</code>. Valid values: <i>uniform</i> , <i>normal</i> , or <i>constant</i>
		Default value: <i>normal</i>
	bias_init_scale	Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <i>uniform</i> .

Amazon SageMaker



Developer Guide

Documentation - This Guide



Search

[What Is Amazon SageMaker?](#)[How It Works](#)[Getting Started](#)[Using Built-in Algorithms](#)[Common Information](#)[Linear Learner](#)[Factorization Machines](#)[How It Works](#)[Hyperparameters](#)[Inference Formats](#)[XGBoost Algorithm](#)[Image Classification Algorithm](#)[Sequence to Sequence \(seq2seq\)](#)[K-Means Algorithm](#)[Principal Component Analysis \(PCA\)](#)[Latent Dirichlet Allocation \(LDA\)](#)[Neural Topic Model \(NTM\)](#)[DeepAR Forecasting](#)[BlazingText](#)[Random Cut Forest](#)[Using Your Own Algorithms](#)

		Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01
	linear_vd	Weight decay for linear terms. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.001
	factors_vd	Weight decay for factorization terms. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.00001
	bias_init_method	Initialization method for the bias term. <ul style="list-style-type: none">• <i>normal</i>: initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>bias_init_sigma</code>.• <i>uniform</i>: initializes weights with random values uniformly sampled from a range specified by <code>-bias_init_scale</code>, <code>+bias_init_scale</code>.• <i>constant</i>: initializes the weights to a scalar value specified by <code>bias_init_value</code>. Valid values: <i>uniform</i> , <i>normal</i> , or <i>constant</i> Default value: <i>normal</i>
	bias_init_scale	Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <i>uniform</i> . Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: -
	bias_init_sigma	Standard deviation for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <i>normal</i> . Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01
	bias_init_value	Initial value of the bias term. Takes effect if <code>bias_init_method</code> is set to <i>constant</i> . Valid values: Float. Suggested value range: [1e-8, 512]. Default value: -
	linear_init_method	Initialization method for linear terms.

Amazon SageMaker



Developer Guide

Documentation - This Guide



Search

[What Is Amazon SageMaker?](#)[How It Works](#)[Getting Started](#)[Using Built-in Algorithms](#)[Common Information](#)[Linear Learner](#)[Factorization Machines](#)[How It Works](#)[Hyperparameters](#)[Inference Formats](#)[XGBoost Algorithm](#)[Image Classification Algorithm](#)[Sequence to Sequence \(seq2seq\)](#)[K-Means Algorithm](#)[Principal Component Analysis \(PCA\)](#)[Latent Dirichlet Allocation \(LDA\)](#)[Neural Topic Model \(NTM\)](#)[DeepAR Forecasting](#)[BlazingText](#)[Random Cut Forest](#)[Using Your Own Algorithms](#)

bias_init_method	Initialization method for the bias term. <ul style="list-style-type: none"><i>normal</i>: initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>bias_init_sigma</code>.<i>uniform</i>: initializes weights with random values uniformly sampled from a range specified by <code>[-bias_init_scale, +bias_init_scale]</code>.<i>constant</i>: initializes the weights to a scalar value specified by <code>bias_init_value</code>. Valid values: <i>uniform</i> , <i>normal</i> , or <i>constant</i> .Default value: <i>normal</i> .Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <i>uniform</i> .Valid values: Non-negative float. Suggested value range: [1e-8, 512].Default value: -.Standard deviation for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <i>normal</i> .Valid values: Non-negative float. Suggested value range: [1e-8, 512].Default value: 0.01.Initial value of the bias term. Takes effect if <code>bias_init_method</code> is set to <i>constant</i> .Valid values: Float. Suggested value range: [1e-8, 512].Default value: -.Initialization method for linear terms. <ul style="list-style-type: none"><i>normal</i>: initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>linear_init_sigma</code>.<i>uniform</i>: initializes weights with random values uniformly sampled from a range specified by <code>[-linear_init_scale, +linear_init_scale]</code>.<i>constant</i>: initializes the weights to a scalar value specified by <code>linear_init_value</code>. Valid values: <i>uniform</i> , <i>normal</i> , or <i>constant</i> .Default value: <i>normal</i> .Range for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <i>uniform</i> .Valid values: Non-negative float. Suggested value range: [1e-8, 512].Default value: .
------------------	---

Amazon SageMaker



Developer Guide

Documentation - This Guide

Search

[What Is Amazon SageMaker?](#)[How It Works](#)[Getting Started](#)[Using Built-in Algorithms](#)[Common Information](#)[Linear Learner](#)[Factorization Machines](#)[How It Works](#)[Hyperparameters](#)[Inference Formats](#)[XGBoost Algorithm](#)[Image Classification Algorithm](#)[Sequence to Sequence \(seq2seq\)](#)[K-Means Algorithm](#)[Principal Component Analysis \(PCA\)](#)[Latent Dirichlet Allocation \(LDA\)](#)[Neural Topic Model \(NTM\)](#)[DeepAR Forecasting](#)[BlazingText](#)[Random Cut Forest](#)[Using Your Own Algorithms](#)

	<ul style="list-style-type: none">constant: initializes the weights to a scalar value specified by <code>bias_init_value</code>. <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code></p> <p>Default value: <code>normal</code></p>
<code>bias_init_scale</code>	Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>uniform</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: -</p>
<code>bias_init_sigma</code>	Standard deviation for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>normal</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>bias_init_value</code>	Initial value of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>constant</code> . <p>Valid values: Float. Suggested value range: [1e-8, \$12]/</p> <p>Default value: -</p>
<code>linear_init_method</code>	Initialization method for linear terms. <ul style="list-style-type: none"><code>normal</code>: initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>linear_init_sigma</code>.<code>uniform</code>: initializes weights with random values uniformly sampled from a range specified by [-<code>linear_init_scale</code>, +<code>linear_init_scale</code>].<code>constant</code>: initializes the weights to a scalar value specified by <code>linear_init_value</code>. <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>
<code>linear_init_scale</code>	Range for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>uniform</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: -</p>
<code>linear_init_sigma</code>	Standard deviation for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>normal</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>

Amazon SageMaker



Developer Guide

Documentation - This Guide



Search

[What Is Amazon SageMaker?](#)[How It Works](#)[Getting Started](#)[Using Built-in Algorithms](#)[Common Information](#)[Linear Learner](#)[Factorization Machines](#)[How It Works](#)[Hyperparameters](#)[Inference Formats](#)[XGBoost Algorithm](#)[Image Classification Algorithm](#)[Sequence to Sequence \(seq2seq\)](#)[K-Means Algorithm](#)[Principal Component Analysis \(PCA\)](#)[Latent Dirichlet Allocation \(LDA\)](#)[Neural Topic Model \(NTM\)](#)[DeepAR Forecasting](#)[BlazingText](#)[Random Cut Forest](#)[Using Your Own Algorithms](#)

	<ul style="list-style-type: none">constant: initializes the weights to a scalar value specified by <code>bias_init_value</code>. <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code></p> <p>Default value: <code>normal</code></p>
<code>bias_init_scale</code>	Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>uniform</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: -</p>
<code>bias_init_sigma</code>	Standard deviation for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>normal</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>bias_init_value</code>	Initial value of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>constant</code> . <p>Valid values: Float. Suggested value range: [1e-8, \$12]/</p> <p>Default value: -</p>
<code>linear_init_method</code>	Initialization method for linear terms. <ul style="list-style-type: none"><code>normal</code>: initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>linear_init_sigma</code>.<code>uniform</code>: initializes weights with random values uniformly sampled from a range specified by [-<code>linear_init_scale</code>, +<code>linear_init_scale</code>].<code>constant</code>: initializes the weights to a scalar value specified by <code>linear_init_value</code>. <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>
<code>linear_init_scale</code>	Range for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>uniform</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: -</p>
<code>linear_init_sigma</code>	Standard deviation for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>normal</code> . <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>

Development and Deployment



- Loss function is one of the most important areas to pay attention to.
- Multi-label cross-entropy loss has worked well in the past.
- This is relatively simple to apply to a wide variety of model types. Ranking loss often works better, but is more complex to apply correctly.
- Scalability is always a big challenge. Offline batch computation and saving the results can help.

Logging and Measurement



- Deploying a recommender system requires some care since a model only succeeds if good behavioral data can be logged.
- Moreover, without good logging it is impossible to assess the quality of the deployment. Tools such as Amazon Kinesis are ideally suited for this purpose.
- Display bias is very strong – this means that customers are more likely to click on a mediocre recommendation that they see than an excellent recommendation they don't see.

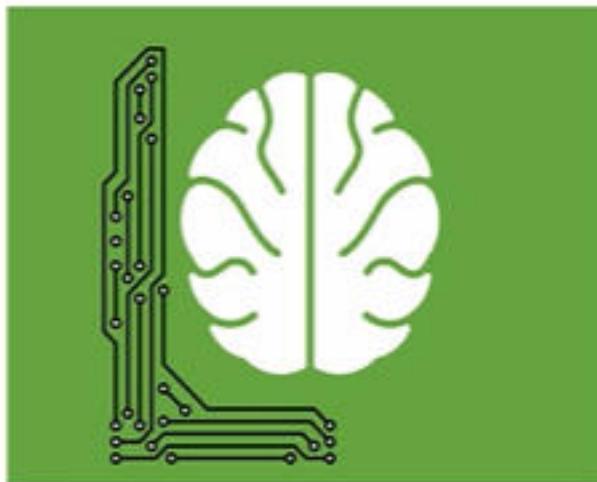
What is Data Science?



- **General Definition:** Processes and systems to extract knowledge or insights from data, either structured or unstructured. (*Wikipedia*)
- **For the purposes of this course:** Managing, analyzing, and visualizing data in support of the Machine Learning workflow.
- But what is Machine Learning?

What is Machine Learning?

Artificial Intelligence machines that improve their predictions by learning from large amounts of input data.



Machine Learning



- **Main idea:** Learning = estimating underlying function f by mapping data attributes to some target value
- **Training set:** A set of labeled examples $(x, f(x))$ where x is the input variables and the label $f(x)$ is the observed target truth
- **Goal:** Given a training set, find approximation \hat{f} of f that best generalizes, or predicts, labels for new examples
 - “Best” is measured by some quality measure
 - **Example:** error rate, sum squared error

Machine Learning



- **Main idea:** Learning = estimating underlying function f by mapping data attributes to some target value
- **Training set:** A set of labeled examples $(x, f(x))$ where x is the input variables and the label $f(x)$ is the observed target truth
- **Goal:** Given a training set, find approximation \hat{f} of f that best generalizes, or predicts, labels for new examples
 - "Best" is measured by some quality measure
 - Example: error rate, sum squared error



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



-10:32

2x

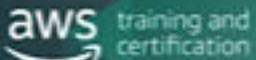


Machine Learning



- **Main idea:** Learning = estimating underlying function f by mapping data attributes to some target value
- **Training set:** A set of labeled examples $(x, f(x))$ where x is the input variables and the label $f(x)$ is the observed target truth
- **Goal:** Given a training set, find approximation \hat{f} of f that best generalizes, or predicts, labels for new examples
 - “Best” is measured by some quality measure
 - **Example:** error rate, sum squared error

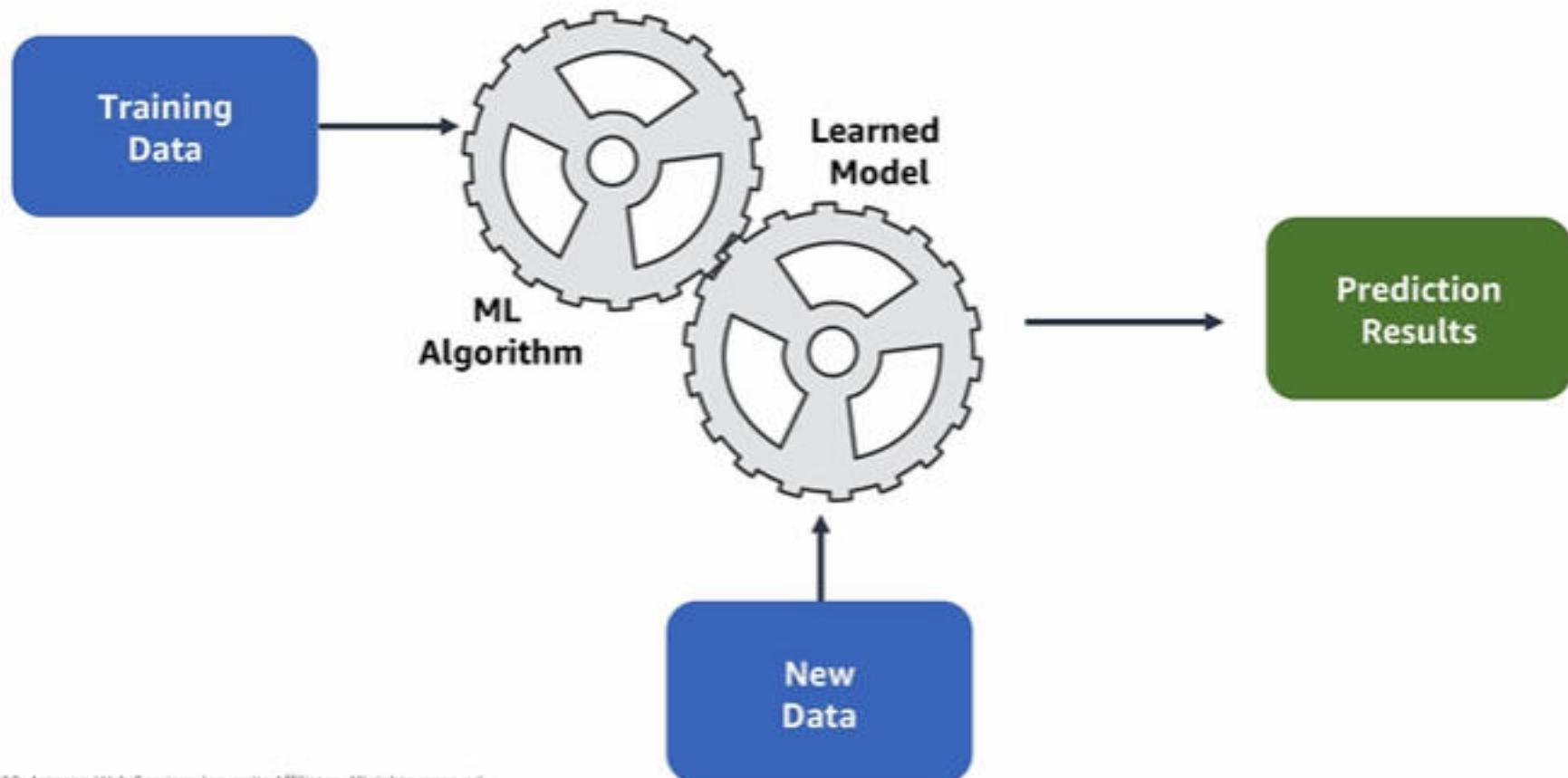
Machine Learning



- **Main idea:** Learning = estimating underlying function f by mapping data attributes to some target value
- **Training set:** A set of labeled examples $(x, f(x))$ where x is the input variables and the label $f(x)$ is the observed target truth
- **Goal:** Given a training set, find approximation \hat{f} of f that best generalizes, or predicts, labels for new examples
 - "Best" is measured by some quality measure
 - **Example:** error rate, sum squared error



Machine Learning



Why Machine Learning?



Difficulty in writing some programs

- Too complex (facial recognition)
- Too much data (stock market predictions)
- Information only available dynamically (recommendation system)

Use of data for improvement

- Humans are used to improving based on experience (data)

A lot of data is available

- Product recommendations
- Fraud detection
- Facial recognition
- Language understanding
-

Why Machine Learning?



Difficulty in writing some programs

- Too complex (facial recognition)
- Too much data (stock market predictions)
- Information only available dynamically (recommendation system)

Use of data for improvement

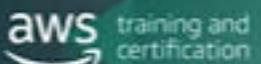
- Humans are used to improving based on experience (data)

A lot of data is available

- Product recommendations
- Fraud detection
- Facial recognition
- Language understanding
- ...



Types of Machine Learning



Supervised Learning



Semi-supervised Learning



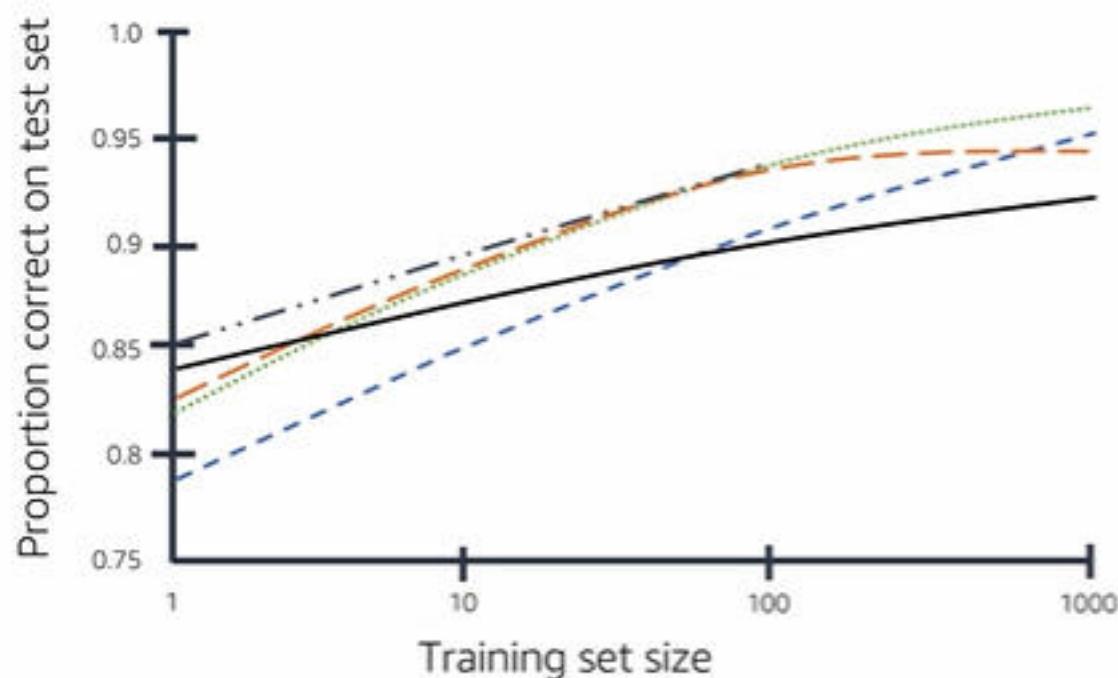
Reinforcement Learning



Unsupervised Learning



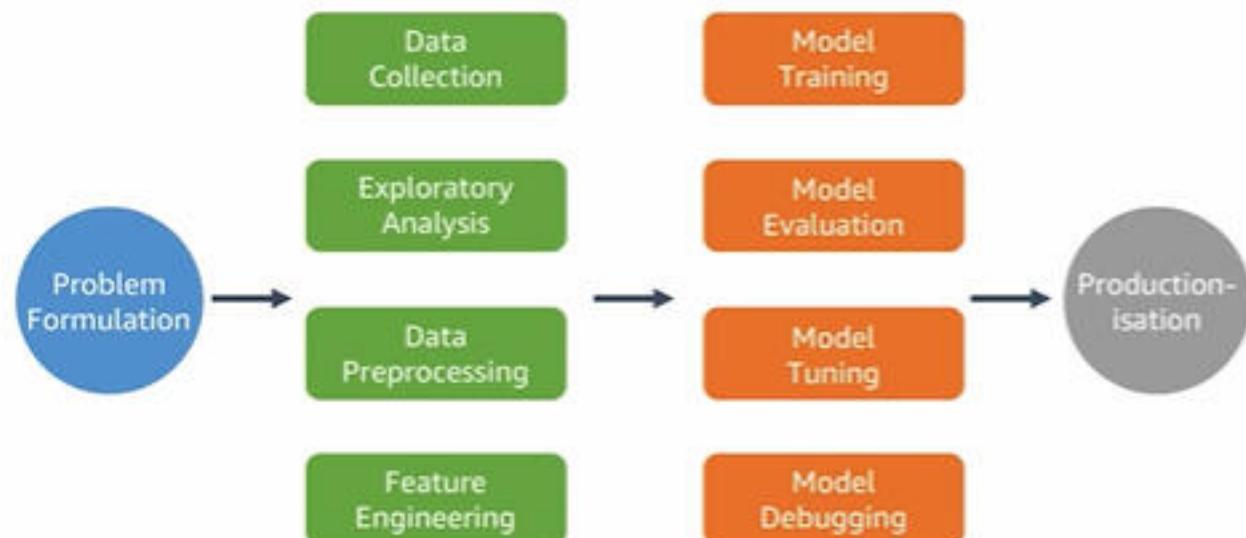
Data Matters



- Unleash the business value in data collected
- Prepare you to do data science projects and to implement production systems
- Predict future events based on past data leading to proactive change than reactive

.....	ABC
- - -	DEF
-----	MNO
- - . -	XYZ
—	TUV

The Data Science and ML Workflow



Important Concepts



- Dataset
- Training set versus test set
- Feature = attribute = independent variable = predictor



Important Concepts



- Label = target = outcome = class = dependent variable = response
- Dimensionality = number of features
- Model selection

Learning with feedback provided



Supervised learning

A “teacher” provides training examples, each with the correct label.

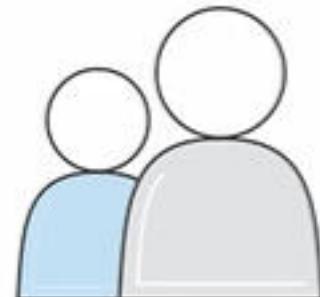
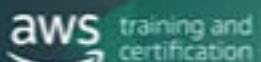


Image	Label
	Earth
	Not Earth
	Not Earth
	Earth

Learning with feedback provided



Supervised learning

A "teacher" provides training examples, each with the correct label.



Image	Label
	Earth
	Not Earth
	Not Earth
	Earth

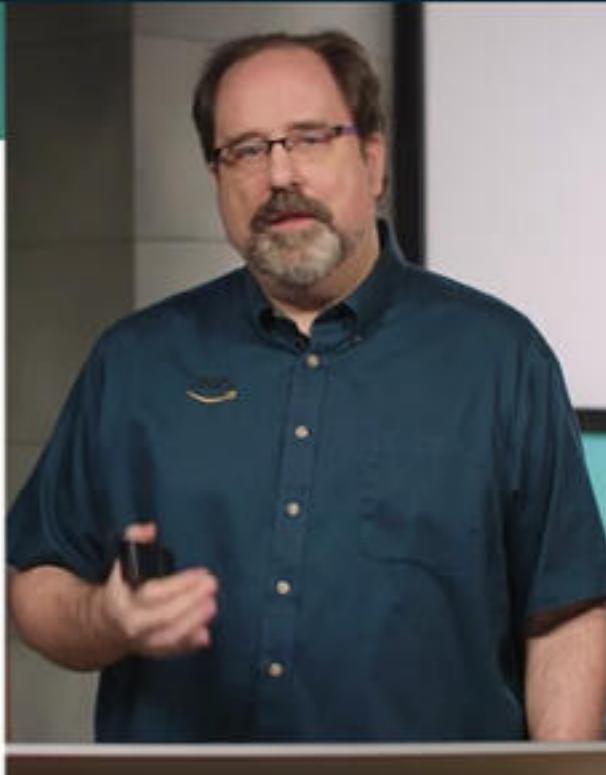


Other types of ML



Unsupervised learning

- Correct label not available for training examples; must find patterns in data (e.g., using clustering)
 - Example: Grouping customers according to what books and movies they like



Other types of ML



Reinforcement learning

- Not told what action is correct, but given some reward or penalty after each action in a sequence
 - Example: Learning how to play soccer



Data quality



- Consistency of the data
- Accuracy of the data
- Noisy data
- Missing data
- Outliers in the data
- Bias
- Variance, etc.

Data quality



- Consistency of the data
- Accuracy of the data
- Noisy data
- Missing data
- Outliers in the data
- Bias
- Variance, etc.



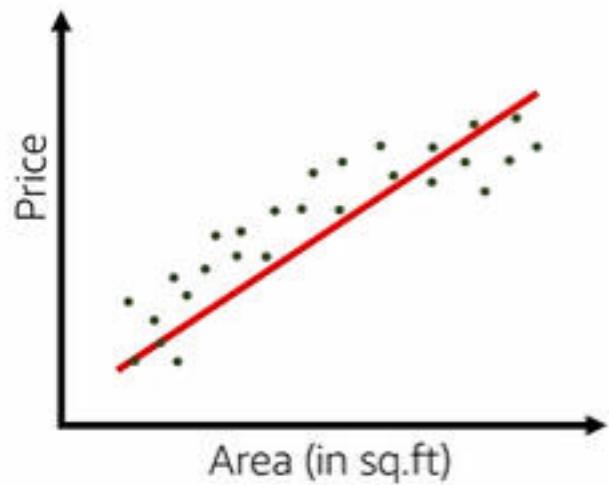
Data quality



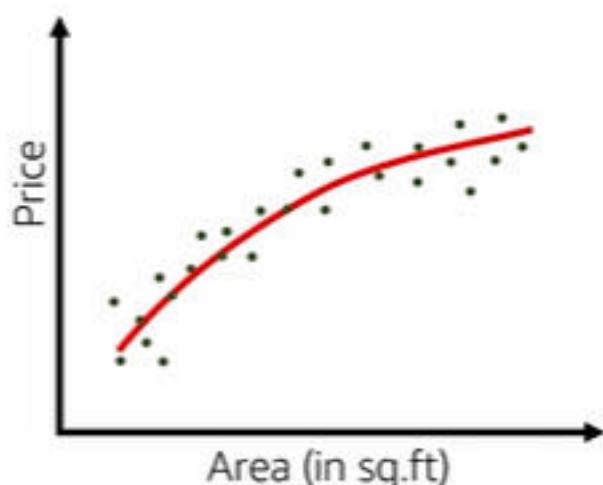
- Consistency of the data
- Accuracy of the data
- Noisy data
- Missing data
- Outliers in the data
- Bias
- Variance, etc.



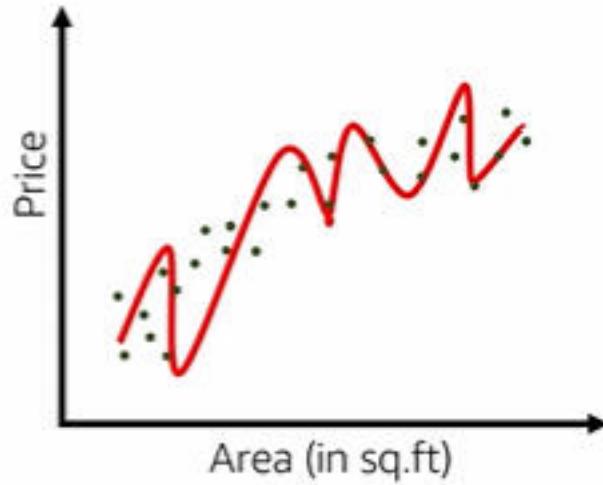
Model quality: Underfitting versus overfitting



Underfitting



Good fit



Overfitting

Overfitting and underfitting



Overfitting

- Failure to generalize: Model performs well on training set but poorly on test set
- Typically indicates that model is **too flexible** for amount of training data
- Flexibility allows it to **"memorize"** the data, including **noise**
- Corresponds to **high variance** – small changes in the training data lead to big changes in the results



Overfitting and underfitting



Overfitting

- Failure to generalize: Model performs well on training set but poorly on test set
- Typically indicates that model is **too flexible** for amount of training data
- Flexibility allows it to “memorize” the data, including **noise**
- Corresponds to **high variance** – small changes in the training data lead to big changes in the results

Underfitting

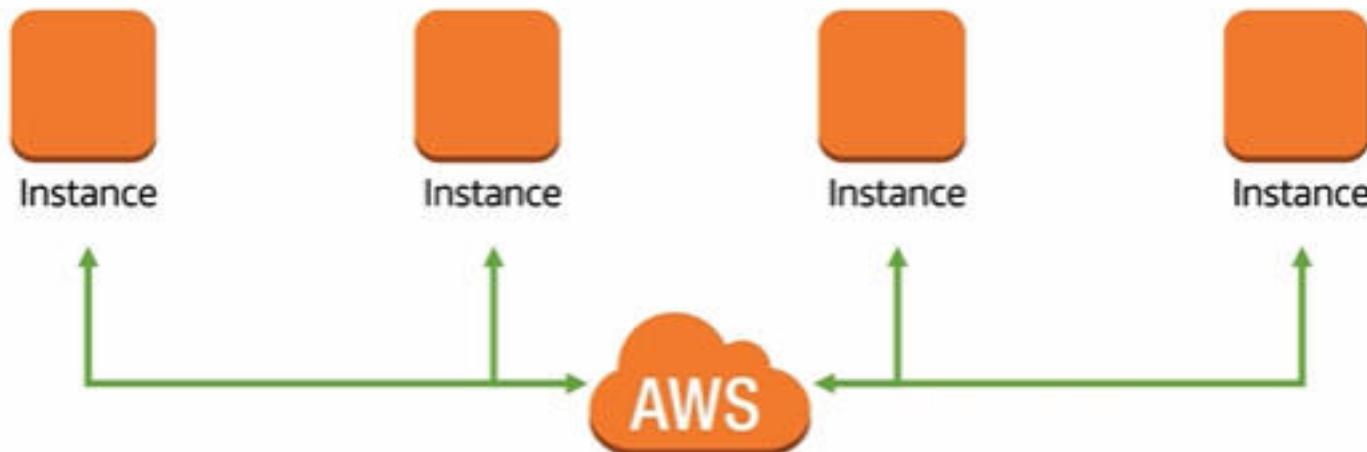
- Failure to capture important patterns in the training data set
- Typically indicates model is **too simple** or there are too few explanatory variables
- **Not flexible** enough to model real patterns
- Corresponds to **high bias** – the results show systematic lack of fit in certain regions

Computation speed and scalability



Use distributed computing systems like Amazon SageMaker or Amazon EC2 instances for training, in order to:

- Increase speed
- Solve prediction time complexity
- Solve space complexity

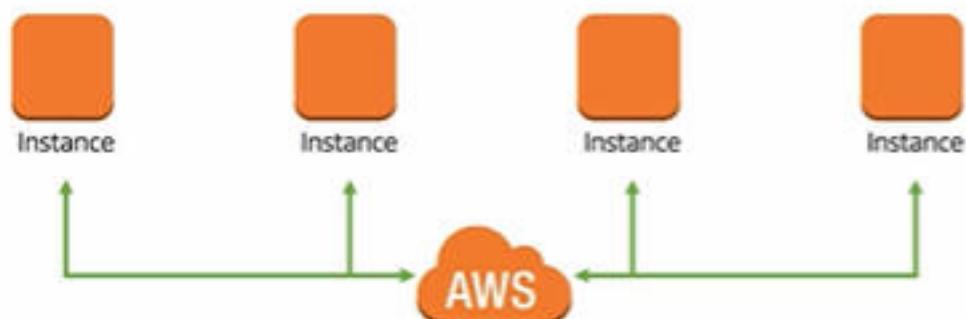


Computation speed and scalability



Use distributed computing systems like Amazon SageMaker or Amazon EC2 instances for training, in order to:

- Increase speed
- Solve prediction time complexity
- Solve space complexity



Topics



- Linear regression
- Univariate linear regression
- Multivariate linear regression

Linear methods

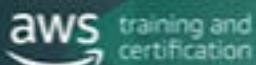
- Parametric methods where function learned has form $f(x) = \phi(w^T x)$ where ϕ is some activation function
- Generally, optimized by learning weights by applying (stochastic) gradient descent to minimize loss function, e.g. $\sum |\hat{y}_i - y_i|^2$
- Simple; a good place to start for a new problem, at least as a baseline
- Methods
 - Linear regression for numeric target outcome
 - Logistic regression for categorical target outcome

Linear regression (univariate)

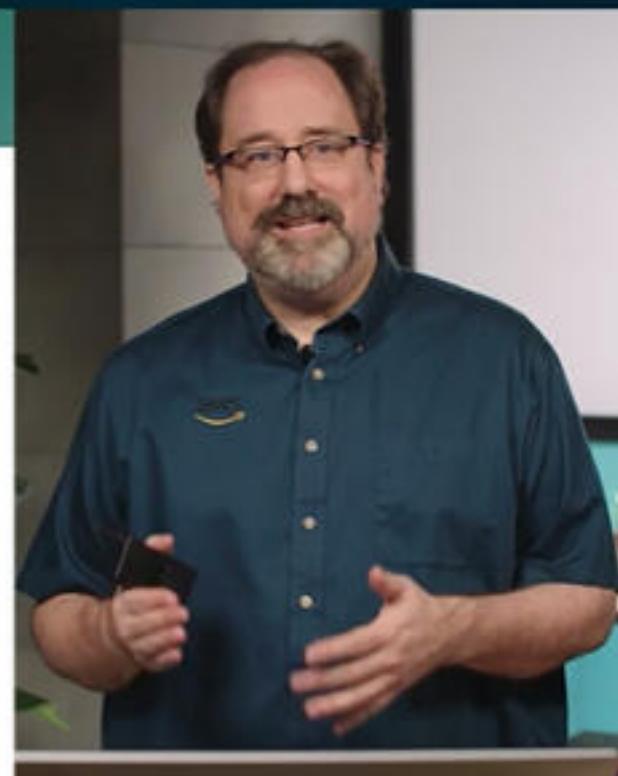
- Model relation between a single feature (explanatory variable x) and a real-valued response (target variable y)
- Given data (x, y) , and a line defined by w_0 (intercept) and w_1 (slope), the vertical offset for each data point from the line is the error between the true label y and the prediction based on x
- The best line minimizes the sum of squared errors (SSE)
- We usually assume the error is Gaussian distributed with mean zero and fixed variance



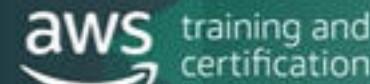
Linear regression (univariate)



- Model relation between a single feature (explanatory variable x) and a real-valued response (target variable y)
- Given data (x, y) , and a line defined by w_0 (intercept) and w_1 (slope), the vertical offset for each data point from the line is the error between the true label y and the prediction based on x
- The best line minimizes the **sum of squared errors (SSE)**
- We usually assume the error is Gaussian distributed with mean zero and fixed variance



Linear regression (multivariate)



- Multiple linear regression includes N explanatory variables with $N \geq 2$:

$$y = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m = \sum_{i=0}^N w_i x_i$$

- Sensitive to correlation between features, resulting in high variance of coefficients
- scikit-learn implementation: `sklearn.linear_model.LinearRegression`

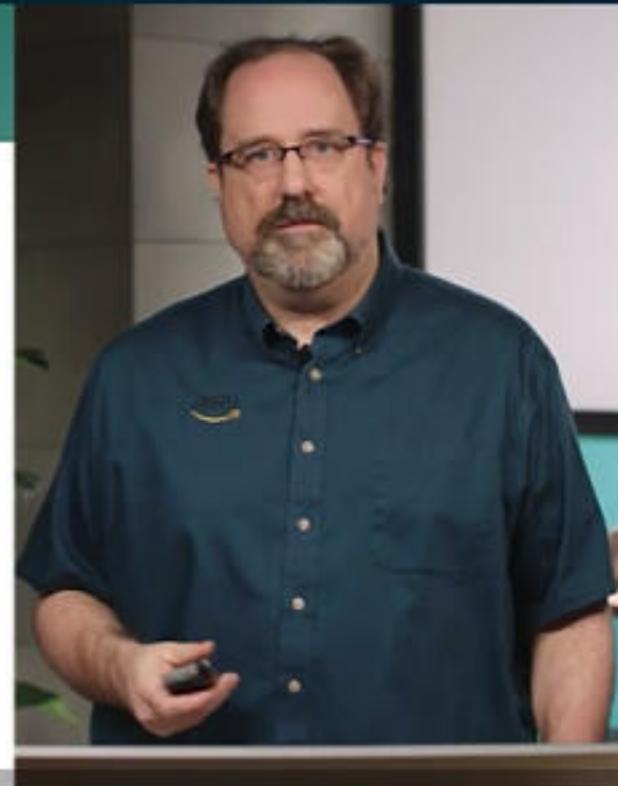
Linear regression (multivariate)



- Multiple linear regression includes N explanatory variables with $N \geq 2$:

$$y = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m = \sum_{i=0}^N w_i x_i$$

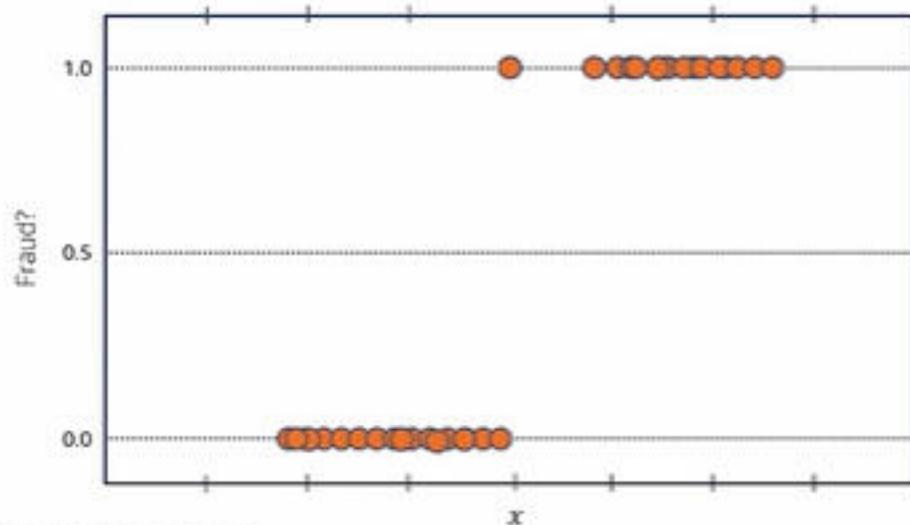
- Sensitive to correlation between features, resulting in high variance of coefficients
- scikit-learn implementation: `sklearn.linear_model.LinearRegression`



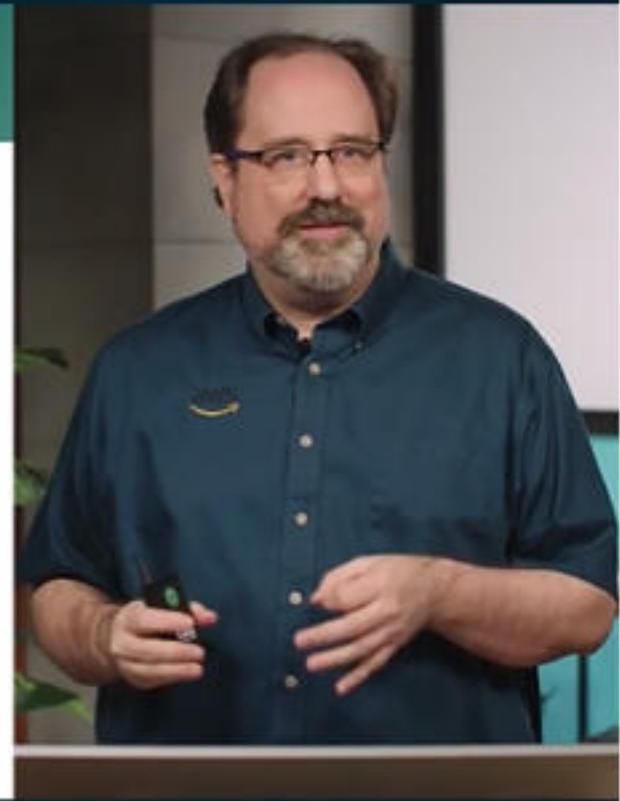
Logistic regression



Predict whether a credit card transaction is fraud



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Logistic regression

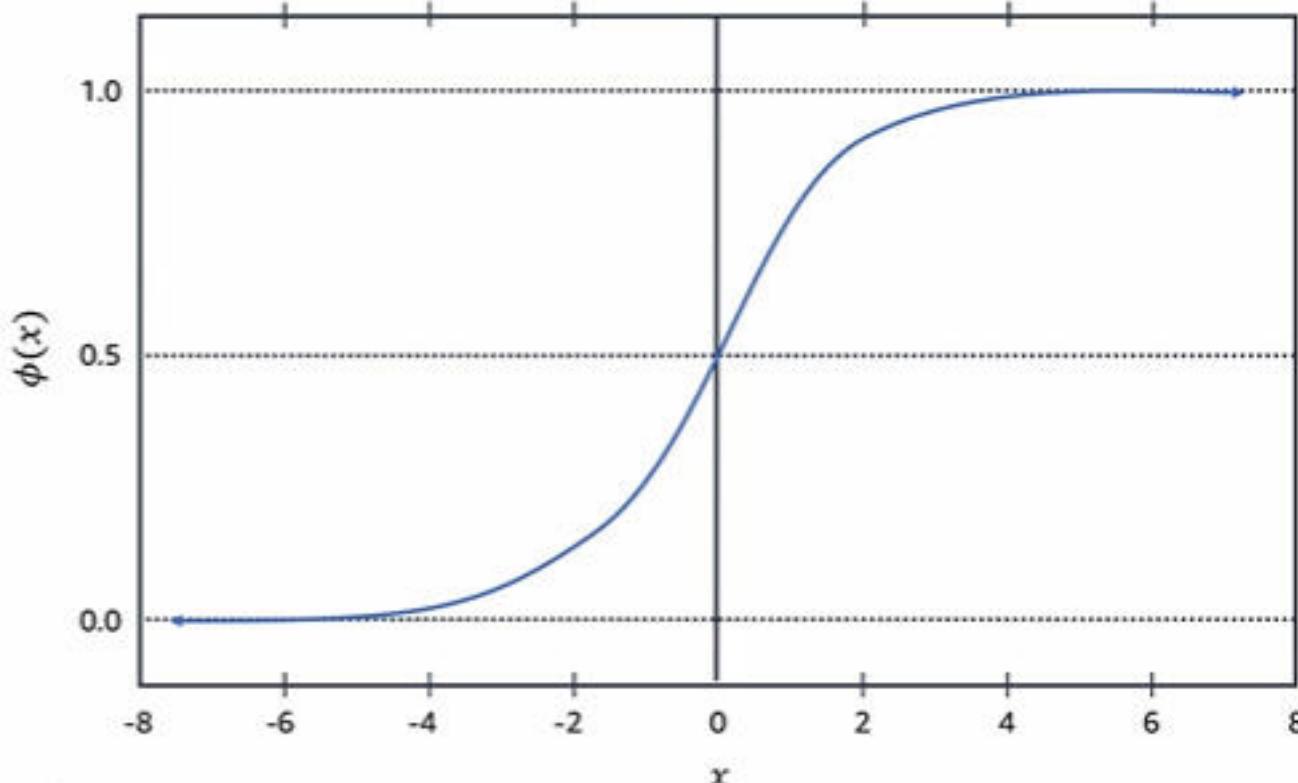


- Estimates the probability of the input belonging to one of two classes: positive and negative
- Vulnerable to outliers in training data
- scikit-learn: `sklearn.linear_model.LogisticRegression`
- Relationship to linear model: $\sigma(z) = \frac{1}{1+e^{-z}}$
 - z is a trained multivariate linear function
 - σ is a fixed univariate function (not trained)
 - Objective function to maximize = probability of the true training labels



Logistic regression: Sigmoid curve

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



Logistic regression



- Model relation between features (explanatory variables x) and the binary responses ($y = 1$ or $y = 0$)
- For all the features, define a linear combination:

$$z = w^T x = w_0 + w_1 x_1 + \cdots + w_n x_n$$

- Define probability of $y = 1$ given x as p and find the logit of p as

$$\text{logit}(p) = \log \frac{p}{(1 - p)}$$

Logistic regression



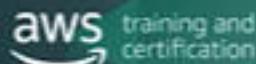
- Model relation between features (explanatory variables x) and the binary responses ($y = 1$ or $y = 0$)
- For all the features, define a linear combination:

$$z = w^T x = w_0 + w_1 x_1 + \dots + w_n x_n$$

- Define probability of $y = 1$ given x as p and find the logit of p as

$$\text{logit}(p) = \log \frac{p}{(1 - p)}$$

Logistic regression

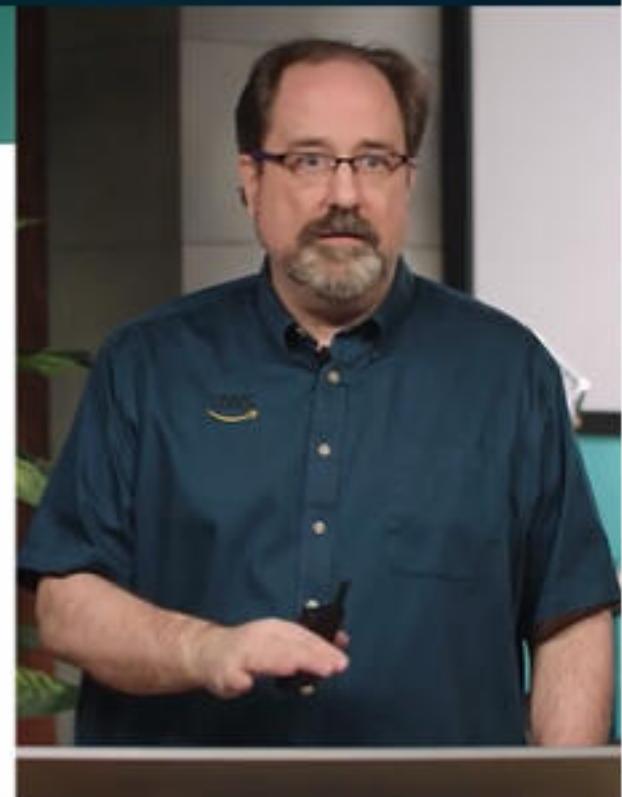


- Model relation between features (explanatory variables x) and the binary responses ($y = 1$ or $y = 0$)
- For all the features, define a linear combination:

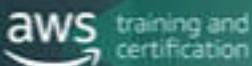
$$z = w^T x = w_0 + w_1 x_1 + \cdots + w_n x_n$$

- Define probability of $y = 1$ given x as p and find the logit of p as

$$\text{logit}(p) = \log \frac{p}{(1 - p)}$$



Logistic regression



- Model relation between features (explanatory variables x) and the binary responses ($y = 1$ or $y = 0$)
- For all the features, define a linear combination:

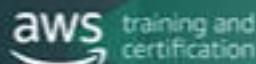
$$z = w^T x = w_0 + w_1 x_1 + \cdots + w_n x_n$$

- Define probability of $y = 1$ given x as p and find the logit of p as

$$\text{logit}(p) = \log \frac{p}{(1 - p)}$$



Logistic regression



- Model relation between features (explanatory variables x) and the binary responses ($y = 1$ or $y = 0$)
- For all the features, define a linear combination:

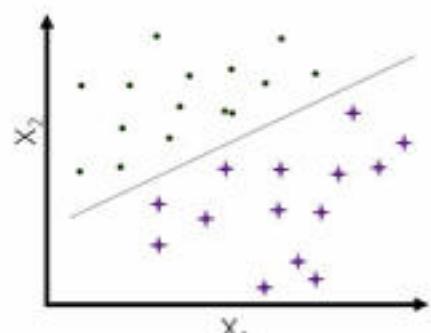
$$z = w^T x = w_0 + w_1 x_1 + \cdots + w_n x_n$$

- Define probability of $y = 1$ given x as p and find the logit of p as

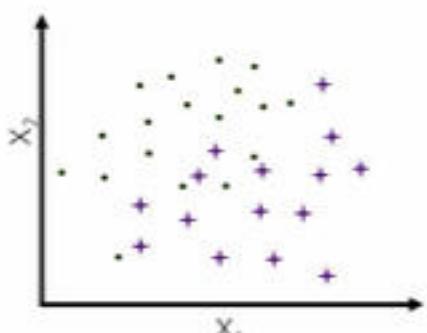
$$\text{logit}(p) = \log \frac{p}{(1 - p)}$$



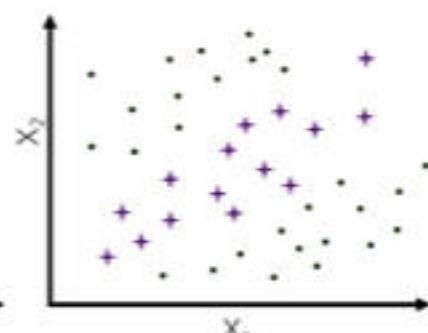
Linearly separable versus non-linearly separable



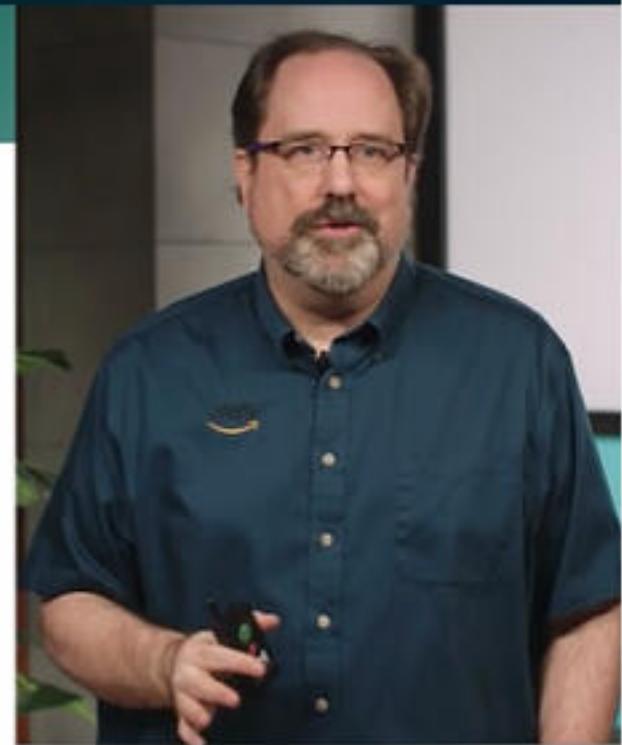
Linearly separable



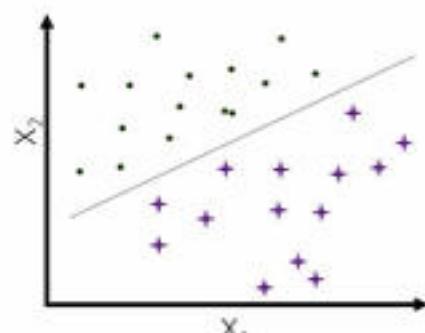
Not linearly separable



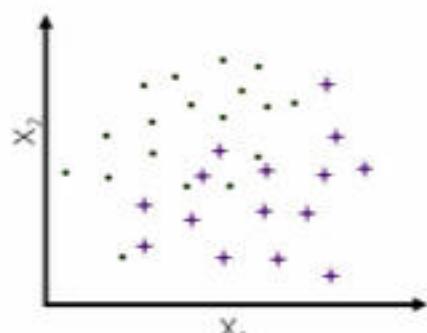
Not linearly separable



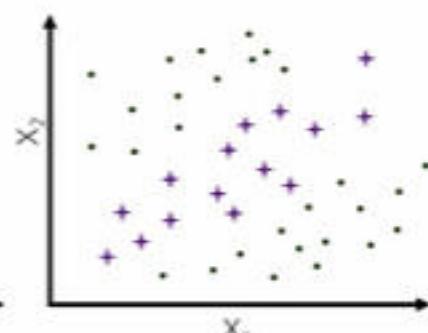
Linearly separable versus non-linearly separable



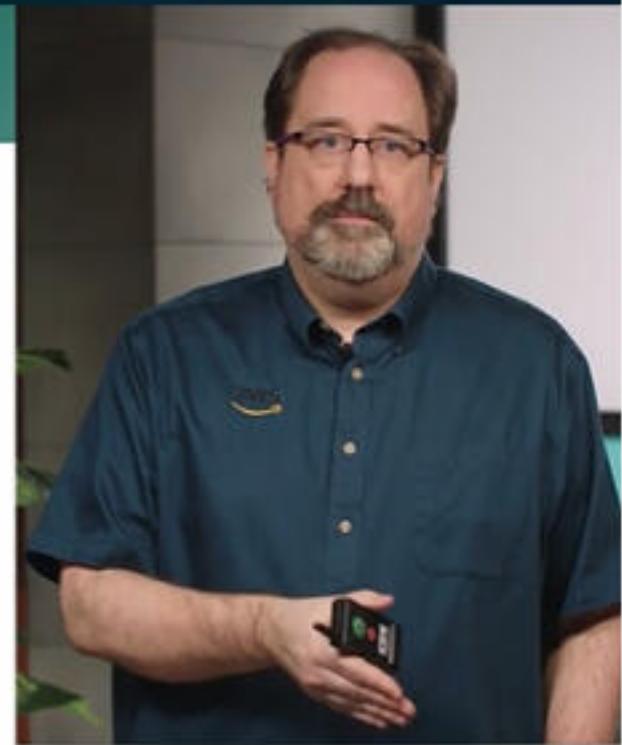
Linearly separable



Not linearly separable

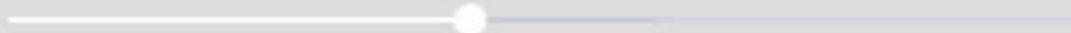


Not linearly separable



Data collection: sampling

- The importance of being unbiased
- Issues to watch for with sampling



-1:01

1x



Data collection: Labeling



- Labeling components and tools
- Managing labelers



Exploratory data analysis: Domain knowledge

- How domain knowledge can help in exploring the data
- What to do when the domain is outside your own expertise



-0:35

1x



Exploratory data analysis: Data issues



- Anticipate various types of data issues



-0:03

1x



Problem formulation



What is the problem you need to solve?

What is the business metric?

Is ML the appropriate approach?

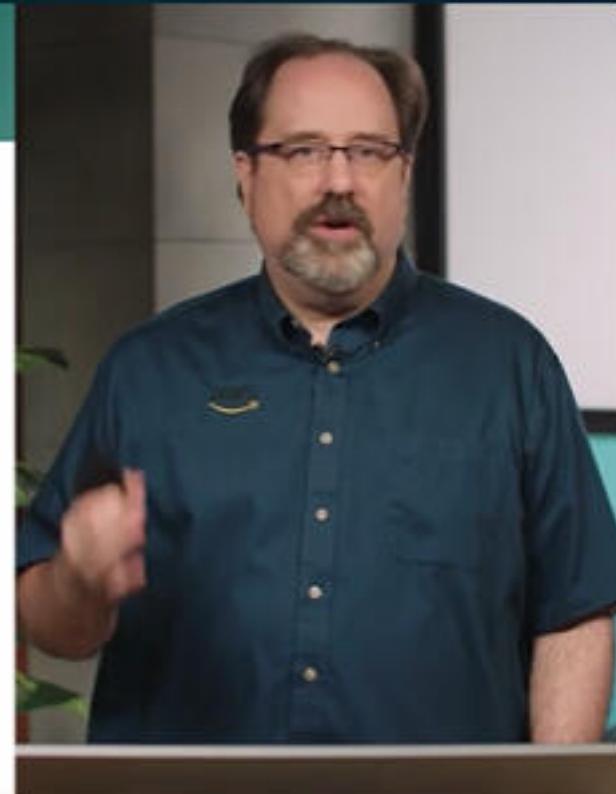
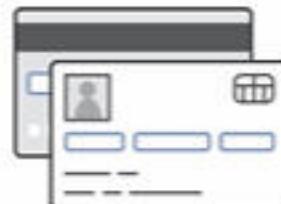
What data is available?

What type of ML problem is it?

What are your goals?

Precisely describe the business problem that you are trying to solve.

Example: Create a way to classify whether the customer credit card transaction is fraudulent.



Problem formulation



What is the problem you need to solve?

What is the business metric?

Is ML the appropriate approach?

What data is available?

What type of ML problem is it?

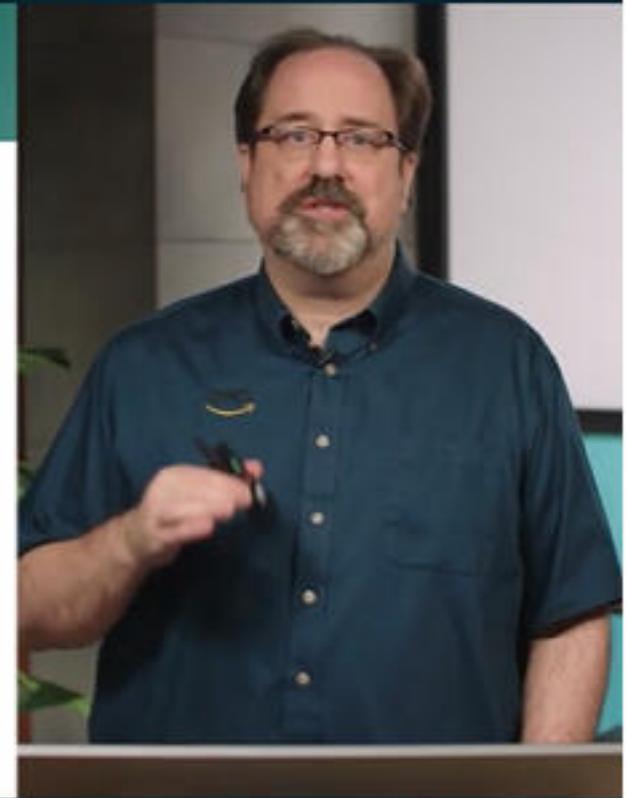
What are your goals?

Determine appropriate metrics to measure:

- Quality
- Impact of the solution

Link financial impact with analytics metrics

Example: Rate at which customers are misclassified



Problem formulation



What is the problem you need to solve?

What is the business metric?

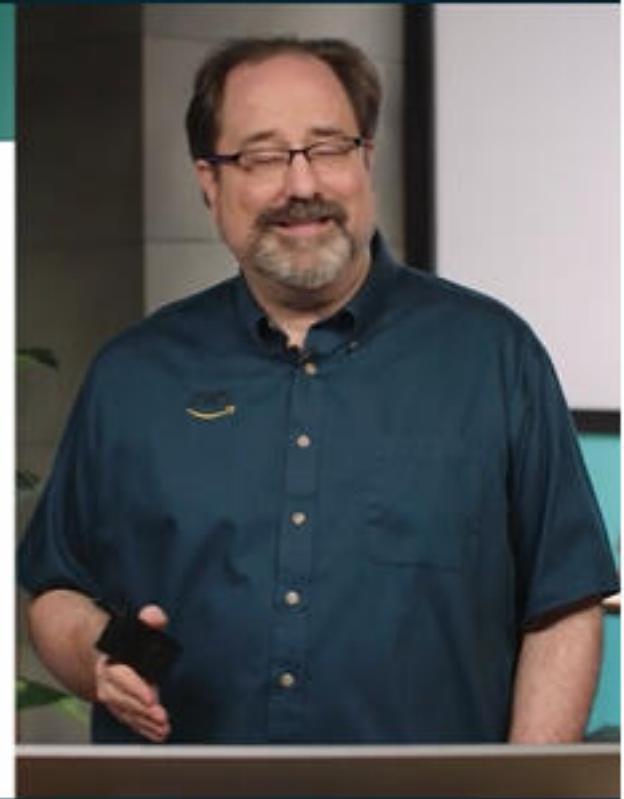
Is ML the appropriate approach?

What data is available?

What type of ML problem is it?

What are your goals?

- Summarize the data available.
- Determine the gap between the data you want and the actual data that's available.
- Is there a way to add more data?
- What are the data sources?



Problem formulation



What is the problem you need to solve?

What is the business metric?

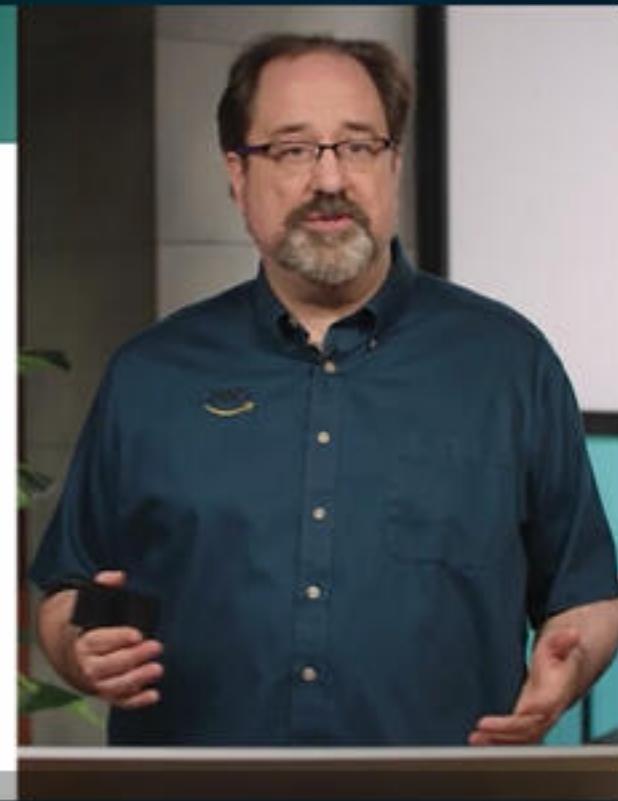
Is ML the appropriate approach?

What data is available?

What type of ML problem is it?

What are your goals?

- Establish technical ML goals.
- Define the criteria for successful outcome of the project.



Data Collection

- Data collection: The process of acquiring training and/or test data
- Motivation:
 - Initial datasets for training models and measuring success
 - Additional data for tuning
 - Replacements for flawed or outdated sets
 - Data and model maintenance post-production
- Sources:
 - Logs
 - Databases
 - Web sites (crawling and scraping)
 - Data providers (public or private)

Data Collection



- **Data collection:** The process of acquiring training and/or test data
- **Motivation:**
 - Initial datasets for training models and measuring success
 - Additional data for tuning
 - Replacements for flawed or outdated sets
 - Data and model maintenance post-production
- **Sources:**
 - Logs
 - Databases
 - Web sites (crawling and scraping)
 - Data providers (public or private)



Open Data



- AWS provides a **comprehensive tool kit** for sharing and analyzing data at any scale.
- When organizations make data open on AWS, **the public can analyze it quickly and easily** with AWS scalable computing and analytics services.



Registry of Open Data on AWS



This registry exists to help people **discover** and **share** datasets that are available via AWS resources.

Registry of Open Data on AWS

[About](#)
This registry exists to help people discover and share datasets that are available via AWS resources. [Learn more about sharing data via AWS.](#)
[See all usage examples for datasets listed in this registry.](#)

Search datasets (currently 54 matching datasets)

Add to this registry
If you want to add a dataset or example of how to use a dataset to this registry, please follow the instructions on the [Registry of Open Data on AWS GitHub repository](#).
Unless specifically stated in the applicable dataset documentation, datasets available through the Registry of Open Data on AWS are not provided and maintained by AWS. Datasets are provided and maintained by a variety of third parties under a variety of licenses. Please check dataset licenses and related documentation to determine if a dataset may be used for your application.

[View details](#) | [Edit this page](#) | [View history](#) | [View contributors](#)

Sentinel-2

[Cloud coverage](#) | [Cloud imagery](#) | [GP](#) | [Natural resources](#)

The **Sentinel-2 mission** is a land monitoring constellation of two satellites that provide high resolution optical imagery and provide continuity for the current SPOT and Landsat missions. The mission provides a global coverage of the Earth's land surface every 5 days, making the data of great use in on-going studies. L1C data are available from June 2015 globally. L2A data are available from April 2017 over wider Europe region, planned to be expanded globally in July 2018.

[View details](#) | [Edit this page](#) | [View history](#) | [View contributors](#)

Usage examples:

- [Amazon Digital Imagery Browser](#) by Astro-Digital
- [QGIS plugin for Sentinel-2 data](#) by Senerge
- [Exploring the Chile wildfires with Landsat and Sentinel-2 imagery](#) by Timothy Whitehead
- [Sentinel-2 Cloudless Atlas](#) by EOX
- [PME Landsat-8/Sentinel-2 File Selector](#) by Safe Software

[See 14 usage examples](#) | [View details](#) | [Edit this page](#) | [View history](#) | [View contributors](#)

Landsat 8

[Cloud coverage](#) | [Cloud imagery](#) | [GP](#) | [Natural resources](#)

An ongoing collection of satellite imagery of all land on Earth produced by the Landsat 8 satellite.

[View details](#) | [Edit this page](#) | [View history](#) | [View contributors](#)

Usage examples:

- [Development Seed Geotambo](#) by Matthew Hansen
- [Using Vector Tiles and AWS Lambda, we can build a really simple API to get Landsat and Sentinel Images](#) by Remote Pixel
- [Sentinel Playground for Landsat](#) by Senerge
- [Exploring the Chile wildfires with Landsat and Sentinel-2 imagery](#) by Timothy Whitehead
- [A Gentle Introduction to GDAL Part 4: Working with Satellite Data](#) by Planet

[See 8 usage examples](#) | [View details](#) | [Edit this page](#) | [View history](#) | [View contributors](#)

Sampling

Selecting a subset of instances
for training and testing

Labeling

Obtaining gold-standard answers
for supervised learning

Terminology: Instance = example = data point

Data Collection



Sampling

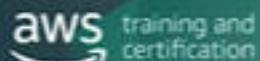
Selecting a subset of instances
for training and testing

Labeling

Obtaining gold-standard answers
for supervised learning

Terminology: Instance = example = data point

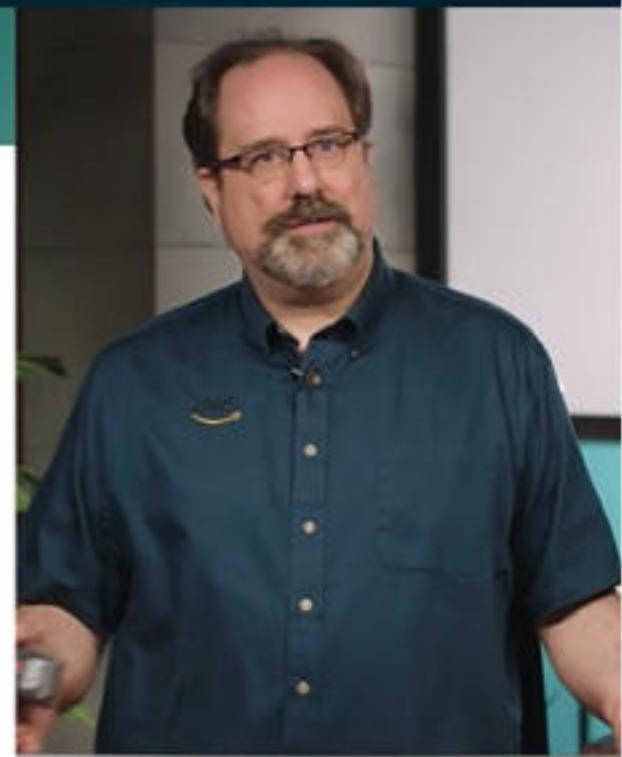
Sampling



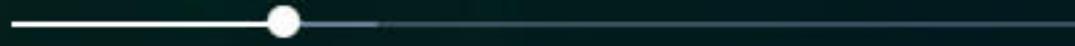
Representativity: Sample needs to be representative of the expected production population; i.e., *unbiased*

- Especially important for testing and measurement sets
- It's also important for training sets to get good generalization

Random sampling: Sampling so that each source data point has equal probability of being selected



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



-3:59

2x



Stratified Sampling



Issue: With random sampling, rare subpopulations can be under-represented (or not represented at all).

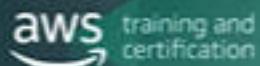
- A subpopulation is usually defined as examples within the same label.

Stratified sampling: Apply random sampling to each subpopulation separately.

Usually, the sampling probability is the same for each stratum.

- If not, weights can be used in metrics and/or directly in training

Other Issues with Sampling



Seasonality

Time of day, day of week, time of year (e.g., seasons), holidays, special events, etc.

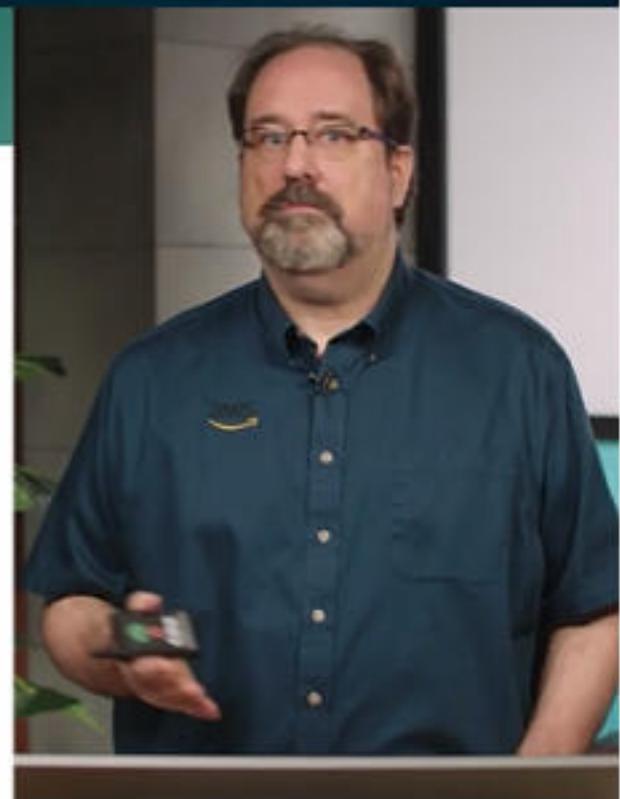
- Stratified sampling across these can minimize bias
- Visualization can help

Trends

Patterns can shift over time, and new patterns can emerge

- To detect, try comparing models trained over different time periods
- Visualization can help

Consider using validation data that was gathered after your training data was gathered



Leakage

Train/test bleed: Inadvertent overlap of training and test data when sampling to create datasets

- Sample from fresh data
- Filter out already selected instances
- Partition source data along some dimension (but avoid bias)
- Be especially careful with time-series data and data with duplicate entries

Leakage: Using information during training or validation that is not available in production

- Kaufman, Shachar, *et al.* Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data* (2012): 15.



-1:15

2x



Leakage

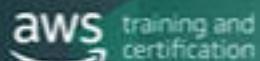
Train/test bleed: Inadvertent overlap of training and test data when sampling to create datasets

- Sample from fresh data
- Filter out already selected instances
- Partition source data along some dimension (but avoid bias)
- Be especially careful with time-series data and data with duplicate entries

Leakage: Using information during training or validation that is not available in production

- Kaufman, Shachar, *et al.* Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data* (2012): 15.

Leakage

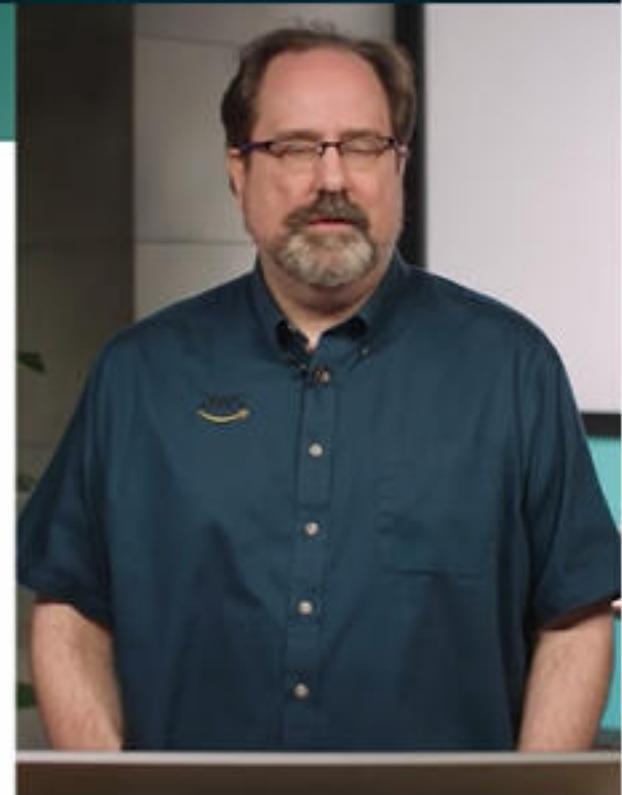


Train/test bleed: Inadvertent overlap of training and test data when sampling to create datasets

- Sample from fresh data
- Filter out already selected instances
- Partition source data along some dimension (but avoid bias)
- Be especially careful with time-series data and data with duplicate entries

Leakage: Using information during training or validation that is not available in production

- Kaufman, Shachar, et al. *Leakage in data mining: Formulation, detection, and avoidance*. *ACM Transactions on Knowledge Discovery from Data* (2012): 15.



- Motivation: Often, labels are not readily available in sampled data.
- Examples
 - Search: The results a customer wanted to receive
 - Music categorization: The genre of a piece
 - Sentiment analysis: The overall attitude of the writer
 - Digitization: The transcription of handwriting
 - Object detection: The localization of objects in images
- Sometimes labels can be inferred (for example, from click-through data)
- Human labels can be preferable to minimize bias, capture subtleties, etc.



-12:36

2x



Labeling Components



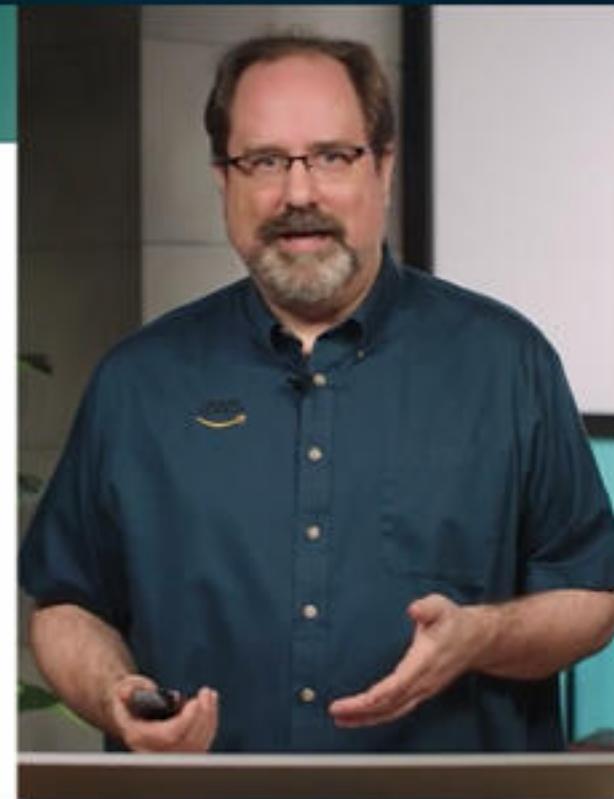
Labeling guidelines

- Instructions to labelers
- Critical to get right
- Minimize ambiguity

Labeling tools

- Technology:
 - Excel spreadsheets
 - Amazon Mechanical Turk
 - Custom-built tools
- Questions:
 - Human Intelligence Tasks (HITs) should be:
 - Simple
 - Unambiguous

Poor design of either can: 1. Impact labeler productivity and quality, 2. Introduce bias



Labeling Components

Labeling guidelines

- Instructions to labelers
- Critical to get right
- Minimize ambiguity

Labeling tools

- Technology:
 - Excel spreadsheets
 - Amazon Mechanical Turk
 - Custom-built tools
- Questions:
 - Human Intelligence Tasks (HITs) should be:
 - Simple
 - Unambiguous

Poor design of either can: 1. Impact labeler productivity and quality, 2. Introduce bias

Amazon Mechanical Turk

- Obtain **human intelligence on demand**
- Access a global, on-demand, 24x7 **workforce**
- Pay only for what you use
- Use for **labeling**

Labeling Tools

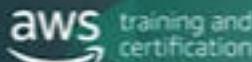
Amazon Mechanical Turk



Managing Labelers

- Motivation
- Plurality
- Gold standard HITs
- Auditors
- Labeler incentives
- Quality and productivity metrics

Managing Labelers



Labeler incentives

- Compensation
- Rewards
- Voluntary
- Gamification



© 2018 Amazon Web Services, Inc. or its Affiliates. All Rights Reserved.



-3:26

2x



Sampling and Treatment Assignment

	Random Assignment	No Random Assignment	
Random Sampling	Ideal experiments: Causal conclusion and can be generalized (rarely available in traditional analysis, but become available in online testing)	Typical survey or observation studies: Cannot establish causation, but can find correlation and can be generalized (additional work needed to infer causation)	Generalization
No Random Sampling	Most experiments: Causal conclusion for the sample only (more work is needed to generalize)	Badly-designed survey or pooled studies: Cannot establish causation, cannot generalize to larger population (more work is needed to draw useful conclusions)	No Generalization
	Causation	Correlation	

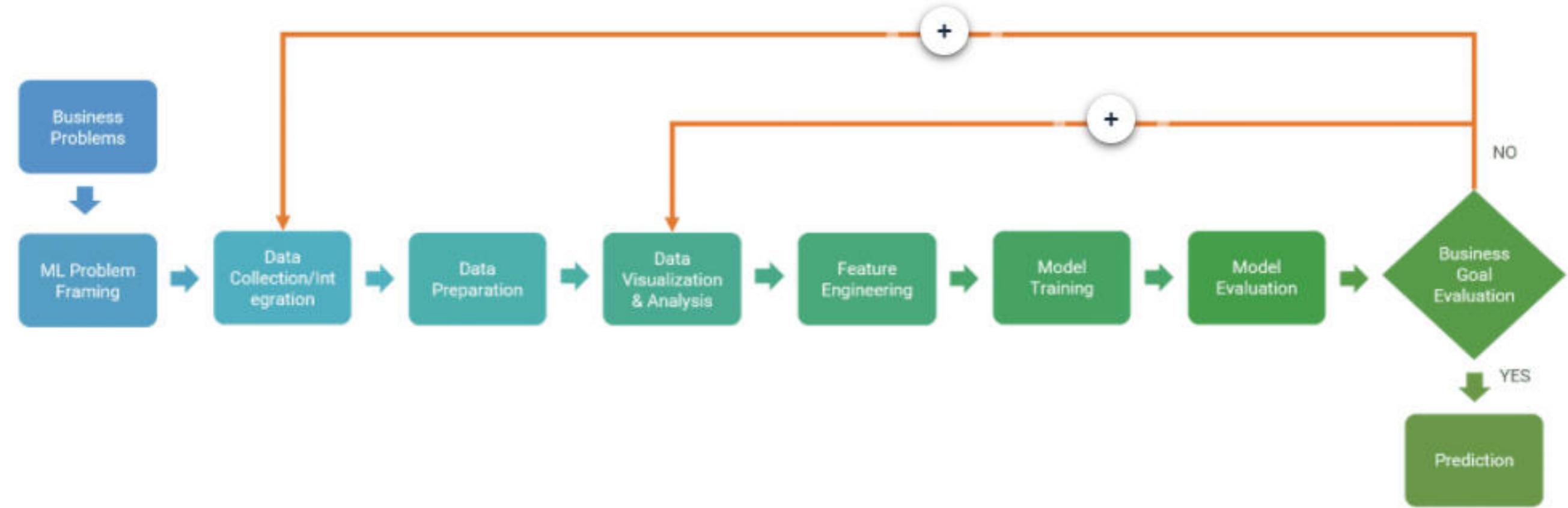
Sampling and Treatment Assignment



	Random Assignment	No Random Assignment	
Random Sampling	Ideal experiments: Causal conclusion and can be generalized (rarely available in traditional analysis, but become available in online testing)	Typical survey or observation studies: Cannot establish causation, but can find correlation and can be generalized (additional work needed to infer causation)	Generalization
No Random Sampling	Most experiments: Causal conclusion for the sample only (more work is needed to generalize)	Badly-designed survey or pooled studies: Cannot establish causation, cannot generalize to larger population (more work is needed to draw useful conclusions)	No Generalization
	Causation	Correlation	



Typical ML workflow

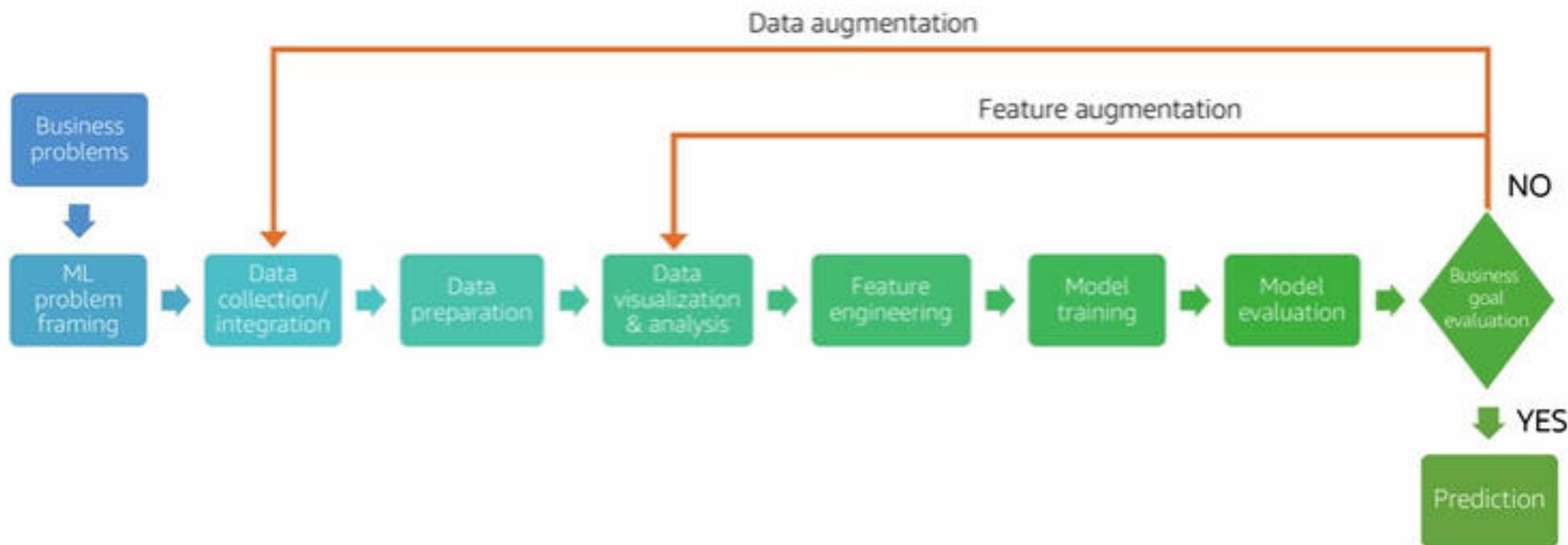


Topics

- ML workflow
- Domain knowledge

The ML workflow

aws training and certification

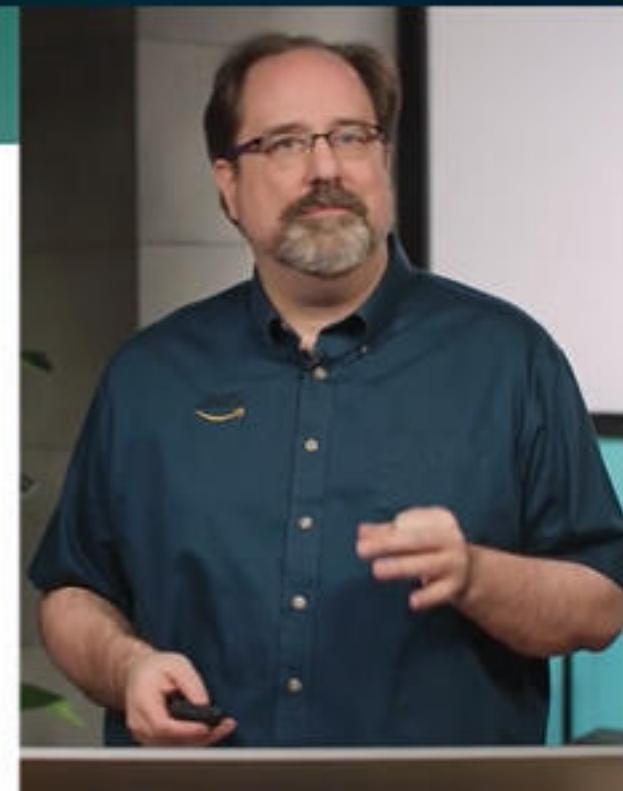


Exploratory data analysis



Important to understand data as much as possible

Informs subsequent steps in the ML process



Domain knowledge



Understand how the domain works, important dynamics and relationships, constraints, how data is generated, etc.

Better understanding of domain leads to better features, better debugging, better metrics, etc.



-2:26

2x



When domain is outside expertise



Consult domain experts

- AWS ML specialist SAs
- AWS Professional Services
- AWS ML Solutions Lab



Seek other resources

- AWS Partner Network



Amazon ML Solutions Lab



Developing machine learning skills through collaboration and education



Brainstorming



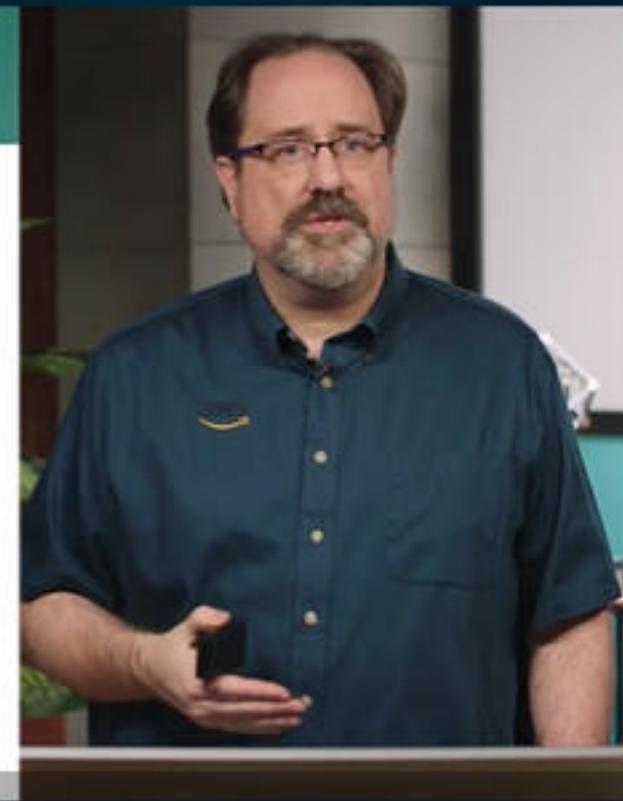
Custom
modeling



Training



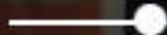
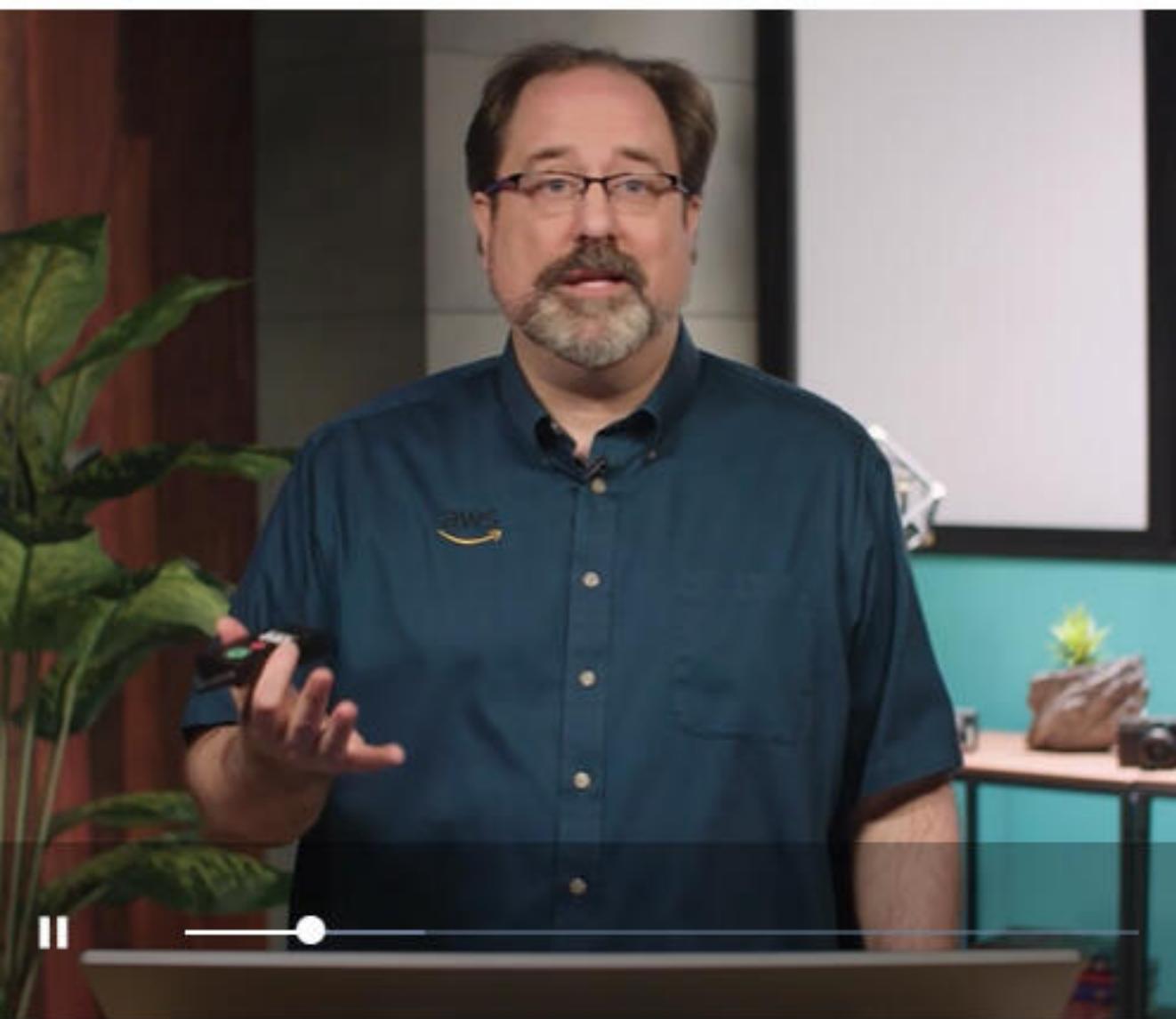
On-site with
Amazon experts



0:21

2x



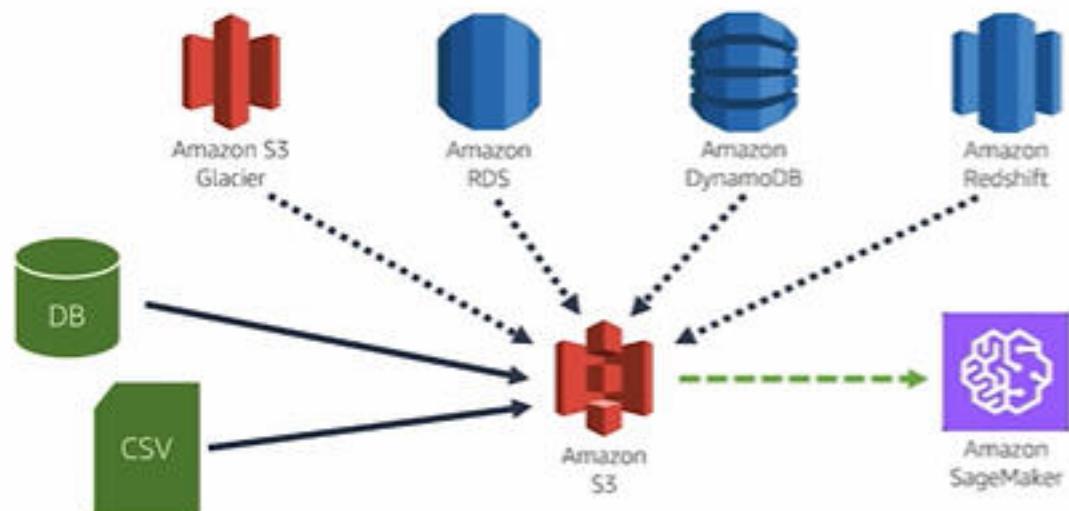
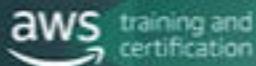


-3:09

2x



Data sources



Dataset schema

*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

ID	Diagnosis	Radius mean
842302	M	17.99
842517	M	20.57
8510426	B	13.54
8510653	B	13.08



Label: Diagnosis
Type: Categorical/Binary
Range: [M=malignant, B=benign]

Label: ID
Type: Text/Numerical
Range: [8670-911320502]

Data schema: Merge/joins

- **Types:** Categorical, ordinal, numerical, date, vector, text, image, unstructured, etc.
- Use pandas DataFrame merge/join to join two datasets

```
df = pd.DataFrame({"Name": ["John Doe", "Jim Smith", "Elizabeth Shane", "David Lee", "Hernando Vasquez"],  
                   "Job": ["Marketing", "HR", "SDE", "SDE", "SDE"]})  
df_1 = pd.DataFrame({"VP": ["Amy Shu", "Jane Heart", "Ashish Kapoor"],  
                     "Job": ["SDE", "HR", "Marketing"]})  
  
df.merge(df_1, on="Job", how = 'inner')
```

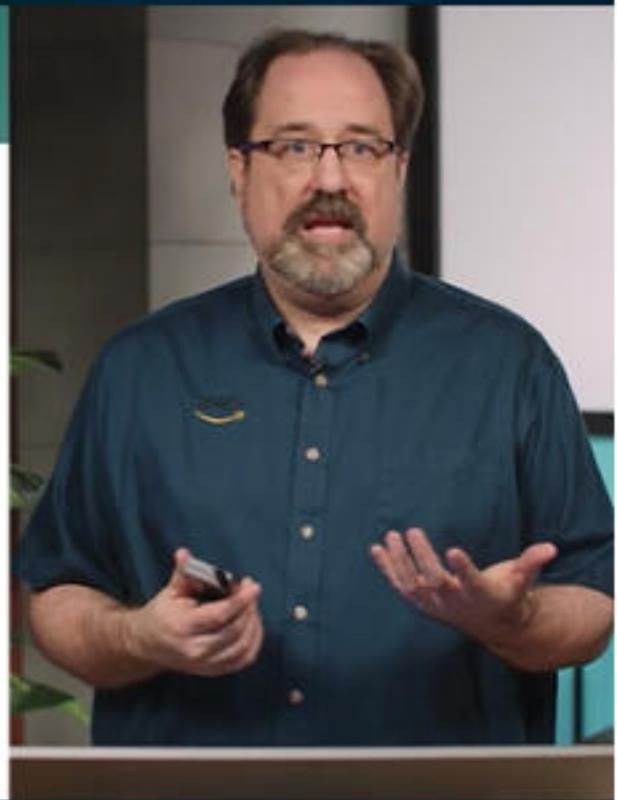
	Job	Name	VP
0	Marketing	John Doe	Ashish Kapoor
1	HR	Jim Smith	Jane Heart
2	SDE	Elizabeth Shane	Amy Shu
3	SDE	David Lee	Amy Shu
4	SDE	Hernando Vasquez	Amy Shu

Data schema: Merge/joins

- **Types:** Categorical, ordinal, numerical, date, vector, text, image, unstructured, etc.
- Use pandas DataFrame merge/join to join two datasets

```
df = pd.DataFrame({"Name": ["John Doe", "Jim Smith", "Elizabeth Shane", "David Lee", "Hernando Vasquez"],  
                   "Job": ["Marketing", "HR", "SDE", "SDE", "SDE"]})  
df_1 = pd.DataFrame({"VP": ["Amy Shu", "Jane Heart", "Ashish Kapoor"],  
                     "Job": ["SDE", "HR", "Marketing"]})  
  
df.merge(df_1, on="Job", how = "inner")
```

	Job	Name	VP
0	Marketing	John Doe	Ashish Kapoor
1	HR	Jim Smith	Jane Heart
2	SDE	Elizabeth Shane	Amy Shu
3	SDE	David Lee	Amy Shu
4	SDE	Hernando Vasquez	Amy Shu



Overall statistics

- Number of instances (i.e. number of rows)
- Number of attributes (i.e. number of columns)

Attribute statistics (univariate)

- Statistics for numeric attributes (mean, variance, etc.) -- `df.describe()`
- Statistics for categorical attributes (histograms, mode, most/least frequent values, percentage, number of unique values)
 - Histogram of values: E.g., `df[<attribute>].value_counts()` or seaborn's `distplot()`
- Target statistics
 - Class distribution: E.g., `df[<target>].value_counts()` or `np.bincount(y)`

Descriptive Statistics

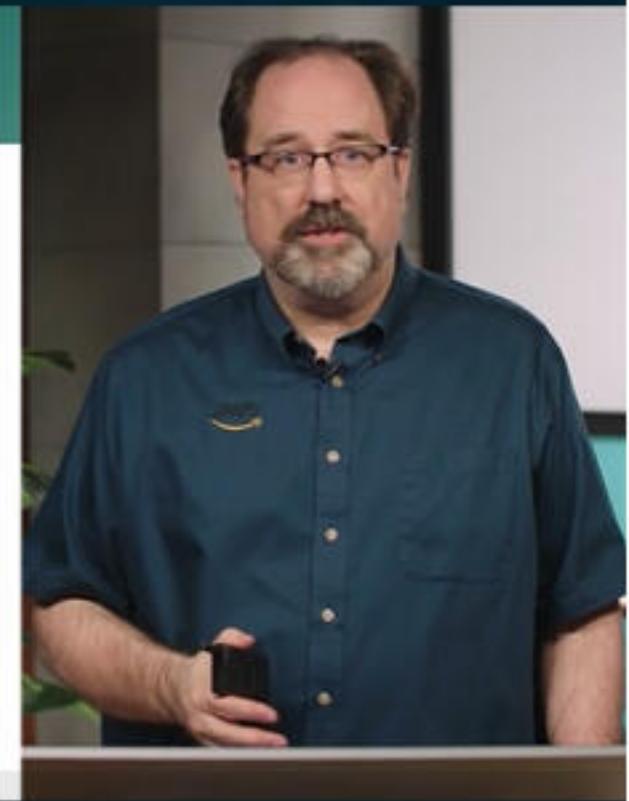


Overall statistics

- Number of instances (i.e. number of rows)
- Number of attributes (i.e. number of columns)

Attribute statistics (univariate)

- Statistics for numeric attributes (mean, variance, etc.) -- `df.describe()`
- Statistics for categorical attributes (histograms, mode, most/least frequent values, percentage, number of unique values)
 - Histogram of values: E.g., `df[<attribute>].value_counts()` or seaborn's `distplot()`
- Target statistics
 - Class distribution: E.g., `df[<target>].value_counts()` or `np.bincount(y)`



Breast Cancer Classification Problem



training and
certification

```
import pandas as pd
from sklearn.datasets import load_breast_cancer

dataset = load_breast_cancer()
cols = ['V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',
        'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'V29',
        'V30', 'V31', 'V32', 'V33', 'V34', 'V35', 'V36', 'V37', 'V38', 'V39']

df = pd.DataFrame(dataset['data'], columns = cols)
df['target'] = dataset.target

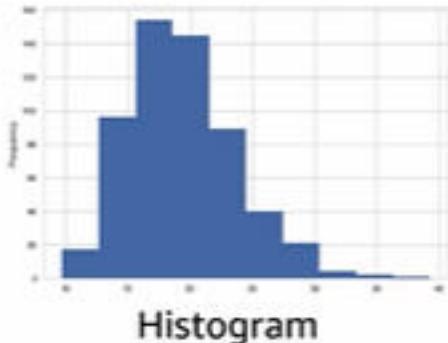
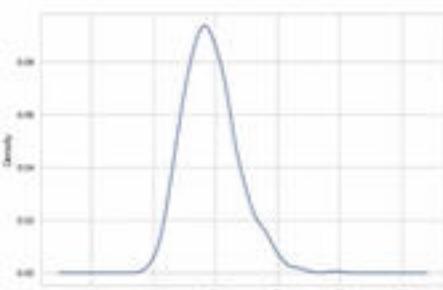
# show first a few rows
df.head()
# show datatype for each column
df.info()
# show summary statistics for each column
df.describe()
# check the target variable properties
df['target'].value_counts()
```

Basic Plots



Density Plot

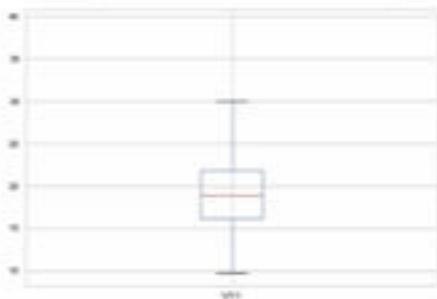
```
df['V11'].plot.kde()  
plt.show()
```



```
df['V11'].plot.hist()  
plt.show()
```

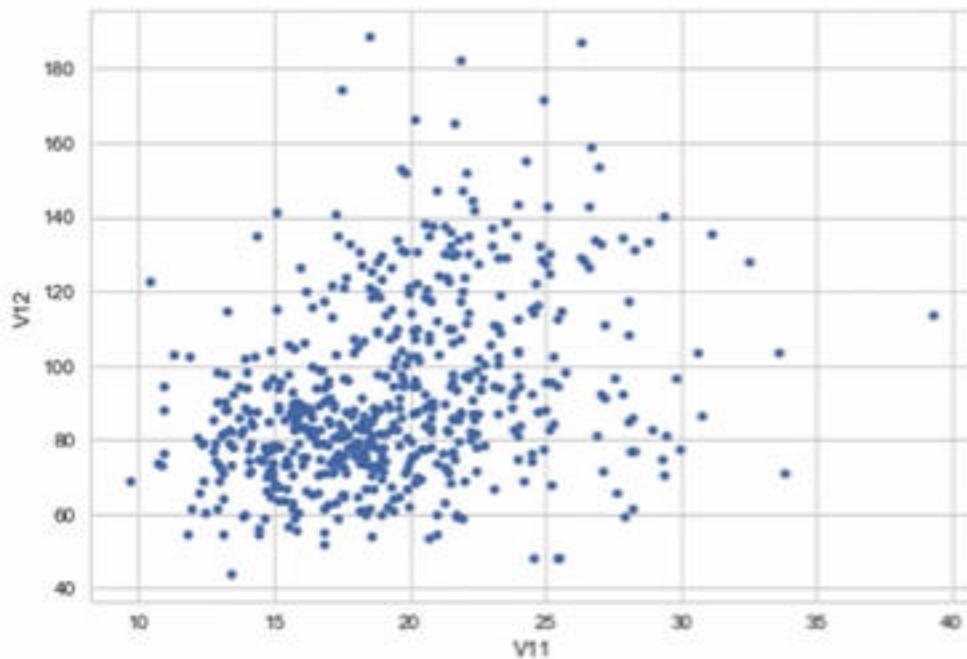
Box Plot

```
df.boxplot(['V11'])  
plt.show()
```

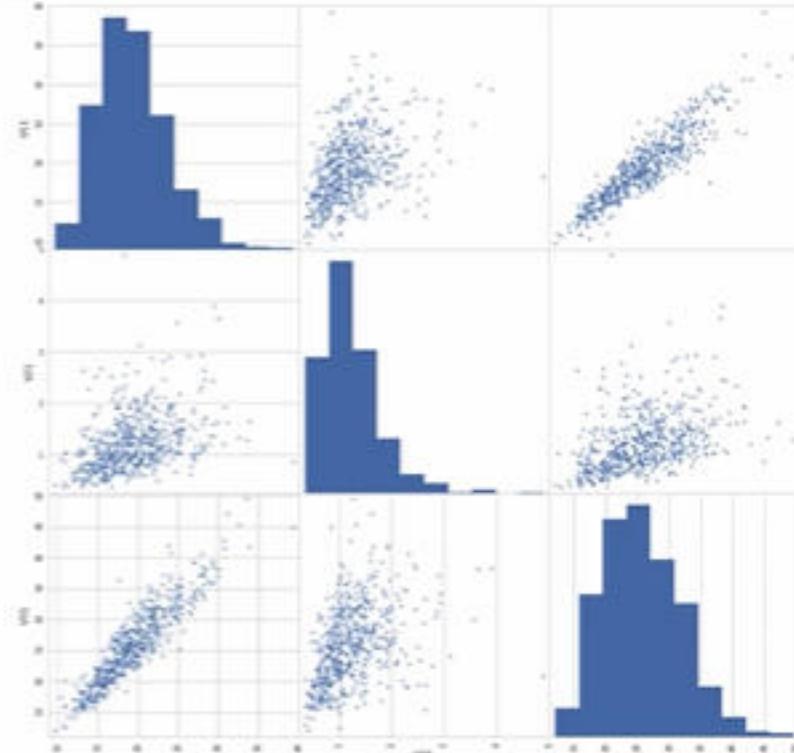


Scatterplot and Scatterplot Matrix

```
df.plot.scatter(x='V11', y='V12')  
plt.show()
```



```
pd.scatter_matrix(df[['V11', 'V21', 'V31']], figsize=(15, 15))
```



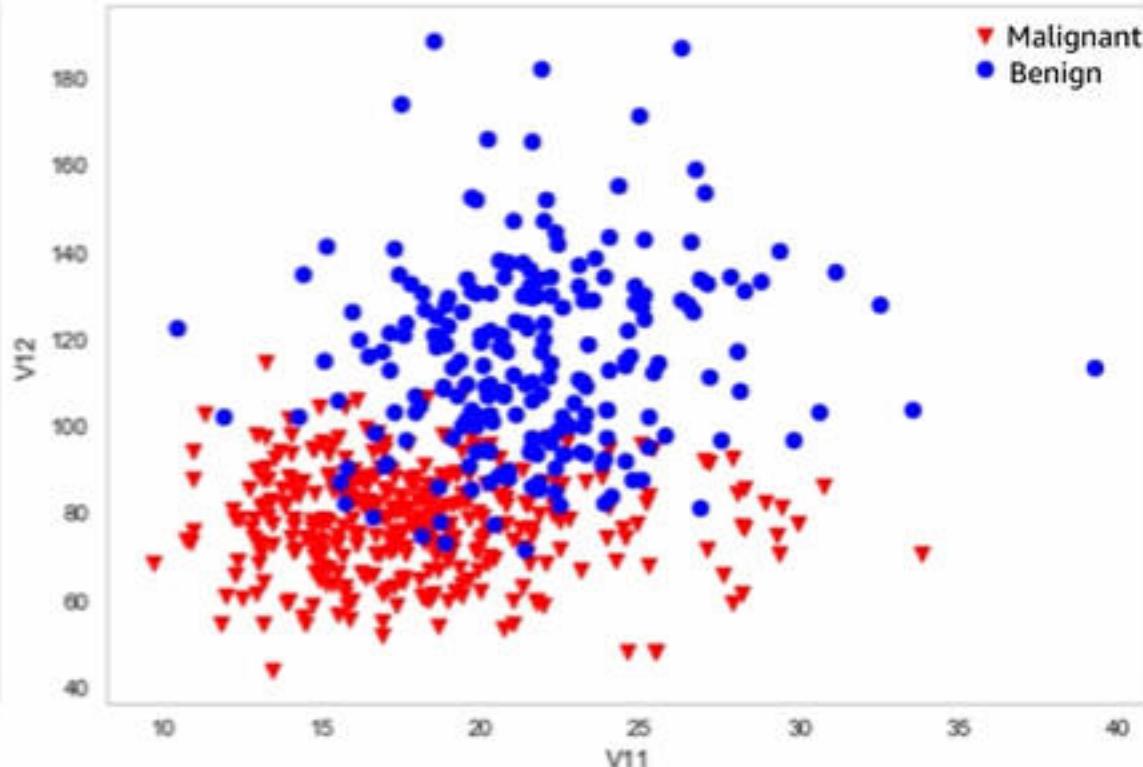
Scatterplot with Identification

```
malignant = df[['V11', 'V12']][df['target'] == 0]
benign = df[['V11', 'V12']][df['target'] == 1]

plt.scatter(benign['V11'],
            benign['V12'],
            s=50,
            c='red',
            marker='v',
            label='Malignant')

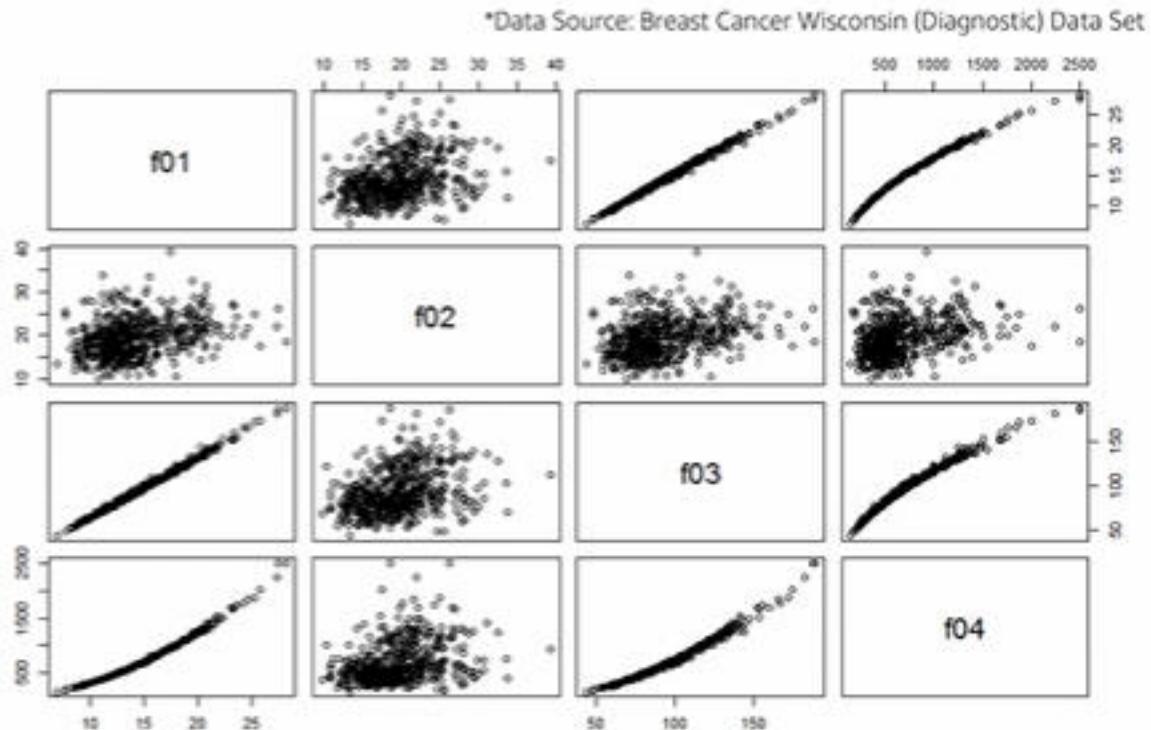
plt.scatter(malignant['V11'],
            malignant['V12'],
            s=50,
            c='blue',
            marker='o',
            label='Benign')

plt.xlabel('V11')
plt.ylabel('V12')
plt.legend()
plt.grid()
plt.show()
```



Scatterplot Matrix

Scatterplot matrices visualize attribute-target and attribute-attribute pairwise relationships



Correlation Matrix



*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

Correlation matrices measure the linear dependence between features; can be visualized with heatmaps

Example: First 10 features in breast cancer dataset

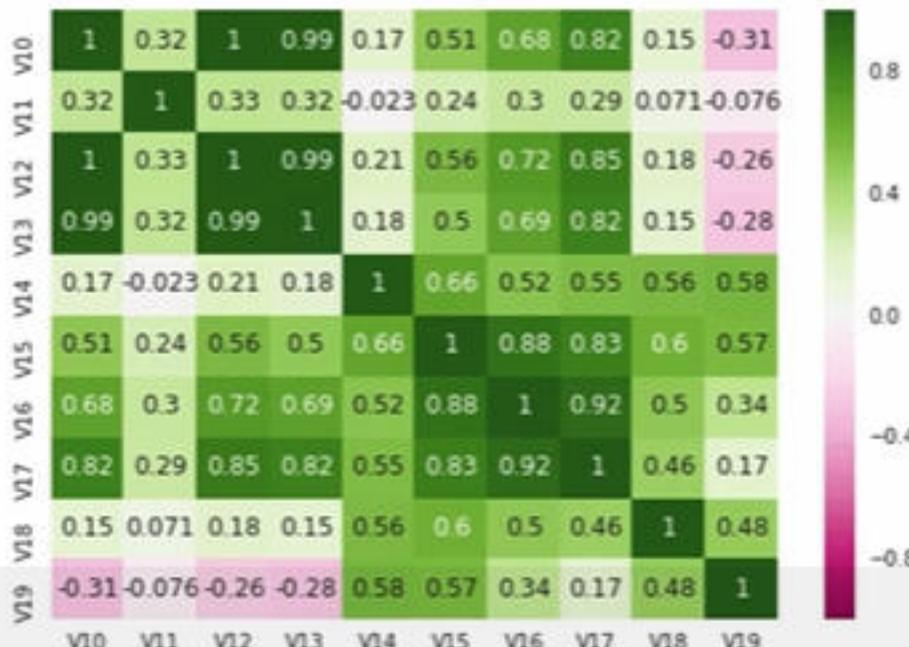
	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19
V10	1.00	0.32	1.00	0.99	0.17	0.51	0.68	0.82	0.15	-0.31
V11	0.32	1.00	0.33	0.32	-0.02	0.24	0.30	0.29	0.07	-0.08
V12	1.00	0.33	1.00	0.99	0.21	0.56	0.72	0.85	0.18	-0.26
V13	0.99	0.32	0.99	1.00	0.18	0.50	0.69	0.82	0.15	-0.28
V14	0.17	-0.02	0.21	0.18	1.00	0.66	0.52	0.55	0.56	0.58
V15	0.51	0.24	0.56	0.50	0.66	1.00	0.88	0.83	0.60	0.57
V16	0.68	0.30	0.72	0.69	0.52	0.88	1.00	0.92	0.50	0.34
V17	0.82	0.29	0.85	0.82	0.55	0.83	0.92	1.00	0.46	0.17
V18	0.15	0.07	0.18	0.15	0.56	0.60	0.50	0.46	1.00	0.48
V19	-0.31	-0.08	-0.26	-0.28	0.58	0.57	0.34	0.17	0.48	1.00



Correlation Matrix Heatmap Using Seaborn



```
import seaborn as sns
sns.heatmap(heatmaps, yticklabels=col, xticklabels=col, cmap='PiYG', annot=True)
```



*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

Correlation Matrix Heatmap



```
col = ['V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19']
heatmap = np.corrcoef(df[col].values.T)

fig, ax = plt.subplots(figsize=(15,15))
im = ax.imshow(heatmap, cmap='PiYG',vmin=-1)
fig.colorbar(im)
ax.grid(False)
[[ax.text(j, i, round(heatmap[i, j],2), ha="center", va="center", color="w")
  for j in range(len(heatmap))] for i in range(len(heatmap))]

ax.set_xticks(np.arange(len(col)))
ax.set_yticks(np.arange(len(col)))
ax.set_xticklabels(col)
ax.set_yticklabels(col)

plt.show()
```



Formulas for Correlations



Pearson Correlation

$$\rho_{xy} = \frac{\sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^N (x_i - \mu_x)^2 \sum_{i=1}^N (y_i - \mu_y)^2}}$$

Variance: $\sigma_x^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu_x)^2$

Covariance between (x, y):

$$\sigma_{xy} = \frac{1}{N} \sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)$$

Correlation between (x, y):

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

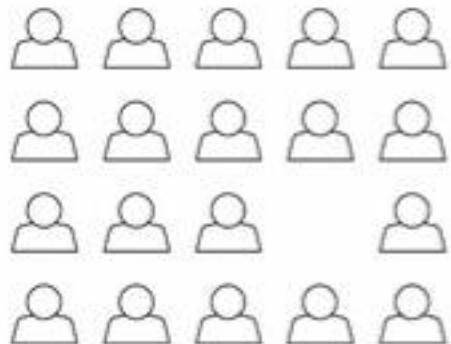


0:06

2x



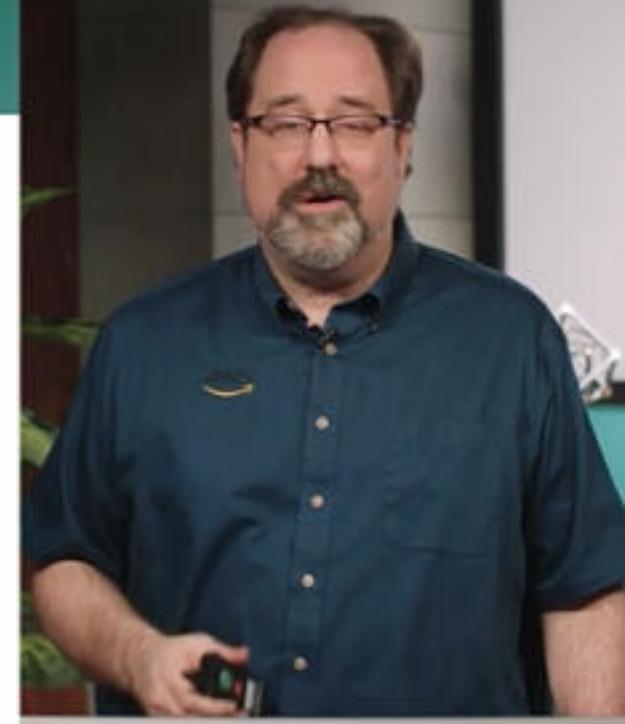
Data Issues



Missing data



Noisy data



4:43

2x



Topics

- Data Issues
 - Messy data
 - Noisy data
 - Biased data
 - Imbalanced data
 - Correlated data



-5:34

2x



Data Issues

ID	Survey Response
1	This is grrreat!
2	This is grrreat!
3	Y E s
4	ur \$0 L33t!
5	¿Qué?
6	或者
7	احب ذلك

Messy data

ID	Length
1	40 km
2	100 m
3	100 mi
4	74 m
5	74 ft
6	29 in
7	1092 nm

Data on different scales

ID	Measurement
1	96 km
2	twenty
3	5:40:27
4	735 cm ³
5	2 cats
6	44 °C
7	346 Mb/s

Mixed type data

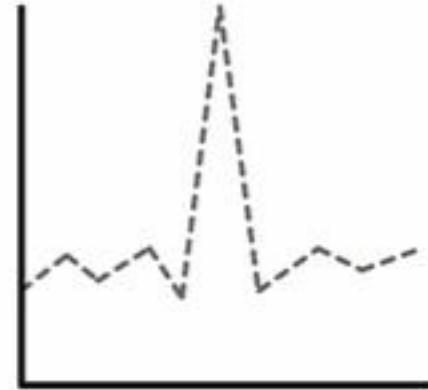
Data Issues



Imbalanced data



Sample bias



Outliers



-1:29

2x



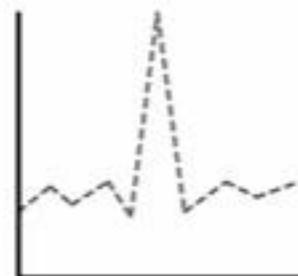
Data Issues



Imbalanced data



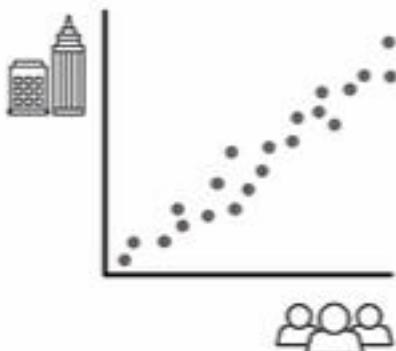
Sample bias



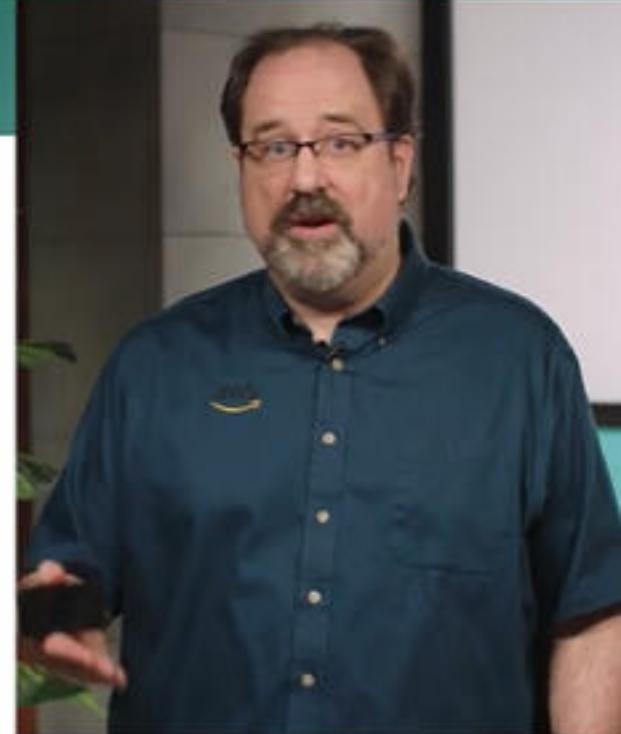
Outliers



Data Issues



Highly correlated features can cause collinearity problems and numerical instability



Data Preprocessing: Handling Missing Values



- Sources of missing values
- Processing/Dropping/Imputing missing values



-0:49

2x



Feature Engineering: Scaling



- Aligning scales
- Types of scaling



-0:19

2x



Feature Engineering: Polynomial Transformation



- Considerations
- Radial basis function





||

0:02

2x



Feature Engineering: Text-based Features



- Types of vectorizing



-0:08

2x



Processing Categoricals



Categorical (also called *discrete*)

Attribute that takes finite set of values

Examples:

`color` $\in \{\text{green, red, blue}\}$



`isFraud` $\in \{\text{false, true}\}$



Categorical Types

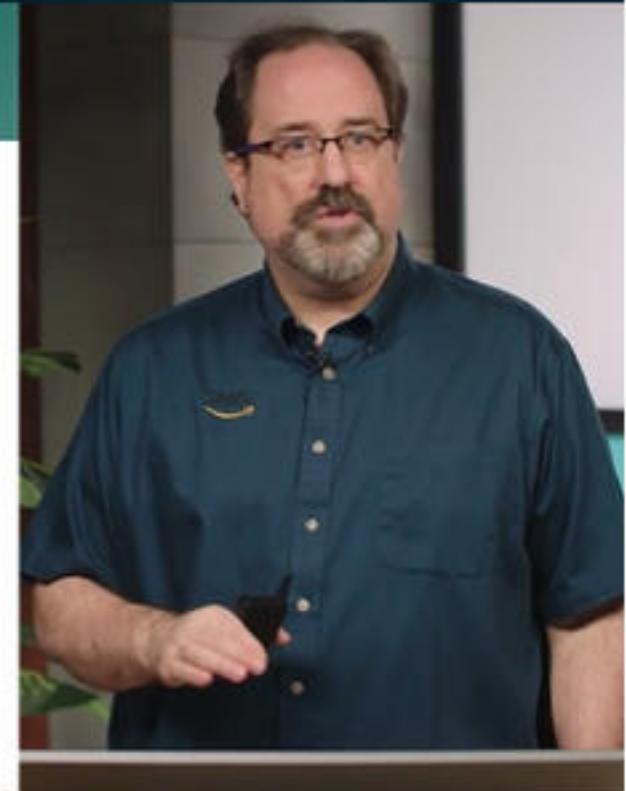
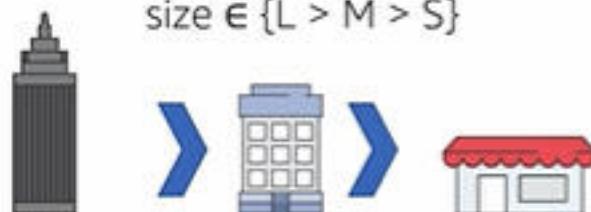


Ordinal

Categories are ordered

Example:

$\text{size} \in \{\text{L} > \text{M} > \text{S}\}$



Categorical Types



Ordinal

Categories are ordered

Example:

$\text{size} \in \{\text{L} > \text{M} > \text{S}\}$



Nominal

Categories are unordered

Example:

color



4:34

2x



Processing Categoricals



Issue

Categoricals are often represented as text
but many algorithms require numericals as input.

We need special encoding to convert
categoricals into numerical representations.

Loan Approval Example

Sample dataset based on house prices and loans approved:

```
import pandas as pd
df = pd.DataFrame([['house',3,2572,'S',1372000,'Y'],
                   ['apartment',2,1386,'N',699000,'N'],
                   ['house',3,1932,'L',800000,'N'],
                   ['house',1,851,'M',451000,'Y'],
                   ['apartment',1,600,'N',325000,'N']])
df.columns = ['type', 'bedrooms', 'area', 'garden_size', 'price', 'loan_approved']
df
```

	type	bedrooms	area	garden_size	price	loan_approved
0	house	3	2572	S	1372000	Y
1	apartment	2	1386	N	699000	N
2	house	3	1932	L	800000	N
3	house	1	851	M	451000	Y
4	apartment	1	600	N	325000	N



Encoding Ordinals



When mapping feature variables to a predefined map, use the **map** function in pandas. (You can also use "inplace = True")

```
mapping = dict({'N':0, 'S':5, 'M':10, 'L':20})
df['num_garden_size'] = df['garden_size'].map(mapping)
df
```

	type	bedrooms	area	garden_size	price	loan_approved	num_garden_size
0	house	3	2572	S	1372000	Y	5
1	apartment	2	1386	N	699000	N	0
2	house	3	1932	L	800000	N	20
3	house	1	851	M	451000	Y	10
4	apartment	1	600	N	325000	N	0



Encoding Nominals



Using `Sklearn.preprocessing.OneHotEncoder`

```
from sklearn.preprocessing import OneHotEncoder

df = pd.DataFrame({"Fruits":["Apple","Banana","Banana","Mango","Banana"]})

type_labelenc = LabelEncoder()
num_type = group_enc.fit_transform(df["Fruits"])
print(num_type)
type_enc = OneHotEncoder()
type_enc.fit(num_type.reshape(-1,1))
print(type_enc.transform(num_type.reshape(-1,1)).toarray())
```

```
[0 1 1 2 1]
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 0.  1.  0.]]
```

Encoding Nominals



Using `Sklearn.preprocessing.OneHotEncoder`

```
from sklearn.preprocessing import OneHotEncoder

df = pd.DataFrame({"Fruits":["Apple","Banana","Banana","Mango","Banana"]})

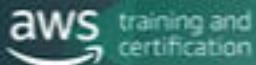
type_labelenc = LabelEncoder()
num_type = type_labelenc.fit_transform(df["Fruits"])
print(num_type)

type_enc = OneHotEncoder()
type_enc.fit(num_type.reshape(-1,1))
print(type_enc.transform(num_type.reshape(-1,1)).toarray())
```

```
[0 1 1 2 1]
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 0.  1.  0.]]
```



Encoding Nominals



Using `pandas.get_dummies()`

```
import pandas as pd

df = pd.DataFrame({"Fruits":["Apple","Banana","Banana","Mango","Banana"]})

pd.get_dummies(df)
```

	Fruits_Apple	Fruits_Banana	Fruits_Mango
0	1	0	0
1	0	1	0
2	0	1	0
3	0	0	1
4	0	1	0



Dropping Missing Values

Default drops the rows with NULL values

```
df.dropna()
```

	Fruits	Number
0	Apple	5.0
2	Banana	3.0
4	Banana	1.0

"axis=1" drops the columns with NULL values

```
df.dropna(axis=1)
```

	Fruits
0	Apple
1	Banana
2	Banana
3	Mango
4	Banana

More complicated dropna () rules:

- df.dropna(how='all')
- df.dropna(thresh=4)
- df.dropna(subset=['Fruits'])



-6:04

2x



Risk of dropping columns

- May lose information in features (underfitting)

Dropping Missing Values



Before dropping or imputing (replacing) missing values, ask:

- What were the **mechanisms** that caused the missing values?
- Are these missing values **missing at random**?
- Are there rows or columns missing that you are **not aware** of?



Imputing Missing Values

Evaluate the **cause** of missingness to determine the best imputation algorithm.

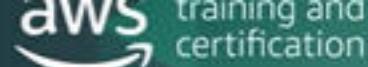
```
from sklearn.preprocessing import Imputer
import numpy as np

arr = np.array([[5,3,2,2],[3,None,1,9],[5,2,7,None]])

imputer = Imputer(strategy='mean')
imp = imputer.fit(arr)
imputer.transform(arr)

array([[ 5. ,  3. ,  2. ,  2. ],
       [ 3. ,  2.5,  1. ,  9. ],
       [ 5. ,  2. ,  7. ,  5.5]])
```

Advanced Methods for Imputing Missing Values



- MICE (Multiple Imputation by Chained Equations):
`sklearn.impute.MICEImputer` (v0.20)
- Python (not sklearn) `fancyimpute` package
 - KNN impute
 - SoftImpute
 - MICE
 - etc

Creating novel features to use as inputs for ML models using domain and data knowledge

scikit-learn: [sklearn.feature_extraction](#)



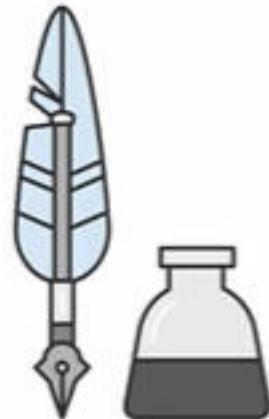
-3:08

2x



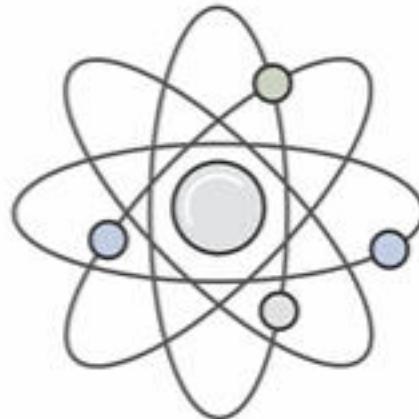
is often more

art



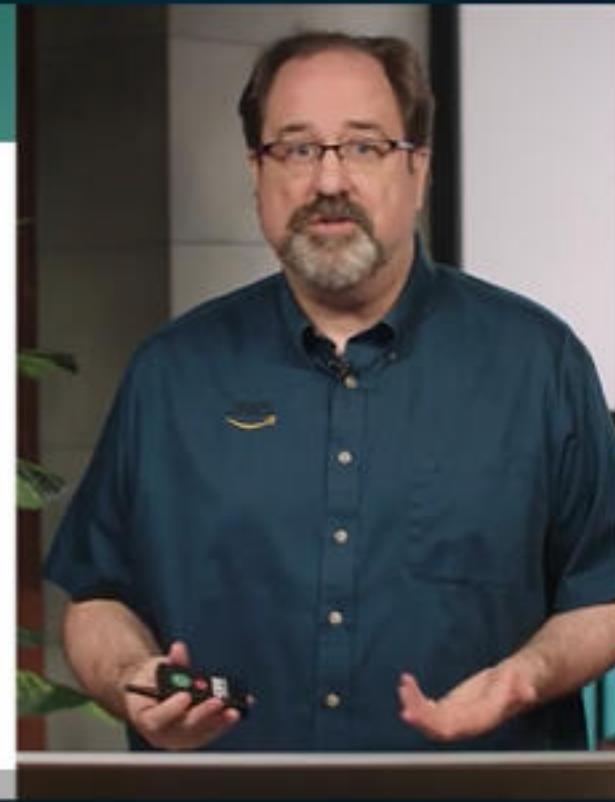
than

science



Some rules of thumb:

Try generating many features first
then
apply dimensionality reduction if needed



Some rules of thumb:

Consider **transformations** of attributes

$$X^2$$

Example: squaring



0:54

2x





Motivation risk:

Many algorithms are sensitive to features being on **different scales**, e.g., gradient descent and kNN.

Type	Bedrooms	Area (sq. ft)	Garden Size	Price	Loan Approved
House	3	2572	Small	1372000	Yes
Apartment	2	1386	N/A	699000	No
House	3	1932	Large	800000	No
House	1	851	Medium	451000	Yes
Apartment	1	600	N/A	325000	No



Different scales

Some algorithms, like decision trees and random forests, aren't sensitive to features on different scales.

Important

Fit the scaler to training data only, then transform both train and validation data.



-5:38

2x



Scaling Transformation



Common choices in sklearn

- Mean/variance standardization
- MinMax scaling
- Maxabs scaling
- Robust scaling
- Normalizer



Mean/Variance Standardization



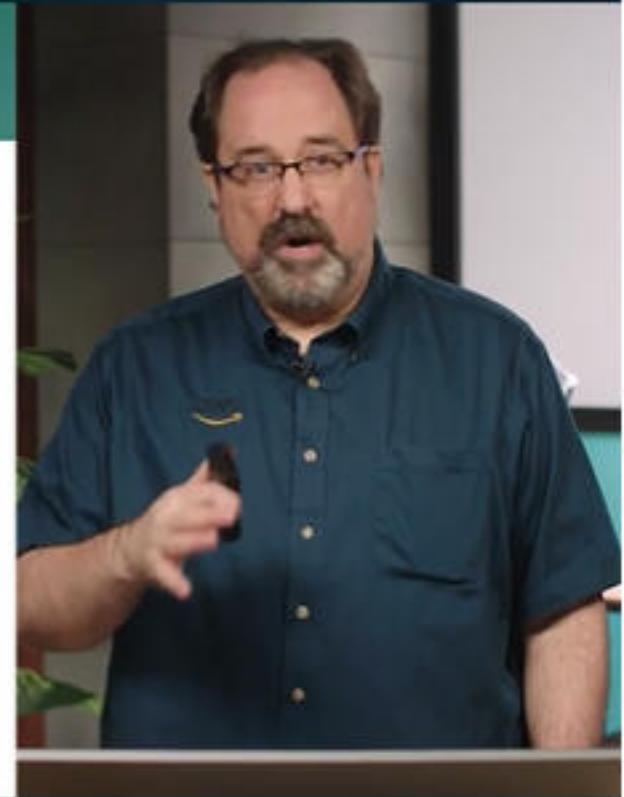
Transform: $x_{i,j}^* = \frac{x_{i,j} - \mu_j}{\sigma_j}$

Scaled values are centered around

mean $\mu_j = 0$

with standard deviation $\sigma_j = 1$ for each data column

scikit-learn: `sklearn.preprocessing.StandardScaler`



Mean/Variance Standardization

Transform: $x_{i,j}^* = \frac{x_{i,j} - \mu_j}{\sigma_j}$

Advantages:

- Many algorithms behave better with smaller values
- Keeps outlier information, but reduces impact

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
arr = np.array([[5,3,2,2],[2,3,1,9],[5,2,7,6]],dtype=float)
print(scale.fit_transform(arr))
print(scale.scale_)

[[ 0.70710678  0.70710678 -0.50800051 -1.27872403]
 [-1.41421356  0.70710678 -0.88900089  1.16247639]
 [ 0.70710678 -1.41421356  1.3970014   0.11624764]]
[ 1.41421356  0.47140452  2.62466929  2.86744176]
```

Mean/Variance Standardization

Transform: $x_{i,j}^* = \frac{x_{i,j} - \mu_j}{\sigma_j}$

Advantages:

- Many algorithms behave better with smaller values
- Keeps outlier information, but reduces impact

```
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
arr = np.array([[5,3,2,2],[2,3,1,9],[5,2,7,6]],dtype=float)
print(scale.fit_transform(arr))
print(scale.scale_)

[[ 0.70710678  0.70710678 -0.50800051 -1.27872403]
 [-1.41421356  0.70710678 -0.88900089  1.16247639]
 [ 0.70710678 -1.41421356  1.3970014   0.11624764]]
[ 1.41421356  0.47140452  2.62466929  2.86744176]
```

MinMax Scaling

Transform: $x_{i,j}^* = \frac{x_i - \min x_j}{\max x_j - \min x_j}$

Scale values so that:

minimum = 0

maximum = 1

scikit-learn: `sklearn.preprocessing.MinMaxScaler`

MinMax Scaling

Transform: $x_{i,j}^* = \frac{x_i - \min x_j}{\max x_j - \min x_j}$

Advantages:

- Robust to small standard deviations

```
from sklearn.preprocessing import MinMaxScaler  
  
scale = MinMaxScaler()  
arr = np.array([[5,3,2,2],[2,3,1,9],[5,2,7,6]],dtype=float)  
print(scale.fit_transform(arr))  
print(scale.scale_)  
print(scale.min_)
```

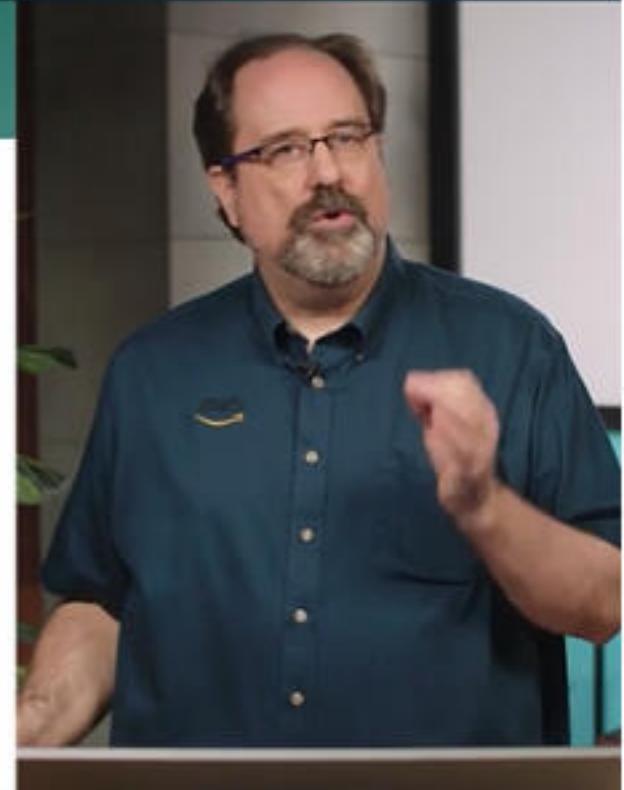
[[1.	1.	0.16666667	0.]
[0.	1.	0.	1.]
[1.	0.	1.	0.57142857]]
[0.33333333	1.	0.16666667	0.14285714]	
[-0.66666667	-2.	-0.16666667	-0.28571429]	

Robust Scaling



Transform: $x_i^* = \frac{x_i - Q_{25}(x)}{Q_{75}(x) - Q_{25}(x)}$

scikit-learn: `sklearn.preprocessing.RobustScaler`



Robust Scaling

Transform: $x_i^* = \frac{x_i - Q_{25}(x)}{Q_{75}(x) - Q_{25}(x)}$

scikit-learn: `sklearn.preprocessing.RobustScaler`

$$\text{Transform: } x_{i,j}^* = \frac{x_{i,j}}{\sigma_j}$$

Scaled values are scaled
with standard deviation $\sigma_j = 1$

scikit-learn: `sklearn.preprocessing.Normalizer`



-0:33

2x



Normalizer



Transform: $x_{i,j}^* = \frac{x_{i,j}}{\sigma_j}$

Scaled values are scaled
with standard deviation $\sigma_j = 1$

scikit-learn: `sklearn.preprocessing.Normalizer`

Normalizer

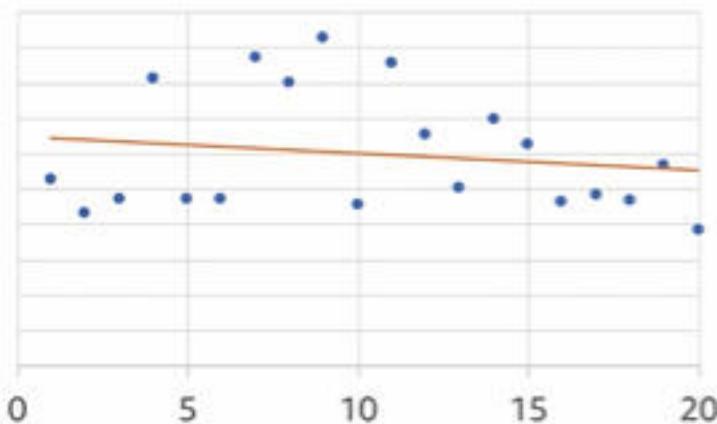
Transform: $x_{i,j} = \frac{x_j - \mu_x}{\sigma_x}$

Rescales x_j to unit norm based on

- L1 norm
- L2 norm
- Max norm

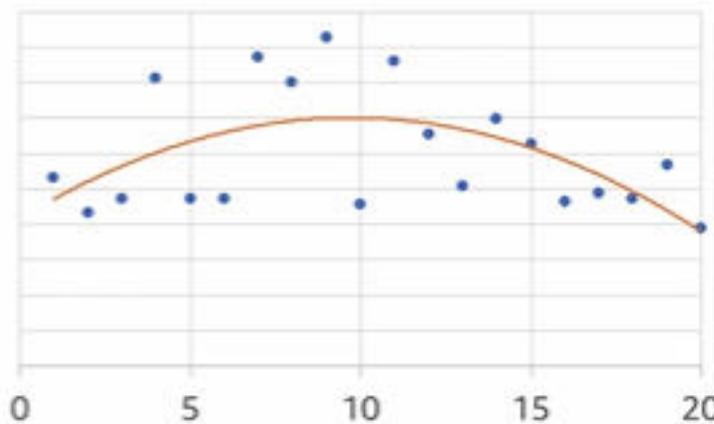
Polynomial Transformation

Sometimes a polynomial relationship with features is a better fit.



Linear fit
 $R^2=0.0283$

$$y = a * x + b$$



Quadratic fit
 $R^2=0.3201$

$$y = a_2 * x^2 + a_1 * x + c$$

Polynomial Transformation



Solution

Apply polynomial transformation to feature

$$y = ax_1 + bx_2 + c$$



$$y = a_1x_1 + a_2x_1^2 + a_3x_1^3 + b_1x_2 + b_2x_2^2 + b_3x_2^3$$



Polynomial Transformation



scikit-learn

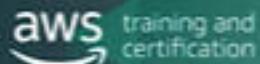
sklearn.preprocessing.PolynomialFeatures

```
from sklearn.preprocessing import PolynomialFeatures

df = pd.DataFrame({'a':np.random.rand(5), 'b':np.random.rand(5)})
cube = PolynomialFeatures(degree=3)
cube_features = quadratic.fit_transform(df)
df_cube = pd.DataFrame(cube_features,columns=cols)
df_cube
```

	ones	a	b	a^2	ab	b^2	a^3	ba^2	ab^2	b^3
0	1.0	0.221993	0.611744	0.049281	0.135803	0.374231	0.010940	0.030147	0.083077	0.228933
1	1.0	0.870732	0.765908	0.758175	0.666901	0.586615	0.660167	0.580692	0.510784	0.449293
2	1.0	0.206719	0.518418	0.042733	0.107167	0.268757	0.008834	0.022153	0.055557	0.139329
3	1.0	0.918611	0.296801	0.843846	0.272644	0.088091	0.775166	0.250454	0.080921	0.026145
4	1.0	0.488411	0.187721	0.238545	0.091685	0.035239	0.116508	0.044780	0.017211	0.006615

Polynomial Transformation



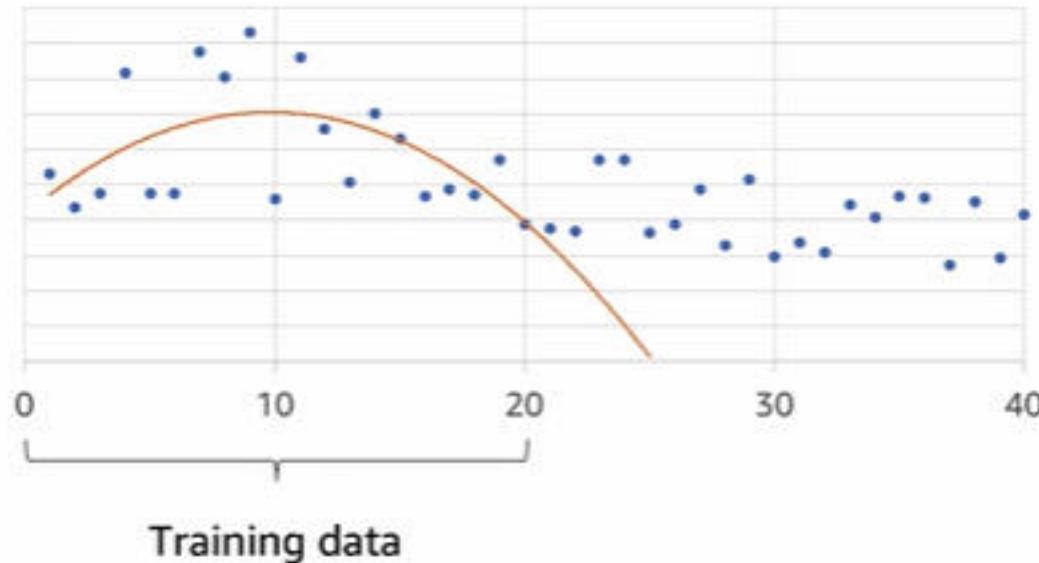
Considerations

- Beware of **overfitting** if the degree is too high.
- Consider non-polynomial transformations as well.
- For example:
 - Log transforms
 - Sigmoid transforms



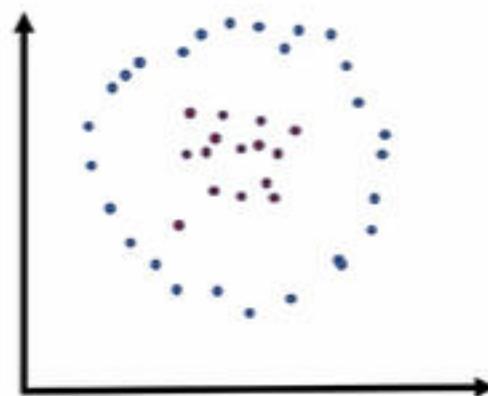
Polynomial Transformation

Risk of **extrapolation beyond the range of the data** when using polynomial transformations



Radial Basis Function

- Transform: $f(x) = f(\|x - c\|)$
- Widely used in Support Vector Machine as a kernel and in Radial Basis Neural Networks (RBNNs)
- Gaussian RBF is the most common RBF used



Bag-of-words model

Represent document as **vector of numbers**, one for each word
(tokenize, count, and normalize)

Note:

- **Sparse matrix implementation** is typically used, ignores relative position of words
- Can be extended to **bag of n-grams** of words or of characters



-1:46

2x





||



0:22

2x



Model Debugging: Validation Set



- The importance of validation sets
- How validation, training, and test sets work together

Model Debugging: Error Analysis



- Looking for patterns in failed predictions
- Common patterns

Model Tuning: Regularization



- Regularization in linear models
- Regularization for regression

Model Tuning: Hyperparameter Tuning



- Defining hyperparameters
- Techniques

Model Tuning: Feature Set Tuning



- Training data basics
- Dimensionality reduction
- Feature extraction techniques

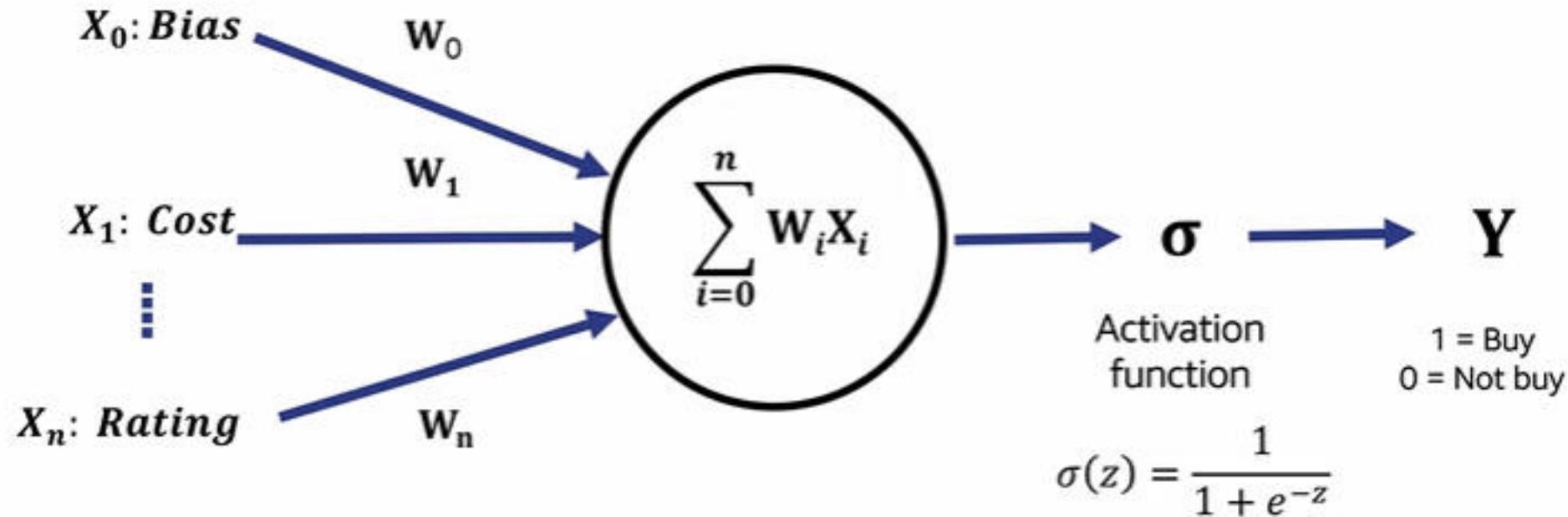
Model Tuning: Feature Selection



- The importance of filtering and selecting your features
- Random forest feature importance



Perceptron

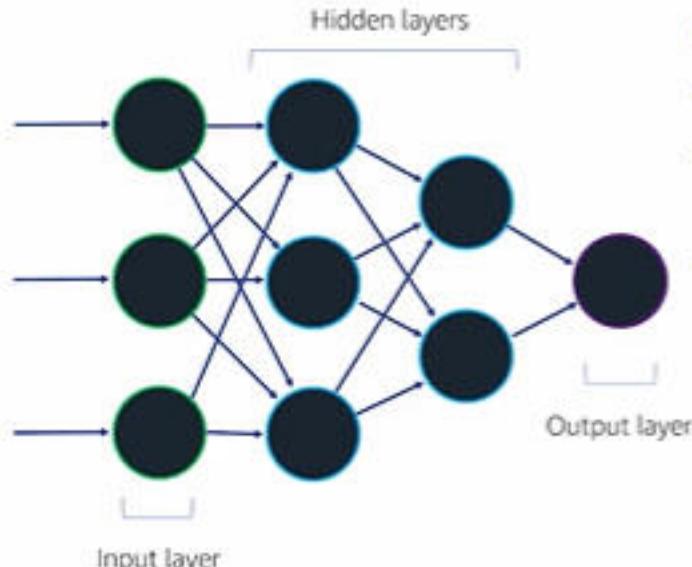
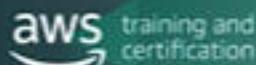


-7:09

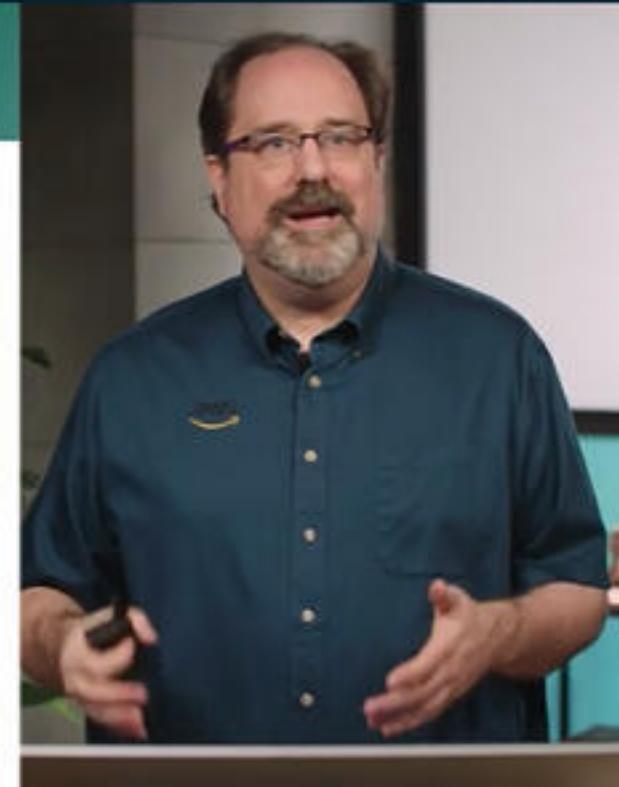
2x



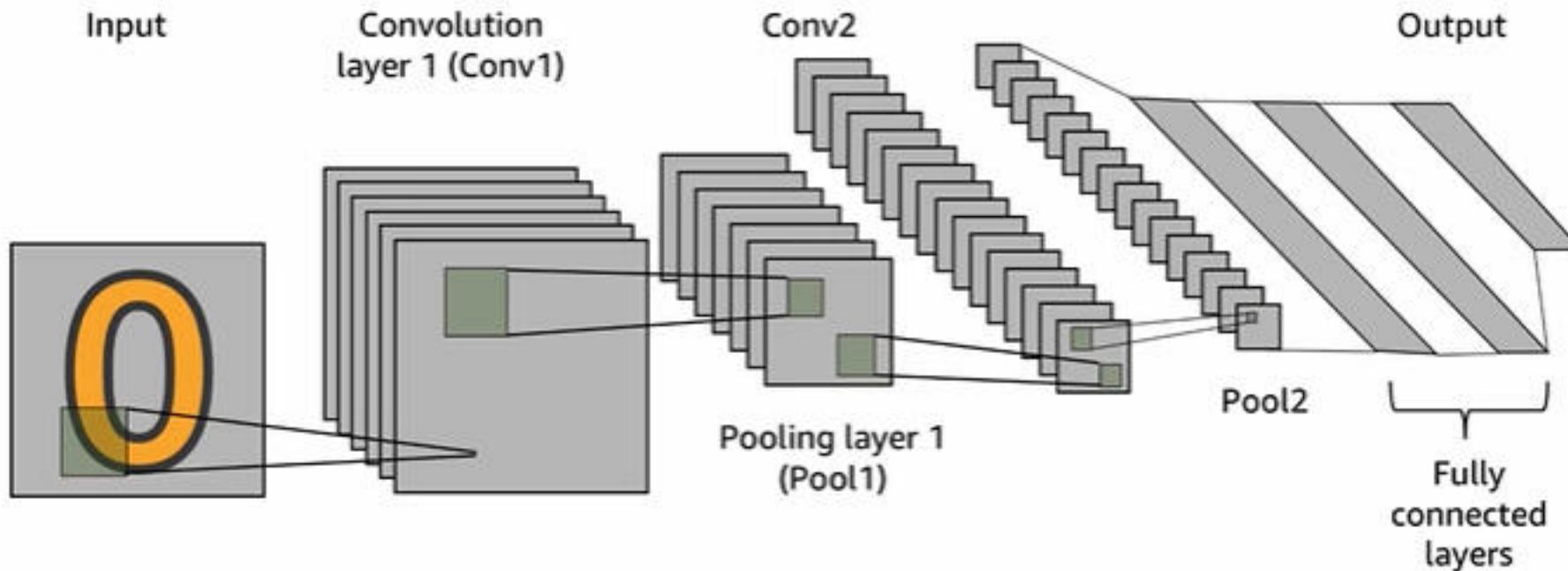
Neural networks



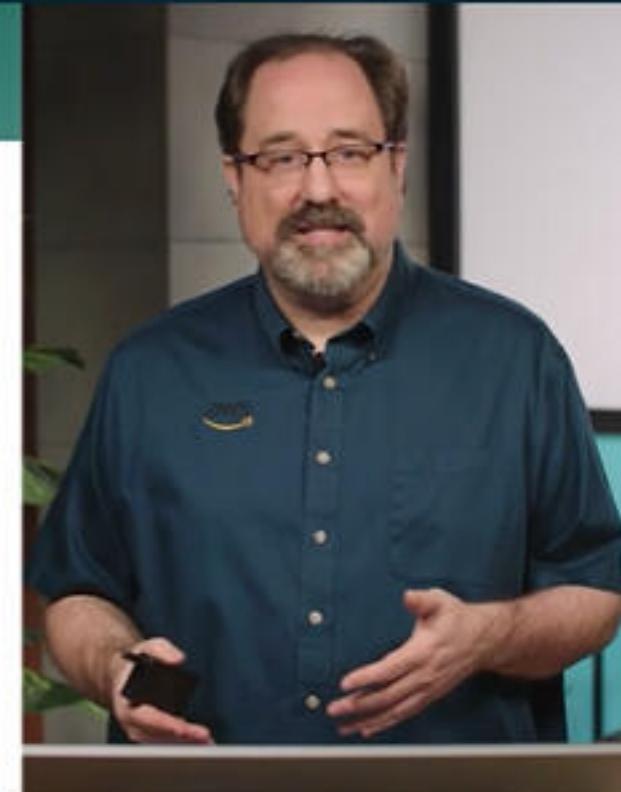
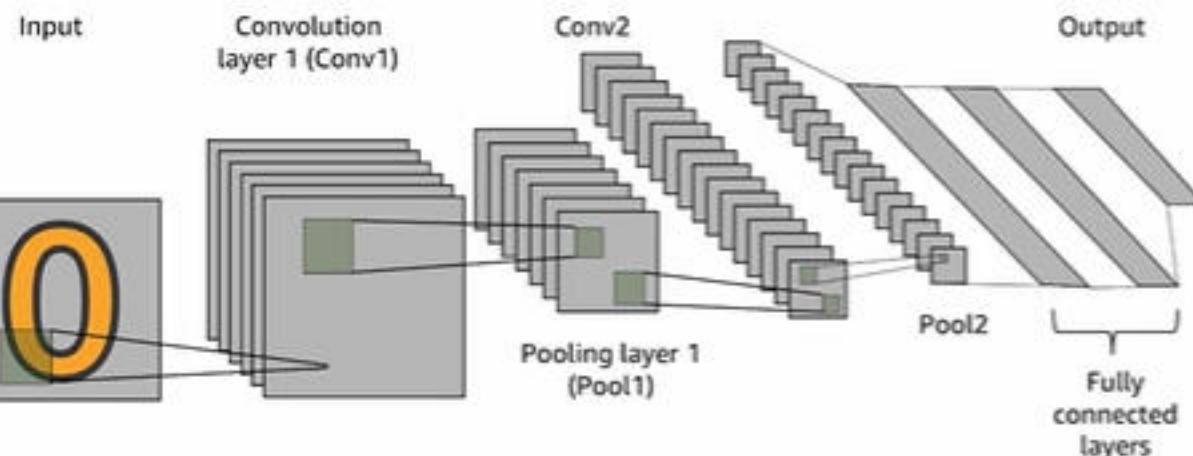
- Generally hard to interpret
- Expensive to train, fast to predict
- scikit-learn:
`sklearn.neural_network.MLPClassifier`
- Deep learning frameworks:
 - MXNet
 - TensorFlow
 - Caffe
 - PyTorch



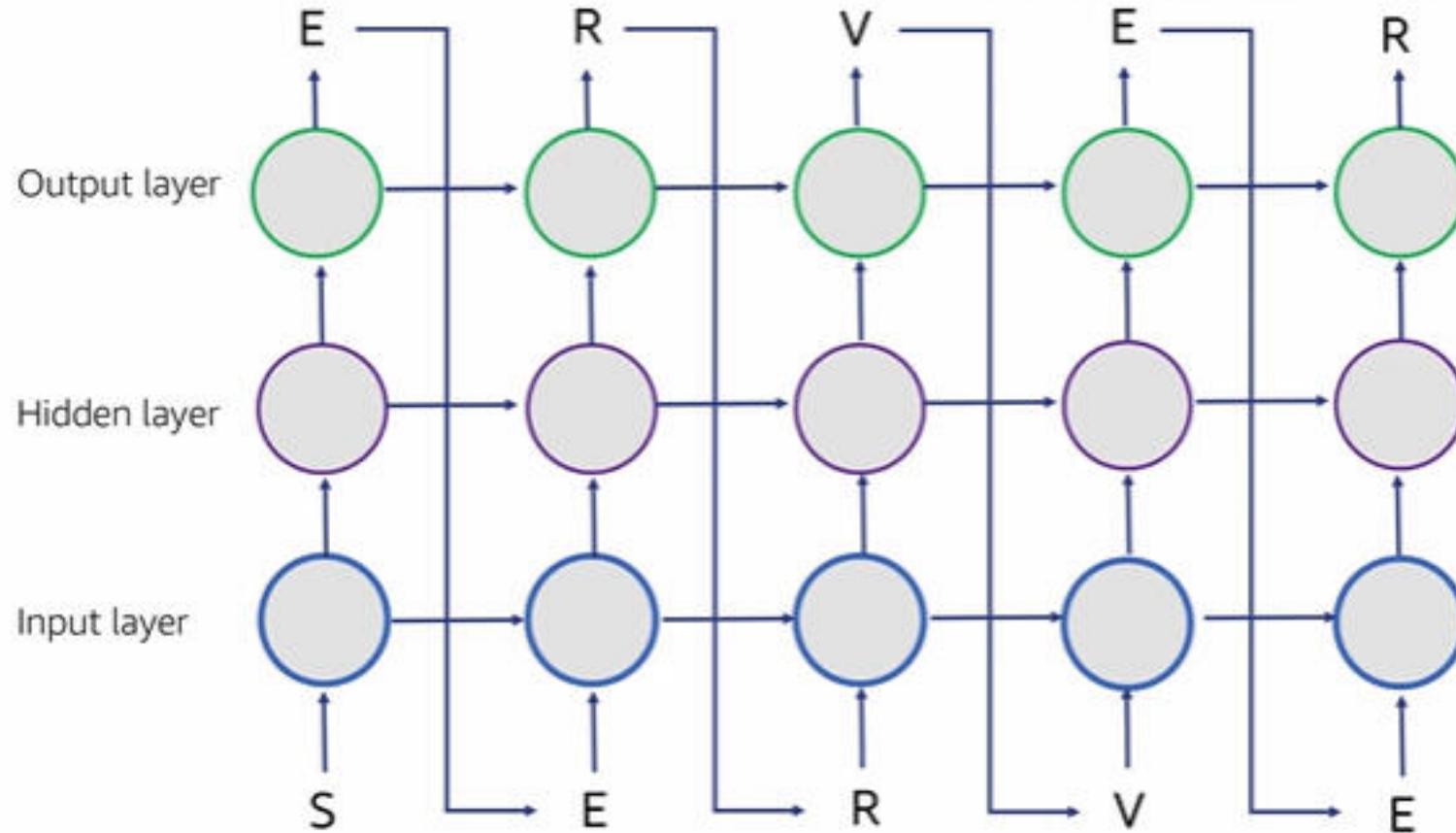
Convolutional neural networks



Convolutional neural networks

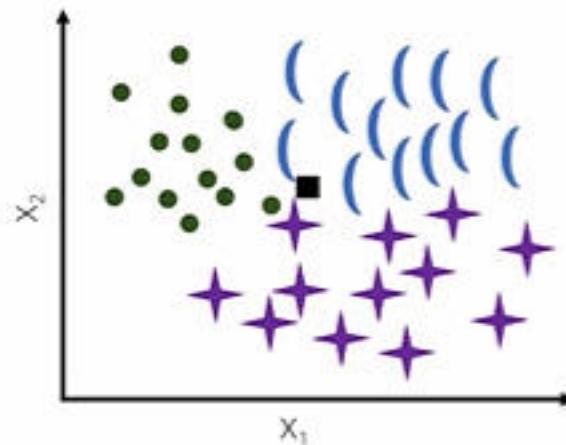


Recurrent neural networks



K-Nearest Neighbors

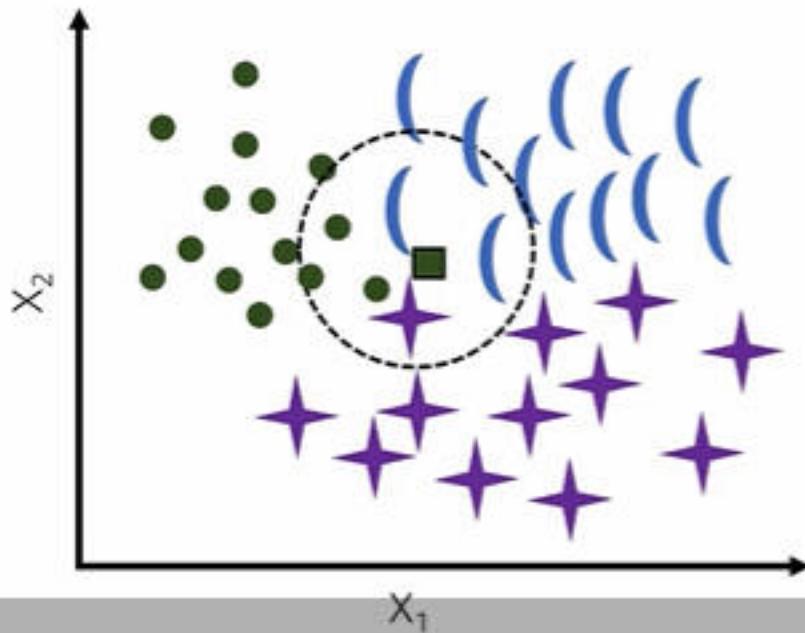
aws training and certification



- Define a distance metric
 - Euclidean distance
 - Manhattan distance
 - Any vector norm
- Choose the number of **k** neighbors
- Find the **k** nearest neighbors of the new observation that we want to classify



K-Nearest Neighbors



- Assign the class label by majority vote
- Important to find the right **k**
 - Commonly use $k = \frac{\sqrt{N}}{2}$ where N = number of samples

K-Nearest Neighbors



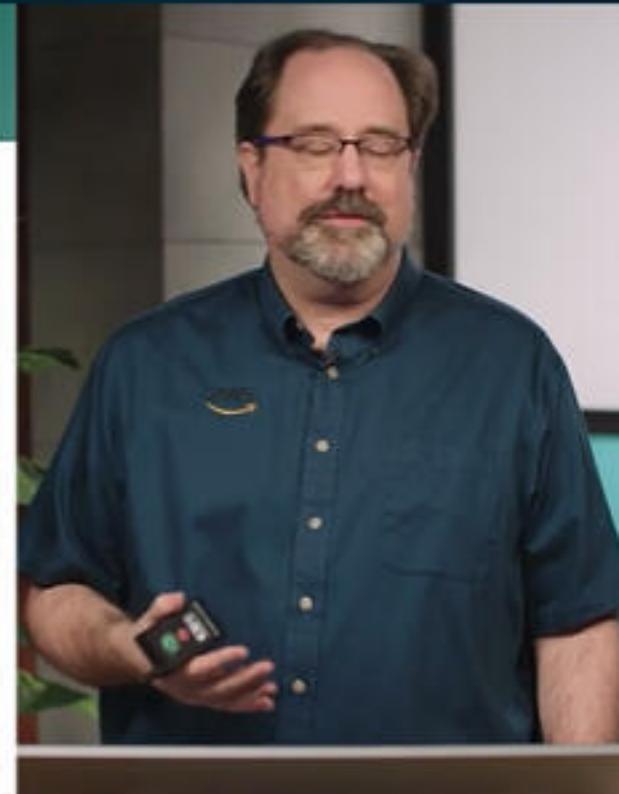
- Non-parametric, instance-based, lazy
 - Non-parametric: Model is not defined by fixed set of parameters
 - Instance-based or lazy learning: Model is the result of effectively memorizing training data
- Requires keeping the original data set
- Space complexity and prediction-time complexity grow with size of training data
- Suffers from curse of dimensionality: points become increasingly isolated with more dimensions, for a fixed-size training dataset
- scikit-learn: `sklearn.neighbors.KNeighborsClassifier`



K-Nearest Neighbors



- Non-parametric, instance-based, lazy
 - Non-parametric: Model is not defined by fixed set of parameters
 - Instance-based or lazy learning: Model is the result of effectively memorizing training data
- Requires keeping the original data set
- Space complexity and prediction-time complexity grow with size of training data
- Suffers from curse of dimensionality: points become increasingly isolated with more dimensions, for a fixed-size training dataset
- scikit-learn: `sklearn.neighbors.KNeighborsClassifier`



K-Nearest Neighbors



- Non-parametric, instance-based, lazy
 - Non-parametric: Model is not defined by fixed set of parameters
 - Instance-based or lazy learning: Model is the result of effectively memorizing training data
- Requires keeping the original data set
- Space complexity and prediction-time complexity grow with size of training data
- Suffers from curse of dimensionality: points become increasingly isolated with more dimensions, for a fixed-size training dataset
- scikit-learn: `sklearn.neighbors.KNeighborsClassifier`

K-Nearest Neighbors



- Non-parametric, instance-based, lazy
 - Non-parametric: Model is not defined by fixed set of parameters
 - Instance-based or lazy learning: Model is the result of effectively memorizing training data
- Requires keeping the original data set
- Space complexity and prediction-time complexity grow with size of training data
- Suffers from curse of dimensionality: points become increasingly isolated with more dimensions, for a fixed-size training dataset
- scikit-learn: `sklearn.neighbors.KNeighborsClassifier`

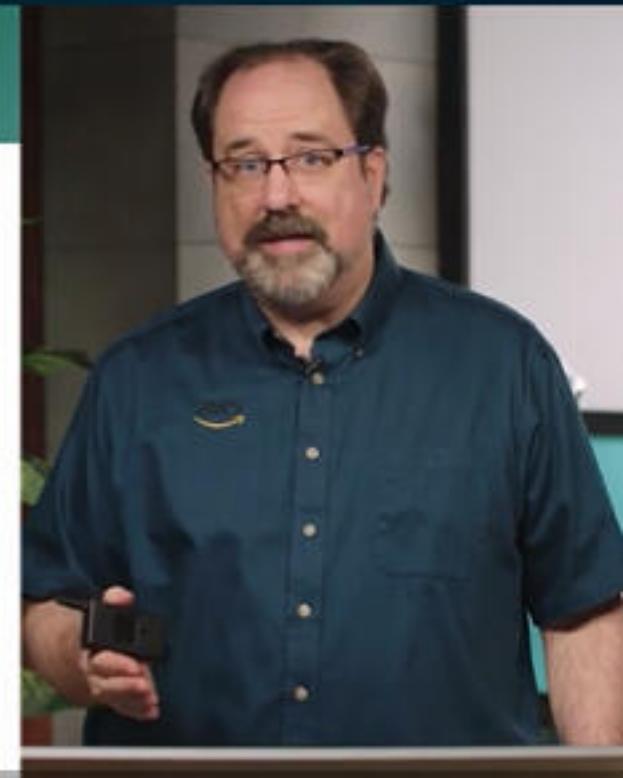
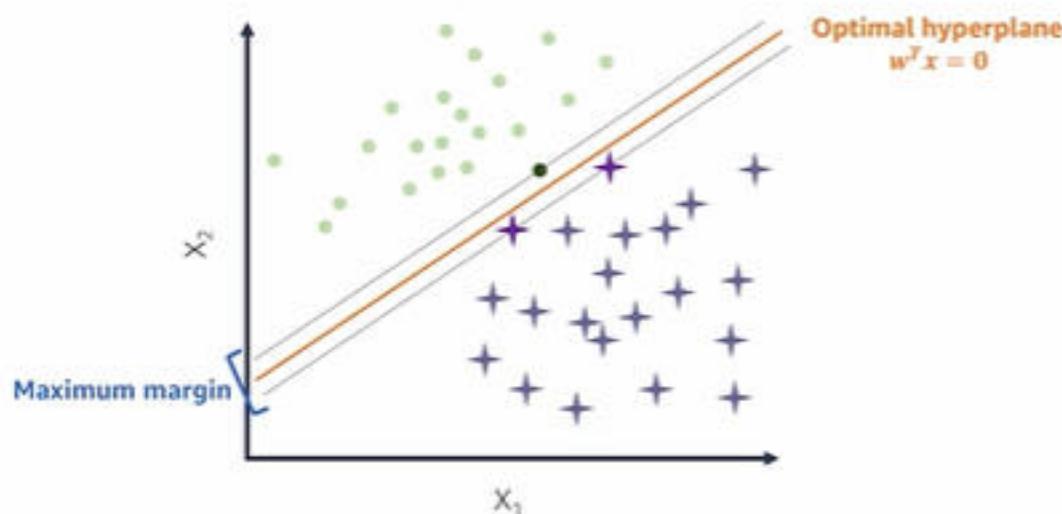


K-Nearest Neighbors



- Non-parametric, instance-based, lazy
 - Non-parametric: Model is not defined by fixed set of parameters
 - Instance-based or lazy learning: Model is the result of effectively memorizing training data
- Requires keeping the original data set
- Space complexity and prediction-time complexity grow with size of training data
- Suffers from curse of dimensionality: points become increasingly isolated with more dimensions, for a fixed-size training dataset
- scikit-learn: `sklearn.neighbors.KNeighborsClassifier`

Linear support vector machines



-3:03

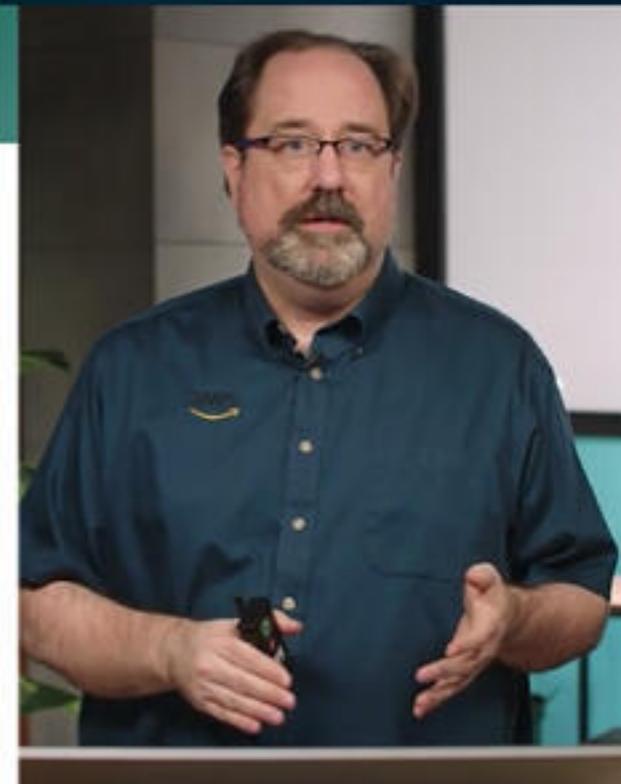
2x



Linear support vector machines



- Popular approach in research, but not in the industry
- Simplest case: Maximize the margin – the distance between the decision boundary (hyperplane) and the *support vectors* (training examples closest to the boundary)
- Max margin picture not applicable in non-separable case
- scikit-learn: sklearn.svm.SVC



Non-linear support vector machines



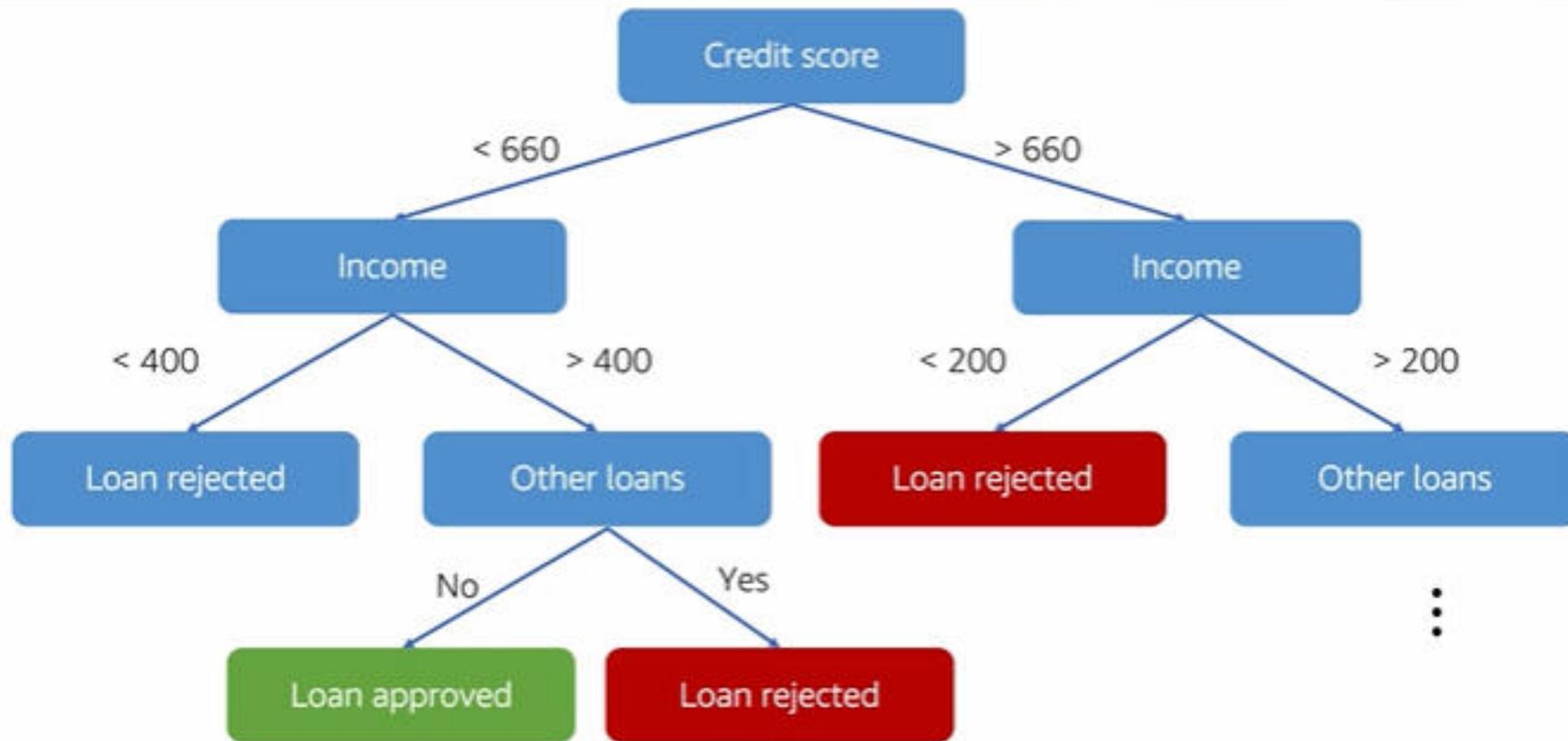
- Also popular approach in research, but not in industry
- “Kernelize” for nonlinear problems:
 - Choose a distance function named a “kernel”
 - Map the learning task to a higher dimension space
 - Apply a linear SVM classifier in the new space
- Not memory-efficient, because it stores the support vectors, which grow with the size of the training data
- Computation is expensive
- scikit-learn: `sklearn.svm.SVC`

Example

Will the loan of \$500,000 be approved for the customer?

Sample	Age	Household Income (k)	Other Loans	Marital Status	Credit Score	Loan Approved
1	60	257	Yes	Married	650	No
2	36	352	Yes	Married	800	Yes
3	43	957	No	Single	720	Yes
4	38	492	No	Single	600	Yes
5	29	181	Yes	Single	585	No
6	49	324	No	Single	690	Yes
7	40	128	No	Married	750	No
:	:	:	:	:	:	:

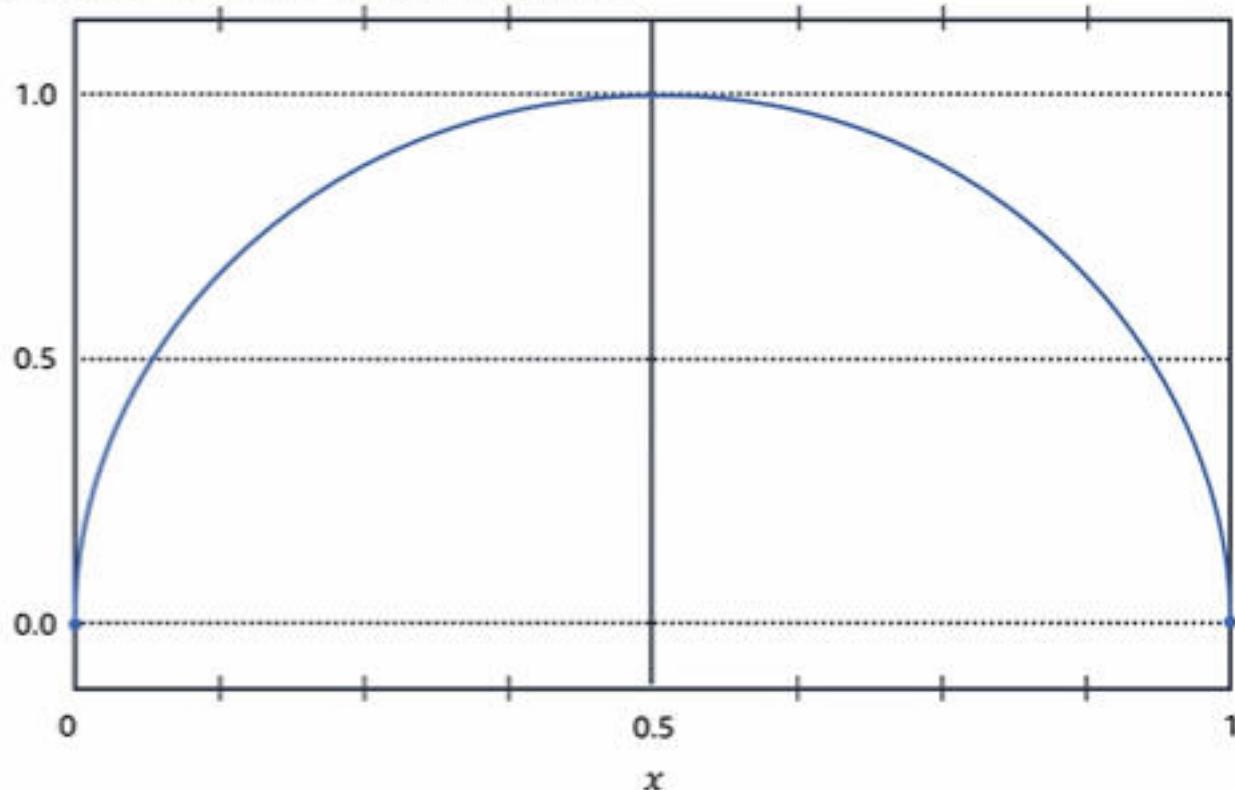
Decision trees



Entropy

- Relative measure of disorder in the data source

$$H(X) = - \sum_{i=1}^N P(x_i) \log(P(x_i))$$

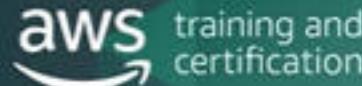


Decision trees



- **Nodes** are **split based** on the feature that has the **largest information gain** (IG) between parent node and its split nodes
- One metric to **quantify IG** is to **compare entropy before and after splitting**
- In a binary case:
 - **Entropy is 0** if all **samples belong to the same class** for a node (i.e., pure)
 - **Entropy is 1** if **samples contain both classes** with equal proportion (i.e., 50% for each class, chaos)
- The **splitting procedure** can go **iteratively** at each child node **until the end-nodes (or leaves) are pure** (i.e., there is only one class in each node)
 - But the splitting procedure usually stops at certain criteria to prevent overfitting

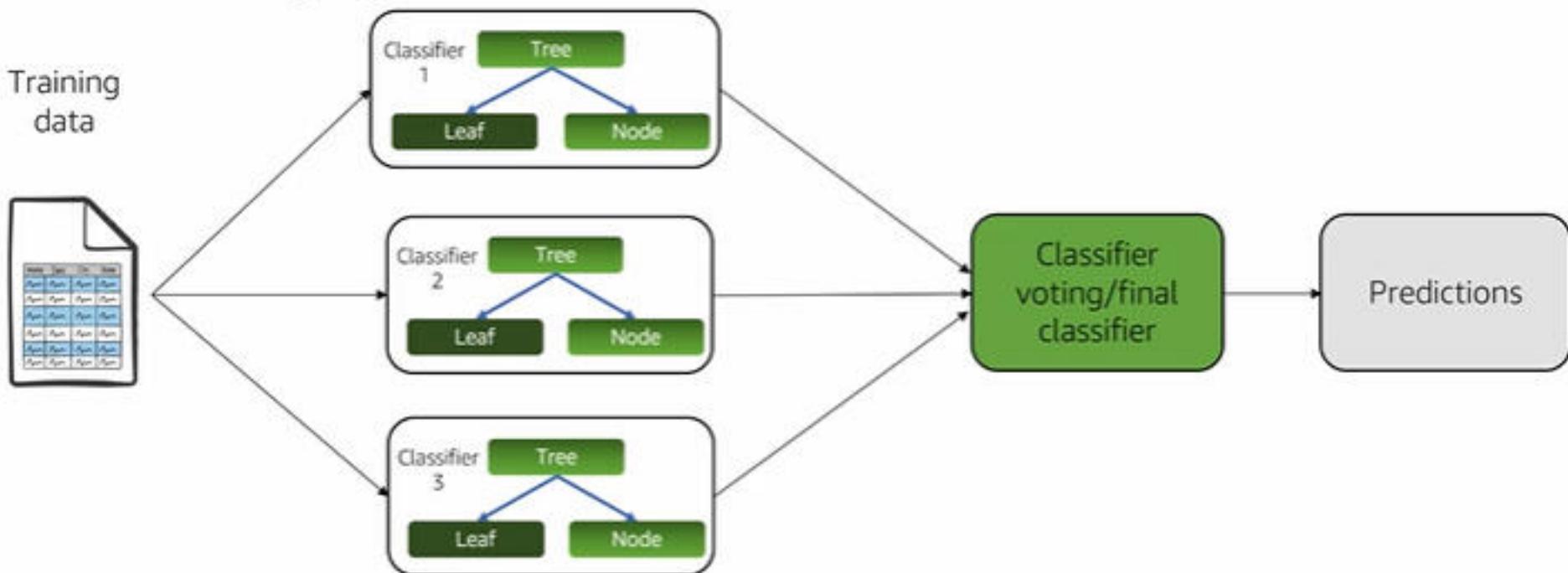
Decision trees



- Train (build the tree) by maximizing IG to choose splits (i.e., the impurity of split sets are lower)
- Easy to interpret (superficially)
- Expressive = flexible
- Less need for feature transformations
- Susceptible to overfitting
- Must “prune” the tree to reduce potential overfitting
- scikit-learn: `sklearn.tree.DecisionTreeClassifier`

Ensemble methods

Learn multiple models and combine their results, usually via majority vote or averaging

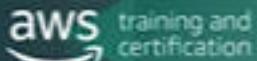


Random forest algorithm



- Set of decision trees, each learned from a different randomly sampled subset with replacement
- Features to split on for each tree, randomly selected subset from original features
- Prediction: Average output probabilities
- Increases diversity through random selection of training dataset and subset of features for each tree
- Reduces variance through averaging
- Each tree typically does not need to be pruned
- More expensive to train and run
- scikit-learn: `sklearn.ensemble.RandomForestClassifier`

Random forest algorithm



- Set of decision trees, each learned from a different randomly sampled subset with replacement
- Features to split on for each tree, randomly selected subset from original features
- Prediction: Average output probabilities
- Increases diversity through random selection of training dataset and subset of features for each tree
- Reduces variance through averaging
- Each tree typically does not need to be pruned
- More expensive to train and run
- scikit-learn: `sklearn.ensemble.RandomForestClassifier`



Model Training and Tuning Motivation



It's rare that the first model you train will meet your business requirements.

Model Training

Improve the model by optimizing parameters or data.



Model Tuning

Analyze the model for generalization quality and sources of underperformance such as overfitting.

Problems With Using Test Set



Motivation

Model training and tuning involve comparing performance for different model or data settings.

Problem

When you use the test set for these comparisons, that effectively makes it part of a training set.

- The model may learn the patterns from the test set during tuning.

Solution

Split training data in two parts: a training set and a validation set.

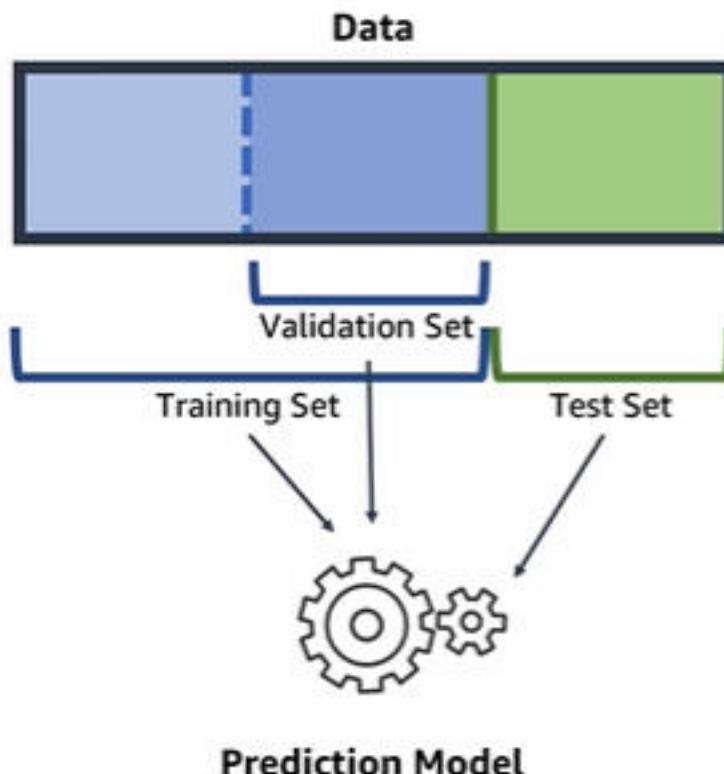
- Use the training set to train candidate models, etc.
- The validation set plays the role of the test set during debugging and tuning.
- Save the test set for measuring the generalization of your final model.

Solution

Split training data in two parts: a training set and a validation set.

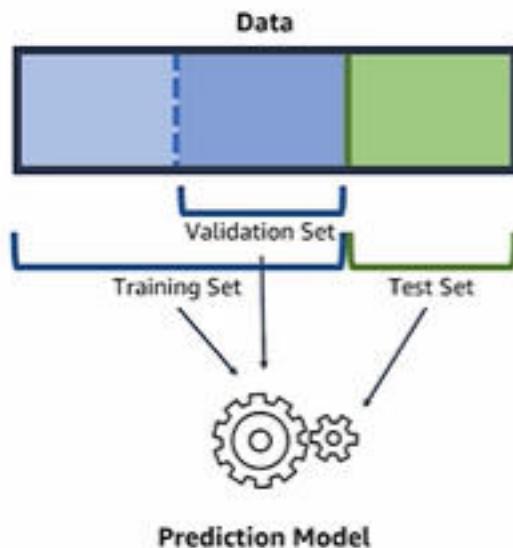
- Use the training set to train candidate models, etc.
- The validation set plays the role of the test set during debugging and tuning.
- Save the test set for measuring the generalization of your final model.

Validation Set



- **Issue:** Splitting the training data into training and validation sets may make it too small or unrepresentative.
- **Solution:** Use the holdout method to get the test set, then use k-fold cross-validation on the training set for debugging and tuning.

Validation Set



- **Issue:** Splitting the training data into training and validation sets may make it too small or unrepresentative.
- **Solution:** Use the holdout method to get the test set, then use k-fold cross-validation on the training set for debugging and tuning.



Data



Validation Set

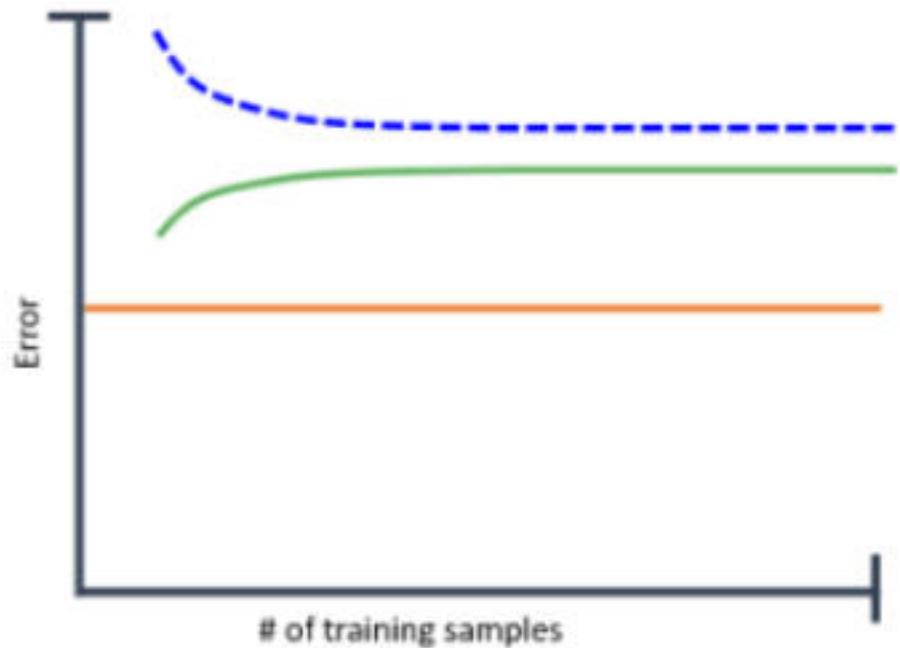
Training Set

Test Set

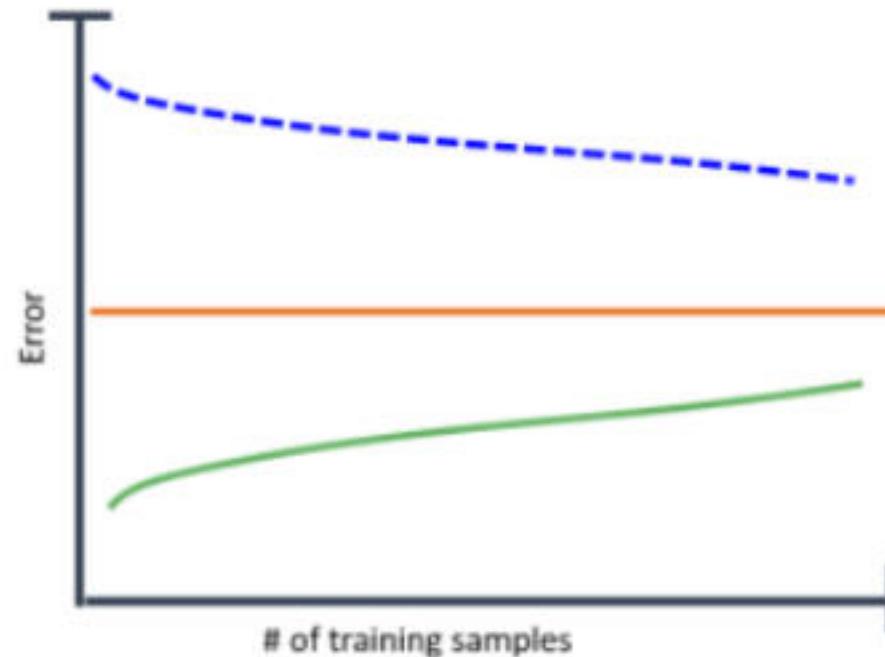


Click to flip

High Bias



High Variance



Click to flip



Defining Bias and Variance



$$\text{Total Error}(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias

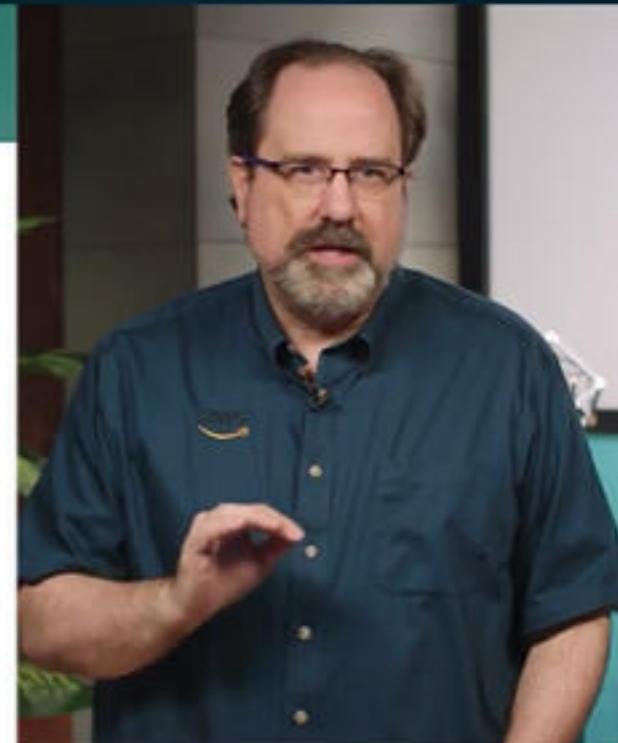
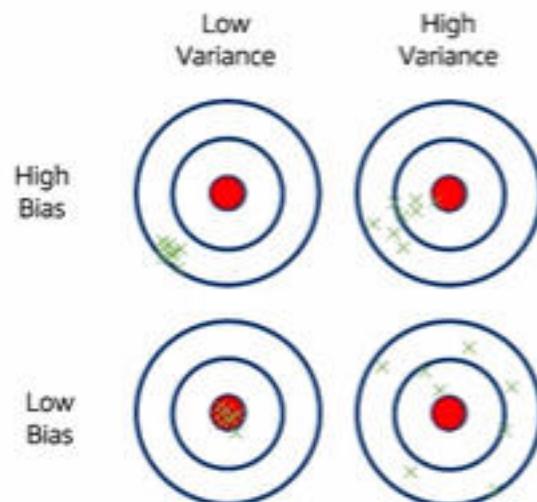
- $\text{Bias} = E[\hat{f}(x)] - f(x)$

Variance

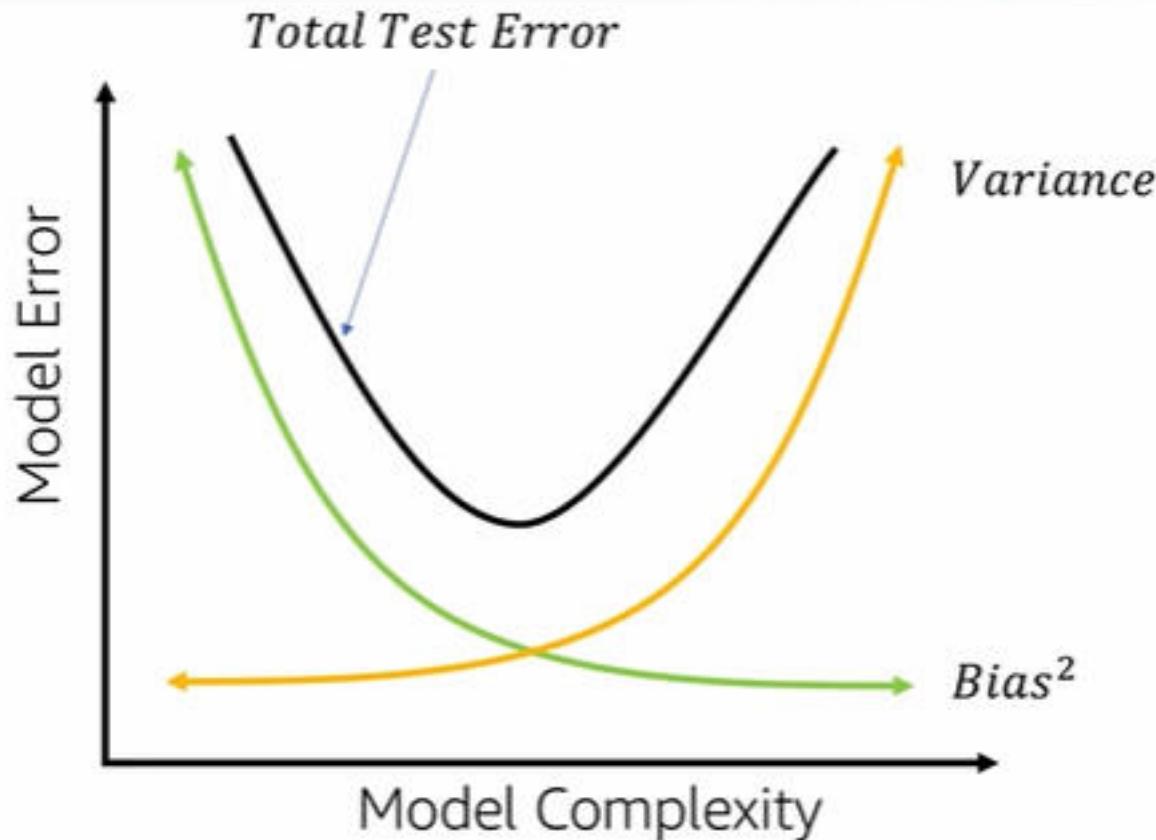
- $\text{Var} = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$

Bias-Variance Tradeoff

aws training and certification



In General



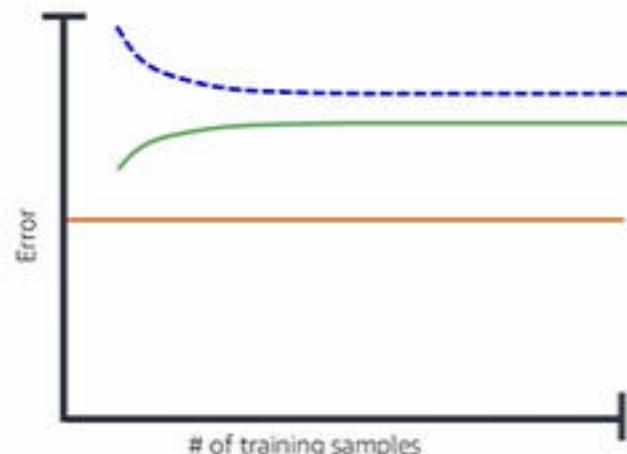
Using Learning Curves to Evaluate the Model



- **Motivation:** Detect if the model is underfitting or overfitting, and impact of **training data size** the error
- **Learning curves:** Plot training dataset and validation dataset error or accuracy against training set size
- **scikit-learn:** `sklearn.learning_curve.learning_curve`
 - Uses stratified k-fold cross-validation by default if output is binary or multiclass (preserves percentage of samples in each class)
 - Note: `sklearn.model_selection.learning_curve` in v.0.18

Learning Curves

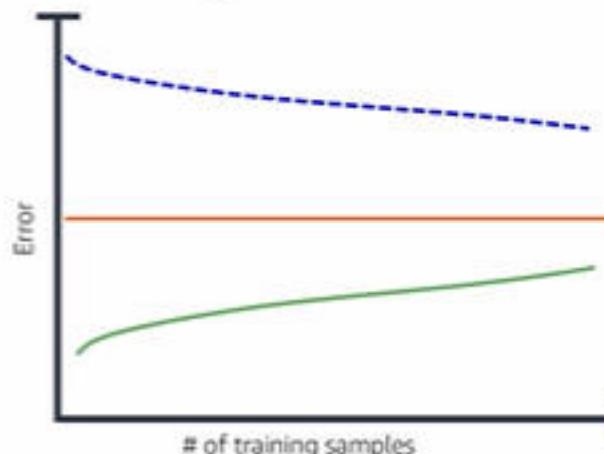
High Bias



Solution

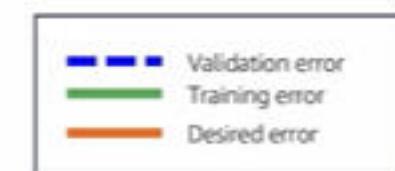
- Increase the number of features
- Decrease the degree of regularization

High Variance

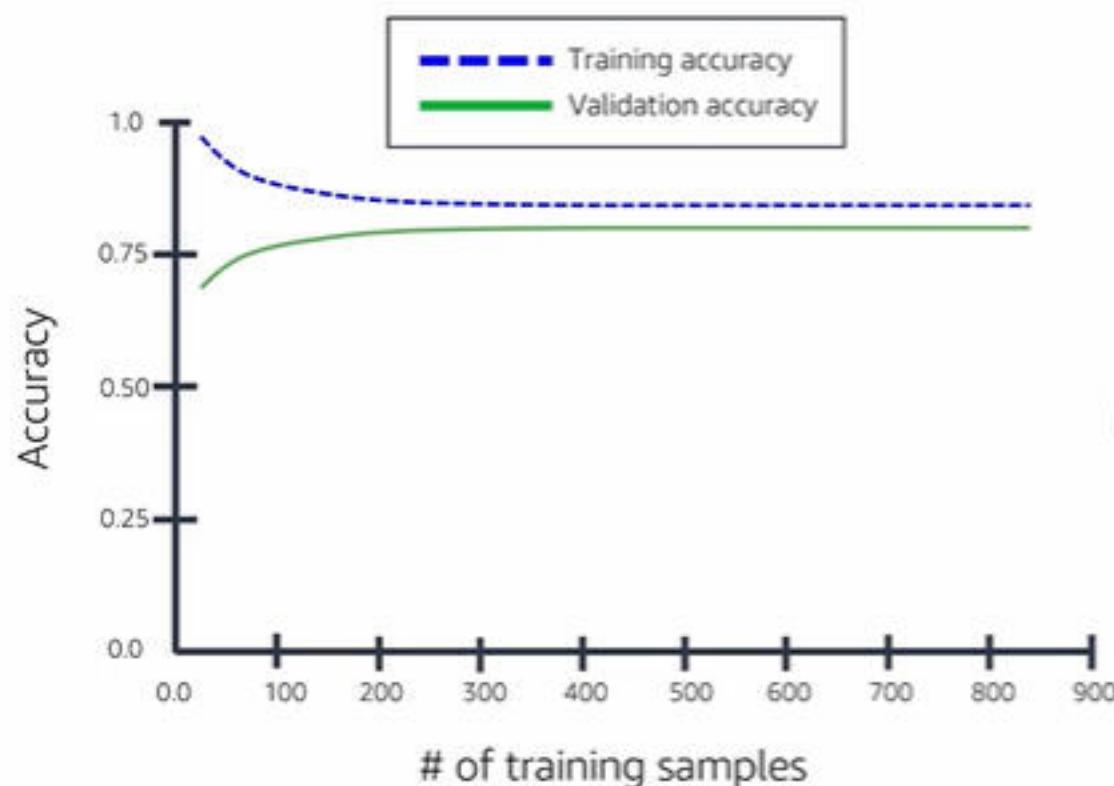


Solution

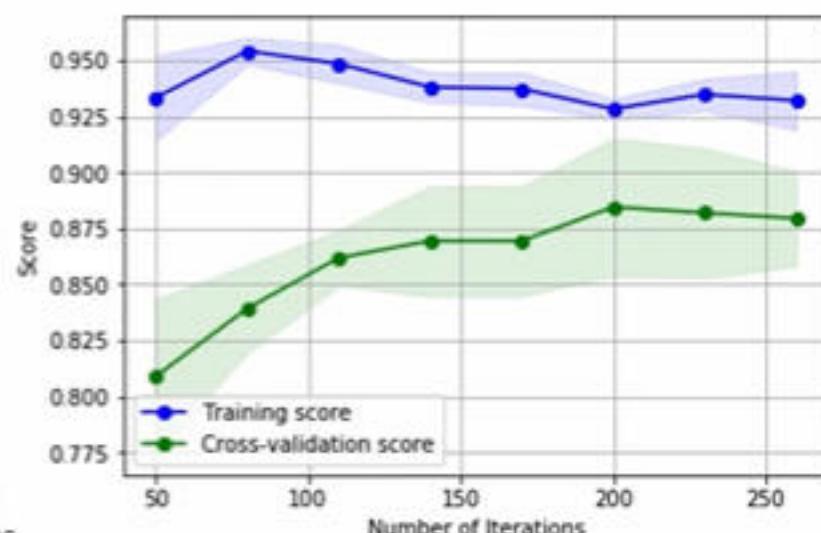
- Increase training data
- Reduce model complexity
 - Decrease the number of features
 - Increase the degree of regularization



Learning Curves on Breast Cancer Example

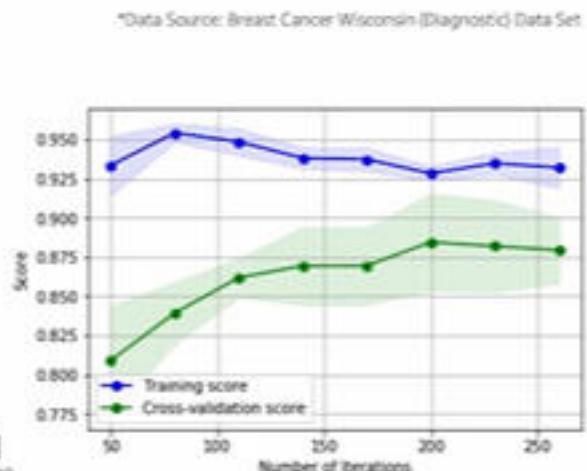
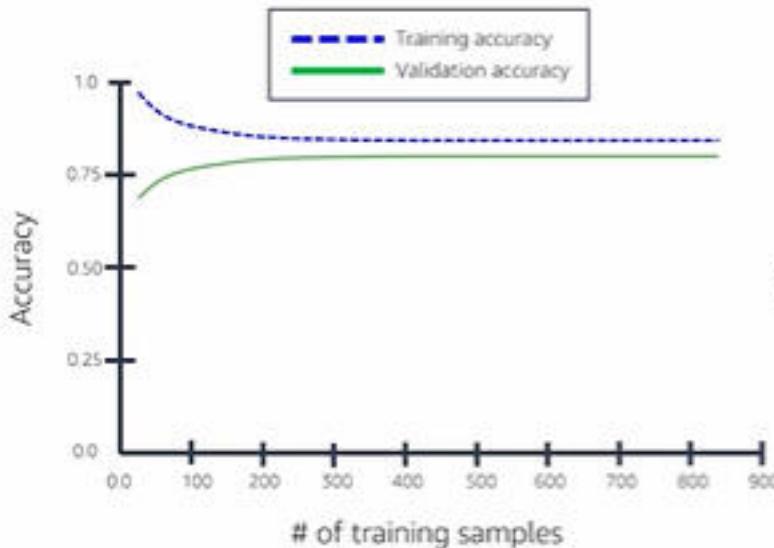


*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set



Learning Curves on Breast Cancer Example

aws training and certification



Error Analysis



- Some common patterns:
 - Data problems (e.g., many variants for the same word)
 - Labeling errors (e.g., data mislabeled)
 - Under/over-represented subclasses (e.g., too many examples of one type)
 - Discriminating information is not captured in features (e.g., customer locations)
- **Note:** It often helps to look at what model is predicting *correctly*

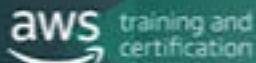


-1:18

2x



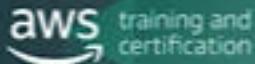
Error Analysis



- Some common patterns:
 - Data problems (e.g., many variants for the same word)
 - Labeling errors (e.g., data mislabeled)
 - Under/over-represented subclasses (e.g., too many examples of one type)
 - Discriminating information is not captured in features (e.g., customer locations)
- **Note:** It often helps to look at what model is predicting *correctly*



Regularization



- **Motivation:** Overfitting often caused by overly-complex models capturing idiosyncrasies in training set
- **Regularization:** Adding penalty score for complexity to cost function

$$cost_{reg} = cost + \frac{\alpha}{2} \text{penalty}$$

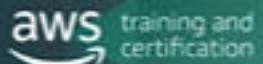


Regularization in Linear Models



- Idea: Large weights correspond to higher complexity
⇒ Regularize by penalizing large weights
- Two standard types:

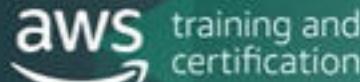
Regularization in Linear Models



- Idea: Large weights correspond to higher complexity
⇒ Regularize by penalizing large weights
- Two standard types:
 - L1 regularization, Lasso: $\text{penalty} = \|\vec{w}\|_1 = \sum_{j=1}^m |w_j|$
 - L2 regularization, Ridge: $\text{penalty} = \|\vec{w}\|_2^2 = \sum_{j=1}^m w_j^2$



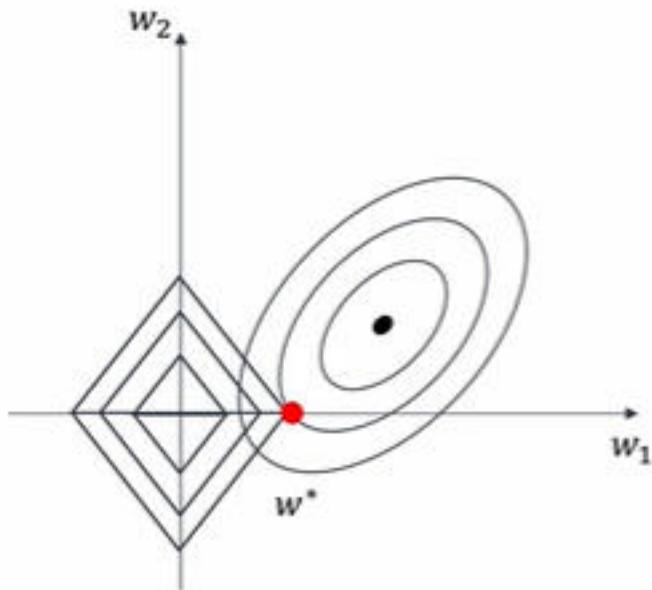
Regularization in Linear Models



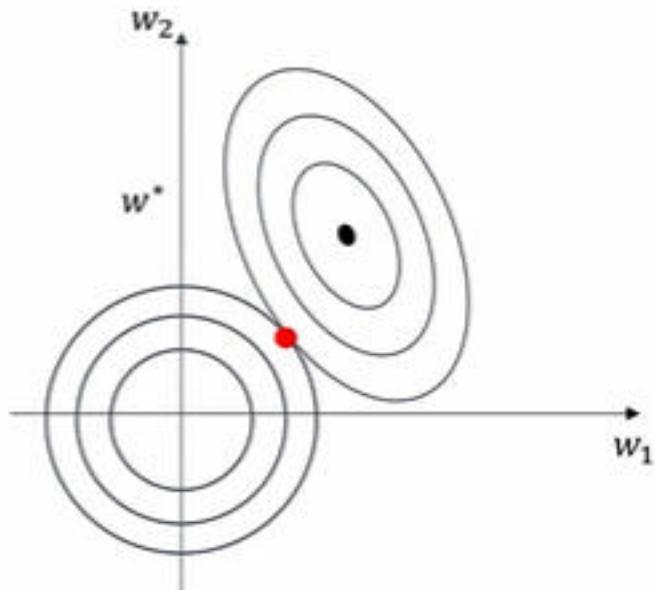
- L2 popular, but L1 useful as feature selection approach since most weights shrink to 0 (*sparsity*)
- **Note:** Important to scale features first!
- **scikit-learn:** Models that support regularization typically provide parameters for type and strength

Regularization in Linear Models

L1



L2



Regularization for Regression



- Ridge regression model: Linear regression with L2
Sklearn: `sklearn.linear_model.Ridge`
- Lasso regression model: Linear regression with L1
Sklearn: `sklearn.linear_model.Lasso`
- Elastic Net regression model: Linear regression with both
Sklearn: `sklearn.linear_model.ElasticNet`
- The strength of regularization $c = \frac{1}{alpha}$

Grid Search



```
from sklearn.datasets import make_classification
from sklearn import svm
from sklearn.model_selection import GridSearchCV

plt.figure(4)
X2, Y2 = make_classification(n_features=5, n_redundant=0, n_informative=2)

parameters = {'kernel':('linear', 'rbf'), 'C':[1, 5, 10, 15], 'degree':[2,3,4,5]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(X2, Y2)

clf.best_params_
{'C': 5, 'degree': 2, 'kernel': 'rbf'}
```



Grid Search

```
from sklearn.datasets import make_classification
from sklearn import svm
from sklearn.model_selection import GridSearchCV

plt.figure(4)
X2, Y2 = make_classification(n_features=5, n_redundant=0, n_informative=2)

parameters = {'kernel':('linear', 'rbf'), 'C':[1, 5, 10, 15], 'degree':[2,3,4,5]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(X2, Y2)

clf.best_params_
{'C': 5, 'degree': 2, 'kernel': 'rbf'}
```

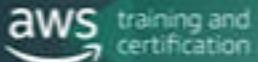
Training Data Tuning



- Training set too small?
 - Sample and label more data if possible
- Training set biased against or missing some important scenarios?
 - Sample and label more data for those scenarios if possible
- Can't easily sample or label more?
 - Consider creating synthetic data (duplication or techniques like **SMOTE**)
- **Important:** Training data doesn't need to be exactly representative ("Whatever works"), but your test set does.



Feature Set Tuning



- Add features that help capture pattern for classes of errors.
- Try different transformations of the same feature.
 - Example: Square the values of a feature (as in earlier example)
- Apply dimensionality reduction to reduce impact of weak features.



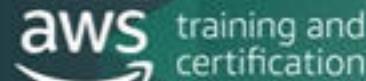
Dimensionality Reduction



- **Motivation:** Common cause of overfitting: too many features for the amount of data
 - This is exacerbated if there are noisy/irrelevant features
 - Also a problem for curse of dimensionality
- **Dimensionality reduction:** Reduce the (effective) dimension of the data with minimal loss of information



Dimensionality Reduction



- **Motivation:** Common cause of overfitting: too many features for the amount of data
 - This is exacerbated if there are noisy/irrelevant features
 - Also a problem for curse of dimensionality
- **Dimensionality reduction:** Reduce the (effective) dimension of the data with minimal loss of information



-1:36

2x



Feature Extraction



Maps data into smaller feature space that captures the bulk of the information in the data

- a.k.a. data compression
- Motivation
 - Improves computational efficiency
 - Reduces curse of dimensionality
- Techniques
 - Principal component analysis (PCA)
 - Linear discriminant analysis (LDA)
 - Kernel versions of these for fundamentally non-linear data

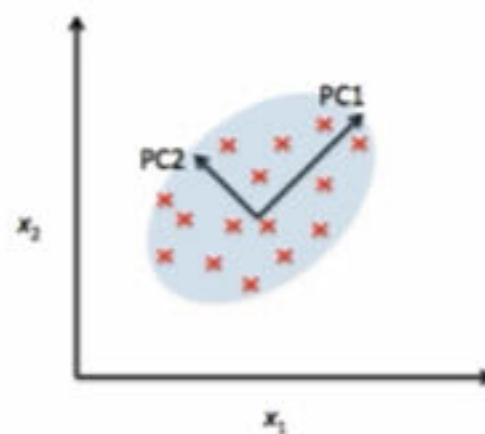


Principal Component Analysis (PCA)



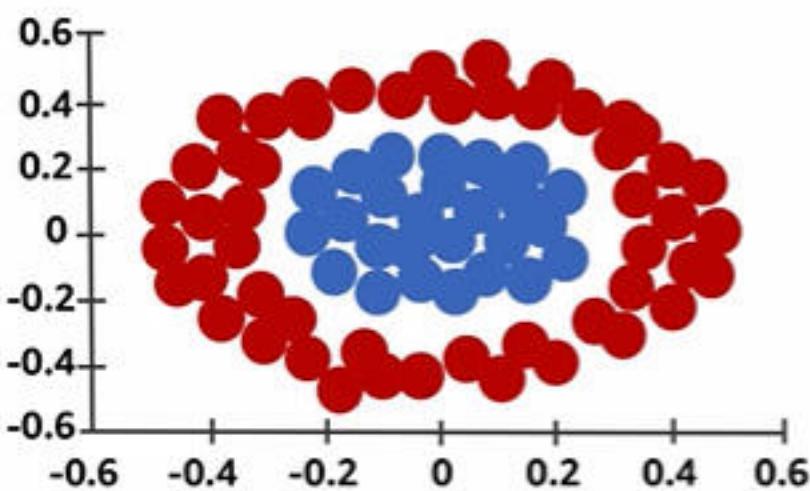
- Is an **unsupervised** linear approach to feature extraction
- Finds patterns based on correlations between features
- Constructs principal components: orthogonal axes in directions of maximum variance
- **scikit-learn:** `sklearn.decomposition.PCA`

```
>>> pca = PCA(n_components=2)  
>>> X_train_pca = pca.fit_transform(X_train_std)  
>>> lr = LogisticRegression()  
>>> lr.fit(X_train_pca)
```

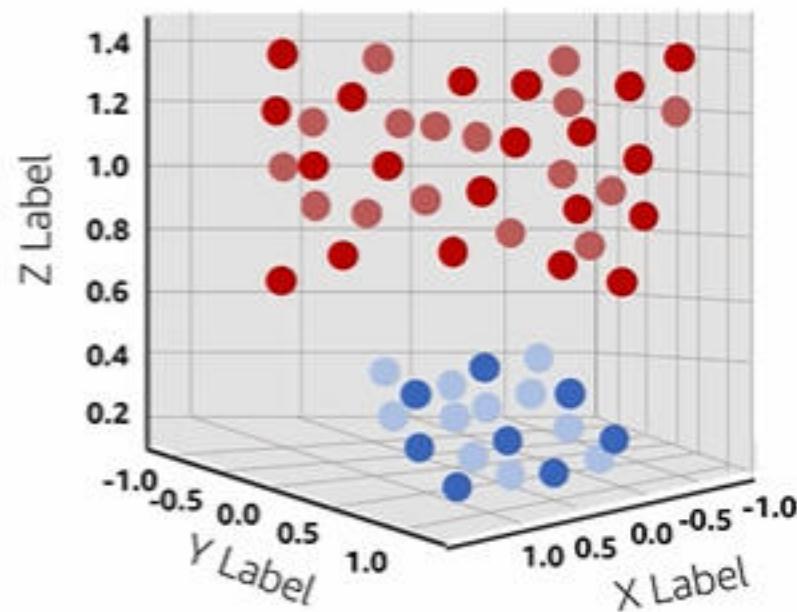


Kernel PCA

Original



Kernel PCA



Linear Discriminant Analysis (LDA)



- A **supervised** linear approach to feature extraction
- Transforms to subspace that maximizes class separability
- Assumes data is normally distributed
- Used for dimensionality reduction of features
- Can reduce to at most $\# \text{classes} - 1$ components
- **scikit-learn:** `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`

Bagging (Bootstrap Aggregating)



Motivation: generate a group of weak learners that when combined together generate higher accuracy

- Create x datasets of size m by randomly sampling original dataset with replacement (duplicates allowed)



Dataset 1



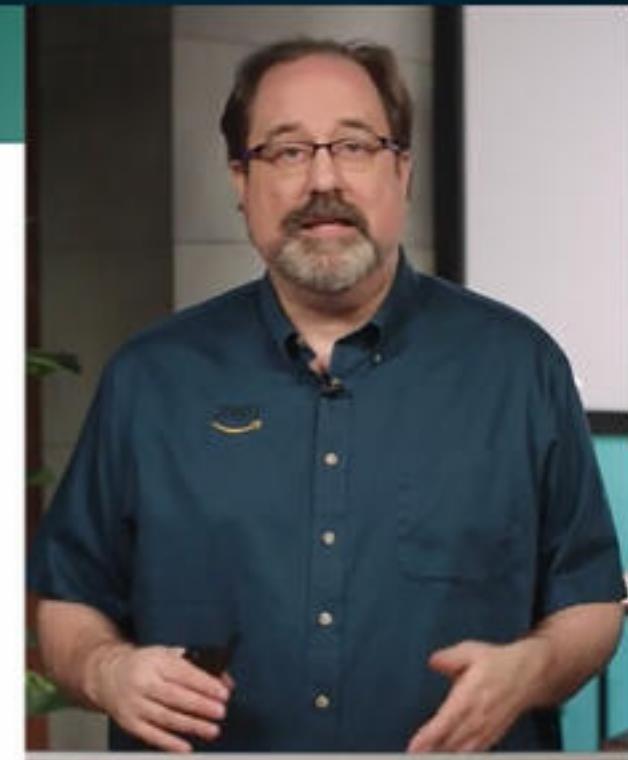
Dataset 2



Dataset 3



Dataset 4



-5:30

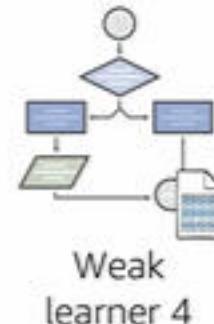
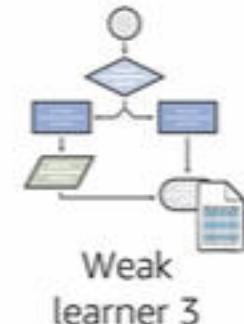
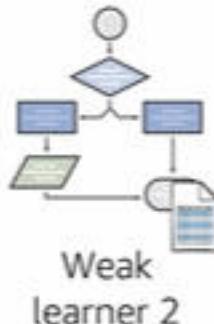
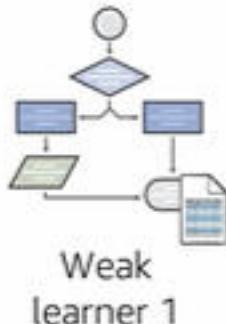
2x



Bagging (Bootstrap Aggregating)



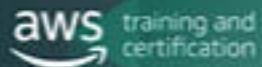
- Train weak learners (decision stumps, logistic regression) on the new datasets to generate predictions
- Choose the output by combining the individual predictions or voting



Voting

-4:21
 Y^{2x}

Bagging



High variance but **low** bias?

Use **bagging**:

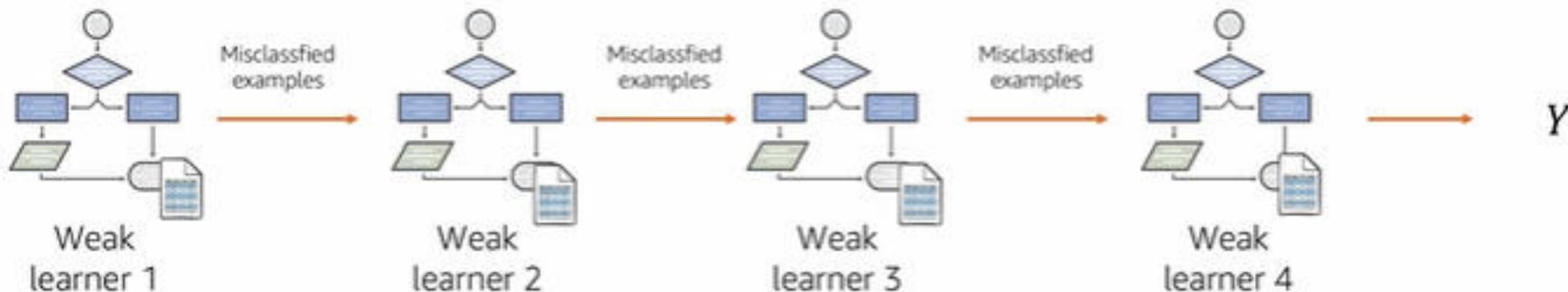
Training many models on random subsets of the data and average/vote on the output.

- Reduces variance
- Keeps bias the same
- **sklearn**:
 - `sklearn.ensemble.BaggingClassifier`
 - `sklearn.ensemble.BaggingRegressor`



Boosting

- Assign strengths to each weak learner
- Iteratively train learners using misclassified examples by the previous weak learners



Boosting

Model has a **high bias** and accepts weights on **individual samples**?

Use **boosting**:

Training a sequence of samples to get a strong model.

- Often times wins on datasets like most Kaggle competitions
- **sklearn**:
 - `sklearn.ensemble.AdaBoostClassifier`
 - `sklearn.ensemble.AdaBoostRegressor`
 - `sklearn.ensemble.GradientBoostingClassifier`
- **XGBoost** library

Productizing a ML Model



Aspects to consider:

- Model hosting
- Model deployment
- Pipelines to provide feature vectors
- Code to provide low-latency and/or high-volume predictions
- Model and data updating and versioning
- Quality monitoring and alarming
- Data and model security and encryption
- Customer privacy, fairness, and trust
- Data provider contractual constraints (e.g., attribution, cross-fertilization)

Productizing a ML Model



Aspects to consider:

- Model hosting
- Model deployment
- Pipelines to provide feature vectors
- Code to provide low-latency and/or high-volume predictions
- Model and data updating and versioning
- Quality monitoring and alarming
- Data and model security and encryption
- Customer privacy, fairness, and trust
- Data provider contractual constraints (e.g., attribution, cross-fertilization)

Types of production environments



Batch predictions

- Useful if all possible inputs known *a priori* (e.g., all product categories for which demand is to be forecast, all keywords to bid)
- Predictions can still be served real-time, simply read from pre-computed values

Online predictions

- Useful if input space is large (e.g., customer's utterances or photos, detail pages to be translated)
- Low latency requirement (e.g., at most 100ms)

Online training

- Sometimes training data patterns change often, so need to train online (e.g., fraud detection)

Types of production environments



Batch predictions

- Useful if all possible inputs known *a priori* (e.g., all product categories for which demand is to be forecast, all keywords to bid)
- Predictions can still be served real-time, simply read from pre-computed values

Online predictions

- Useful if input space is large (e.g., customer's utterances or photos, detail pages to be translated)
- Low latency requirement (e.g., at most 100ms)

Online training

- Sometimes training data patterns change often, so need to train online (e.g., fraud detection)

Business Metrics vs. Model Metrics



- Business metrics may not be the same as the performance metrics that are optimized during training. Why?
 - Example: Click-through rate
- Ideally, performance metrics are highly correlated with business metrics.



Confusion Matrix

Predicted class	
P	P
	True Positives (TP)
N	N
	False Negatives (FN)
N	P
	False Positives (FP)
N	N
	True Negatives (TN)

Breast Cancer Example

		Predict Label
		Confusion Matrix
		1 (M) 0 (B)
True Label	1 (M)	146 24
True Label	0 (B)	11 274

*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

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

Also called *score*

Confusion Matrix

Predicted class		
P	P	True Positives (TP)
	N	False Negatives (FN)
N	P	False Positives (FP)
	N	True Negatives (TN)

Breast Cancer Example

		Predict Label
True Label	Confusion Matrix	
	1 (M)	0 (B)
1 (M)	146	24
0 (B)	11	274

*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

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

Also called *score*

Confusion Matrix

		Predicted class	
		P	N
Actual class P	True Positives (TP)	False Negatives (FN)	
	False Positives (FP)	True Negatives (TN)	

Breast Cancer Example

		Predict Label	
True Label	Confusion Matrix	$1 (M)$	$0 (B)$
	$1 (M)$	146	24
$0 (B)$	11	274	

*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

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

Also called *score*

Issue: In many applications, TN dwarfs the other categories, making accuracy useless for comparing models.

Precision: Proportion of positive predictions that are actually correct

- $Precision = \frac{TP}{TP+FP}$
- Example: $\frac{146}{146+11} = 0.9299$

Issue: In many applications, TN dwarfs the other categories, making accuracy useless for comparing models.

Recall: Proportion of positive set that are identified as positive

- $Recall = \frac{TP}{TP+FN}$
- Example: $\frac{146}{146+24} = 0.8588$

Issue: In many applications, TN dwarfs the other categories, making accuracy useless for comparing models.

F₁-Score: Combination (harmonic mean) of precision and recall

- $$F_1 - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Using Entire Dataset to Train the Model



```
from sklearn import svm
from sklearn.metrics import confusion_matrix

## Attributes to be included in the model
col = ['V10', 'V11', 'V12', 'V13', 'V14', 'V15']

clf_full = svm.SVC()
clf_full.fit(df[col], df['target'])

pred = clf_full.predict(df[col])
print(clf_full.score(df[col],df['target']))

print(confusion_matrix(y_true=df['target'],
                       y_pred=pred, labels=[1, 0]))
```

		Predict Label
True Label	Confusion Matrix	
	1 (M)	0 (B)
1 (M)	357	0
0 (B)	3	209

*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

Score = 0.9947

This leads to overfitting.

Using Entire Dataset to Train the Model



```
from sklearn import svm
from sklearn.metrics import confusion_matrix

## Attributes to be included in the model
col = ['V10', 'V11', 'V12', 'V13', 'V14', 'V15']

clf_full = svm.SVC()
clf_full.fit(df[col], df['target'])

pred = clf_full.predict(df[col])
print(clf_full.score(df[col], df['target']))

print(confusion_matrix(y_true=df['target'],
                       y_pred=pred, labels=[1, 0]))
```

True Label	Predict Label	
	1 (M)	0 (B)
1 (M)	357	0
0 (B)	3	209

*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

Score = 0.9947

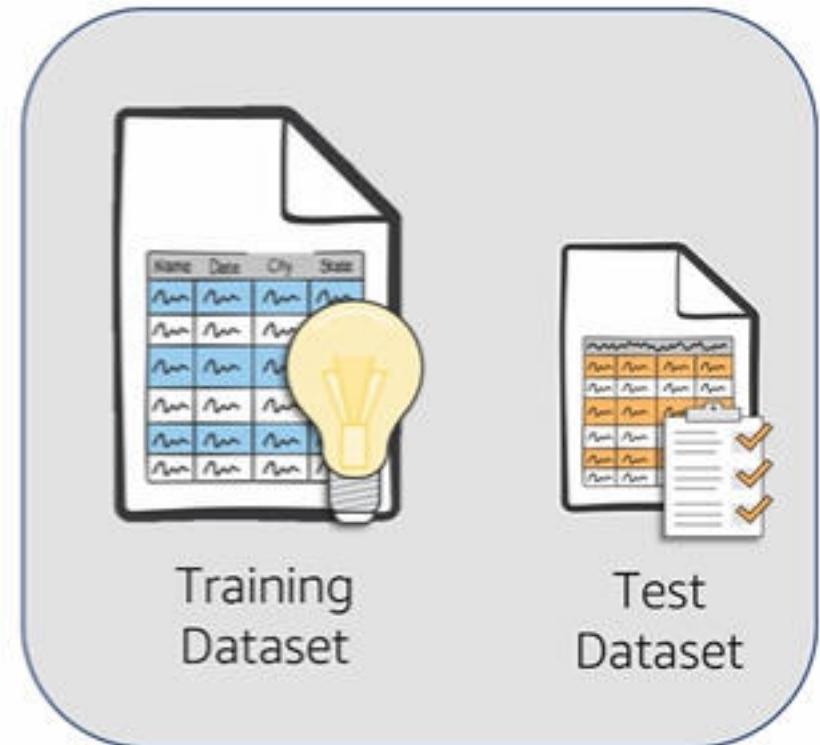
This leads to overfitting.



Holdout Method

Split data set into separate training and test sets:

- **Training set:** Used to train, tune, and select model
- **Test set:** Used to evaluate final model



Using Training and Testing Data Sets



```
from sklearn.model_selection import train_test_split

# split data into training and test sets
train, test = train_test_split(df[col + ['target']],
                               test_size = 0.3)

# Fit the SVM model
clf = svm.SVC()
clf.fit(train[col], train['target'])
```

Using Training and Testing Data Sets

```
from sklearn.model_selection import train_test_split

# split data into training and test sets
train, test = train_test_split(df[col + ['target']],
                               test_size = 0.3)

# Fit the SVM model
clf = svm.SVC()
clf.fit(train[col], train['target'])
```

```
# Check the confusion matrix and score for training data
pred = clf.predict(train[col])
print(confusion_matrix(y_true=train['target'],
                       y_pred=pred, labels=[1, 0]))

print(clf.score(train[col],train['target']))
```

Using Training and Testing Data Sets



```
from sklearn.model_selection import train_test_split

# split data into training and test sets
train, test = train_test_split(df[col + ['target']],
                               test_size = 0.3)

# Fit the SVM model
clf = svm.SVC()
clf.fit(train[col], train['target'])
```

```
# Check the confusion matrix and score for training data
pred = clf.predict(train[col])
print(confusion_matrix(y_true=train['target'],
                       y_pred=pred, labels=[1, 0]))

print(clf.score(train[col],train['target']))
```

```
# Check the confusion matrix and score for test data
pred = clf.predict(test[col])
print(confusion_matrix(y_true=test['target'],
                       y_pred=pred, labels=[1, 0]))

print(clf.score(test[col], test['target']))
```

Still Overfitting—Too Many Variables

*Data Source: Breast Cancer Wisconsin (Diagnostic) Data Set

Training Dataset

Predict Label		
True Label	1 (B)	0 (M)
Confusion Matrix	1 (B)	0 (M)
1 (B)	258	0
0 (M)	2	138

Score = 0.9949

Testing Dataset

Predict Label		
True Label	1 (B)	0 (M)
Confusion Matrix	1 (B)	0 (M)
1 (B)	98	1
0 (M)	66	6

Score = 0.6081

- **Issue: Small sets**
 - Smaller training set → not enough data for good training
 - Unrepresentative test set → invalid metrics
- **K-Fold cross-validation**
 - Randomly partition data into K “folds”
 - For each fold, train model on other $K-1$ folds and evaluate on that
 - Train on all data
 - Average metric across K folds estimates test metric for trained model

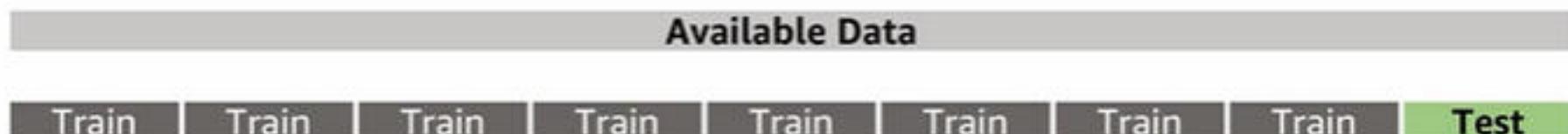
K-Fold Cross-Validation



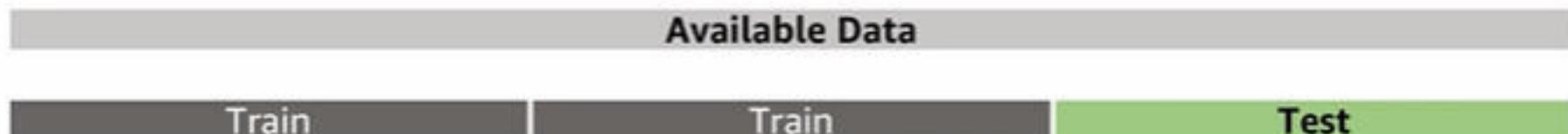
- **Issue: Small sets**
 - Smaller training set → not enough data for good training
 - Unrepresentative test set → invalid metrics
- **K-Fold cross-validation**
 - Randomly partition data into K “folds”
 - For each fold, train model on other $K-1$ folds and evaluate on that
 - Train on all data
 - Average metric across K folds estimates test metric for trained model

K-Fold CV: Choosing K

- Large → more time, more variance



- Small → more bias

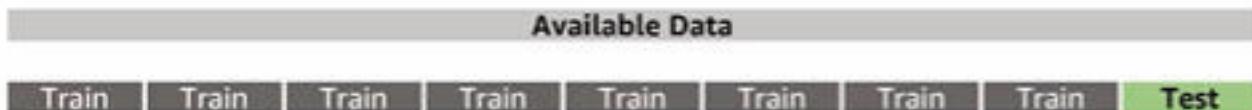


- 5-10 value of K is typical

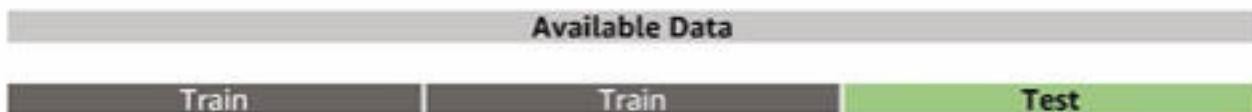
K-Fold CV: Choosing K



- Large → more time, more variance



- Small → more bias



- 5-10 value of K is typical



K-Fold Cross-Validation



Leave-one-out cross-validation

- $K = \text{number of data points}$
- Used for very small sets

Stratified K-fold cross-validation

- Preserve class proportions in the folds
- Used for imbalanced data
- There are seasonality or subgroups

Issue

- Confusion matrix doesn't make sense for regression

Solution

- Use regression metrics
 - Mean squared error
 - R^2

Mean Squared Error



- Average squared error over entire dataset

$$\text{Mean squared error (MSE)} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

- Very commonly used
- scikit-learn: `sklearn.metrics.mean_squared_error`

R²: Coefficient of Determination



- $R^2 = 1 - \frac{\text{Sum of Squared Error (SSE)}}{\text{Var}(y)}$ which is between 0 and 1
- Interpretation: Fraction of variance accounted for by the model
- Basically, standardized version of MSE
- Good R^2 are determined by actual problem
- R^2 always increases when more variables are added to the model

$$\text{Adjusted } R^2 = 1 - (1 - R^2) \frac{\text{no. of data pts.} - 1}{\text{no. of data pts.} - \text{no. of variables} - 1}$$

- Takes into account of the effect of adding more variables such that it only increases when the added variables have significant effect in prediction

R²: Coefficient of Determination



- $R^2 = 1 - \frac{\text{Sum of Squared Error (SSE)}}{\text{Var}(y)}$ which is between 0 and 1
- Interpretation: Fraction of variance accounted for by the model
- Basically, standardized version of MSE
- Good R^2 are determined by actual problem
- R^2 always increases when more variables are added to the model

$$\text{Adjusted } R^2 = 1 - (1 - R^2) \frac{\text{no. of data pts.} - 1}{\text{no. of data pts.} - \text{no. of variables} - 1}$$

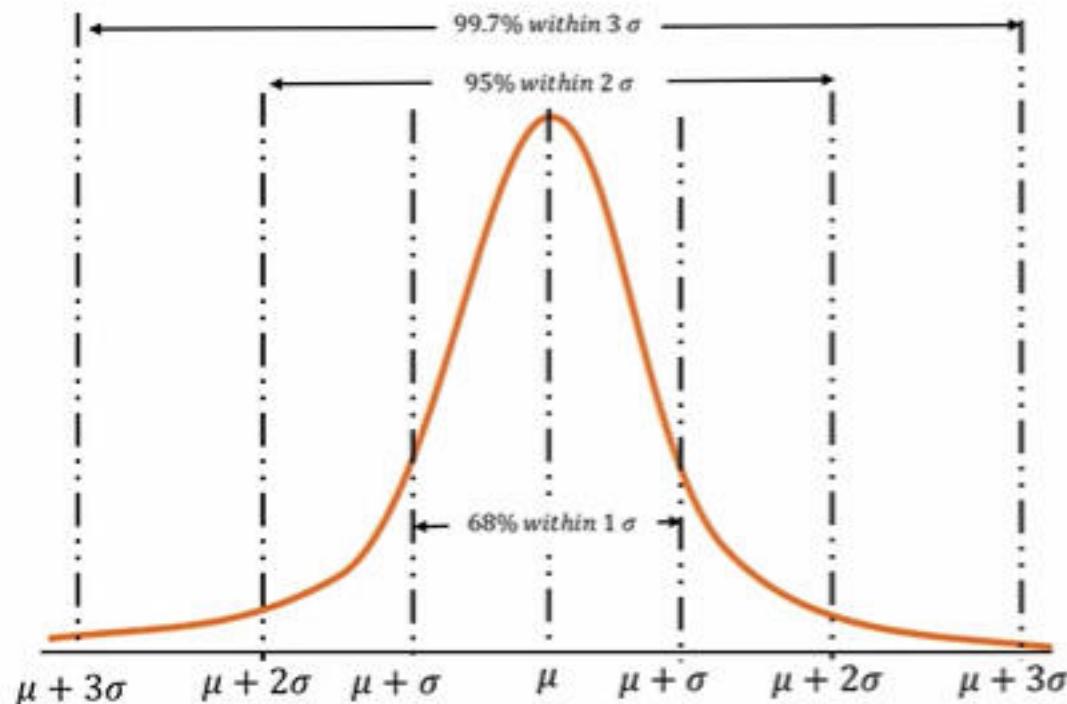
- Takes into account of the effect of adding more variables such that it only increases when the added variables have significant effect in prediction

R²: Coefficient of Determination



- R^2 will always increase when more explanatory variables are added to the model, highest R^2 may not be the best model
- Adjusted- R^2 is a better metric for multiple variates regression
- scikit-learn: `sklearn.metrics.r2_score`

Normal (Gaussian) Distribution



Probability Density Function

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

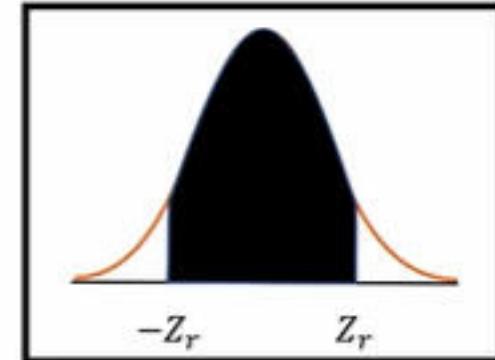
- μ is mean or expectation of the distribution
- σ is standard deviation
- σ^2 is variance

Why do we study normal distribution so often?

Central Limit Theorem: no matter what is the original distribution of X , the mean of X (i.e. \bar{X}) will follow a normal distribution: $\bar{X} \sim N\left(\mu, \frac{\sigma^2}{n}\right)$.

Confidence Interval

- An average computed on a sample is merely an *estimate* of the true population mean.
- **Confidence interval:** Quantifies margin-of-error between sample metric and true metric due to *sampling randomness*.
- **Informal interpretation:** With $x\%$ confidence, true metric lies within the interval.
- **Precisely:** If the true distribution is as stated, then with $x\%$ probability the observed value is in the interval.
- **Z-score:** Quantifies how much the value is above or below the mean in terms of its standard deviation

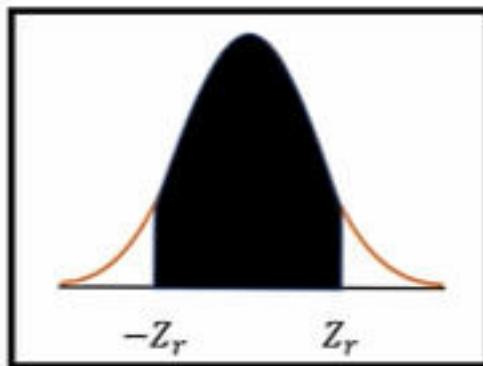


Prob = r	z -score = z_r
0.90	1.645
0.95	1.96
0.98	2.326
0.99	2.576

Confidence Interval

For population proportion (i.e. truth), the confidence interval is:

- $CI = p \pm z(p(1 - p)/n)^{1/2}$
- Where p is sample proportion, n is sample size n , and z is z-score defined in the table above, which is determined by the confidence level.



Prob = r	z-score = z_r
0.90	1.645
0.95	1.96
0.98	2.326
0.99	2.576

Confidence Interval

For **population mean** (i.e. truth), the confidence interval is:

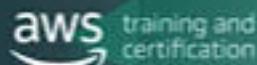
- $CI = \bar{X} \pm z^*s/n^{1/2}$
- Where \bar{X} is sample mean, n is sample size, s is sample standard deviation, and z is z-score
- **Assumptions:** n > 30 and p not close to 0 or 1
- Doesn't capture error due to other sources of bias

Prob = r	z-score = z_r
0.90	1.645
0.95	1.96
0.98	2.326
0.99	2.576

Data Storage Formats

- Row-oriented formats:
 - Comma/tab-separated values (CSV/TSV)
 - Read-only DB (RODB): Internal read-only file-based store with fast key-based access
 - Avro: allows schema evolution for Hadoop
- Column-oriented formats:
 - Parquet: Type-aware and indexed for Hadoop
 - Optimized row columnar (ORC): Type-aware, indexed, and with statistics for Hadoop
- User-defined formats:
 - JavaScript object notation (JSON): For key-value objects
 - Hierarchical data format 5 (HDF5): Flexible data model with chunks
- Compression can be applied to all formats
- Usual trade-offs: Read/write speeds, size, platform-dependency, ability for schema to evolve, schema/data separability, type richness

Data Storage Formats



- Row-oriented formats:
 - Comma/tab-separated values (CSV/TSV)
 - Read-only DB (RODB): Internal read-only file-based store with fast key-based access
 - Avro: allows schema evolution for Hadoop
- Column-oriented formats:
 - Parquet: Type-aware and indexed for Hadoop
 - Optimized row columnar (ORC): Type-aware, indexed, and with statistics for Hadoop
- User-defined formats:
 - JavaScript object notation (JSON): For key-value objects
 - Hierarchical data format 5 (HDF5): Flexible data model with chunks
- Compression can be applied to all formats
- Usual trade-offs: Read/write speeds, size, platform-dependency, ability for schema to evolve, schema/data separability, type richness



Predictive Model Markup Language (PMML):

- Vendor-independent XML-based language for storing ML models
- Support varies in different libraries:
 - KNIME (analytics/ML library): Full support
 - Scikit-learn: Extensive support
 - Spark MLlib: Limited support

Custom methods:

- **Scikit-learn**: Uses the Python pickle method to serialize/deserialize Python objects
- **Spark MLlib**: Transformers and Estimators implement MLWritable
- **TensorFlow (deep learning library)**: Allows saving of MetaGraph
- **MxNet (deep learning library)**: Saves into JSON

Model Deployment



Technology transfer: Experimental framework may not suffice for production

- A/B testing or shadow testing: Helps catch production issues early

Zinkevich, Martin. *Rules of Machine Learning: Best Practices for ML Engineering.* (2017).

Model Deployment



Technology transfer: Experimental framework may not suffice for production

- A/B testing or shadow testing: Helps catch production issues early

Zinkevich, Martin. *Rules of Machine Learning: Best Practices for ML Engineering.* (2017).



Model Deployment



Technology transfer: Experimental framework may not suffice for production

- A/B testing or shadow testing: Helps catch production issues early

Zinkevich, Martin. *Rules of Machine Learning: Best Practices for ML Engineering.* (2017).

Model Deployment



Technology transfer: Experimental framework may not suffice for production

- A/B testing or shadow testing: Helps catch production issues early

Zinkevich, Martin. *Rules of Machine Learning: Best Practices for ML Engineering.* (2017).

- Make sure that you handle training and evaluation data in accordance with data classification.
- Models may need to be treated with same classification level as source data. *Why?*

Information Security



- Make sure that you handle training and evaluation data in accordance with data classification.
- Models may need to be treated with same classification level as source data. *Why?*



Monitoring



- It's important to monitor quality metrics and business impacts with dashboards, alarms, user feedback, etc.:
 - The real-world domain may change over time.
 - The software environment may change.
 - High profile special cases may fail.
 - There may be a change in business goals.



© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



-5:43

2x



Monitoring



- It's important to monitor quality metrics and business impacts with dashboards, alarms, user feedback, etc.:
 - The real-world domain may change over time.
 - The software environment may change.
 - High profile special cases may fail.
 - There may be a change in business goals.

Monitoring

- It's important to monitor quality metrics and business impacts with dashboards, alarms, user feedback, etc.:
 - The real-world domain may change over time.
 - The software environment may change.
 - High profile special cases may fail.
 - There may be a change in business goals.



4:32

2x



Maintenance



- Performance deterioration may require new tuning:
 - Changing goals may require new metrics.
 - A changing domain may require changes to validation set.
 - Your validation set may be replaced over time to avoid overfitting.

Maintenance



- Performance deterioration may require new tuning:
 - Changing goals may require new metrics.
 - A changing domain may require changes to validation set.
 - Your validation set may be replaced over time to avoid overfitting.

Maintenance



- Performance deterioration may require new tuning:
 - Changing goals may require new metrics.
 - A changing domain may require changes to validation set.
 - Your validation set may be replaced over time to avoid overfitting.

Customer Obsession

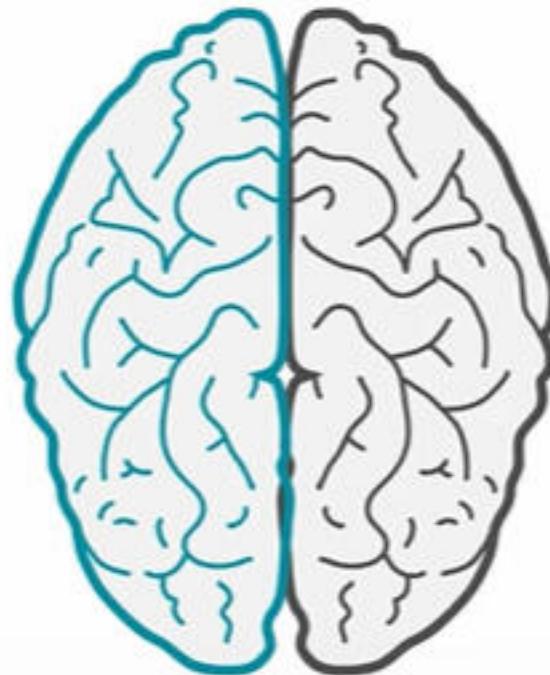


- Think carefully about impact on customer perception and trust.
- Give ML solutions the “creepiness sniff test” and “*The front page of a newspaper test.*”
- Provide explanations to customers.

Artificial Intelligence on AWS



Powerful **tools**
for all developers



to add **intelligence** to
your applications

Artificial Intelligence on AWS



Amazon SageMaker

Build, train, and deploy machine learning models at scale.



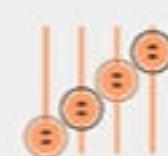
Pre-built
notebooks



Built-in, high
performance
algorithms



One-click
training



Hyperparameter
optimization



One-click
deployment



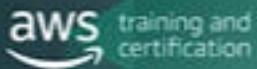
Fully managed
hosting with
auto-scaling

Build

Train

Deploy

Artificial Intelligence on AWS



Amazon SageMaker

Build, train, and deploy machine learning models at scale.



Pre-built
notebooks



Built-in, high
performance
algorithms



One-click
training



Hyperparameter
optimization



One-click
deployment



Fully managed
hosting with
auto-scaling

Build

Train

Deploy



Artificial Intelligence on AWS



Amazon SageMaker

Build, train, and deploy machine learning models at scale.



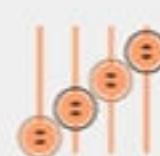
Pre-built
notebooks



Built-in, high
performance
algorithms



One-click
training



Hyperparameter
optimization



One-click
deployment



Fully managed
hosting with
auto-scaling

Build

Train

Deploy

Artificial Intelligence on AWS



Amazon Rekognition Image

Deep learning-based **image** analysis

Amazon Rekognition Video

Deep learning-based **video** analysis

Built on technology used by
Amazon Prime Photos
to analyze billions of images daily

Artificial Intelligence on AWS



Amazon Lex

Build chatbots to engage customers

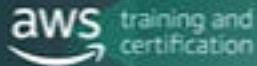
Uses the same technology as
Amazon Alexa
to provide advanced
deep learning functionalities
Of ASR and NLU

Amazon Polly

Natural sounding text to speech, which includes:

- More than two dozen languages
- Wide variety of natural-sounding voices

Artificial Intelligence on AWS



Amazon Polly

Natural sounding text to speech, which includes:

- More than two dozen languages
- Wide variety of natural-sounding voices



Amazon Comprehend

Natural language processing (NLP) service that enables you to:

- Discover insights and relationships in text
- Identify language based on the text
- Extract key phrases, places, people, brands, or events
- Understand how positive or negative the text is
- Automatically organizes a collection of text files by topic

Amazon Translate

Neural machine translation service that enables you to:

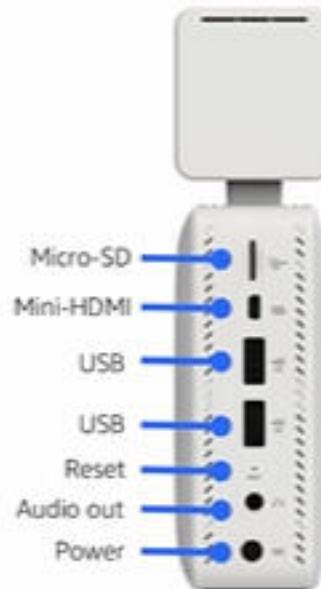
- Perform fluent translation of text
- Localize content for international users
- Easily translate large volumes of text efficiently

AWS DeepLens



HD video camera

Custom-designed
deep learning
inference engine



Micro-SD

Mini-HDMI

USB

USB

Reset

Audio out

Power



HD video camera with
on-board compute
optimized for deep
learning



From unboxing to first
inference in <10
minutes



Integrates with Amazon
SageMaker and AWS
Lambda



Tutorials, examples, demos,
and pre-built models

- **AWS Glue:** Data integration service for managing ETL jobs
- **Deep Scalable Sparse Tensor Network Engine (DSSTNE):** Neural network engine