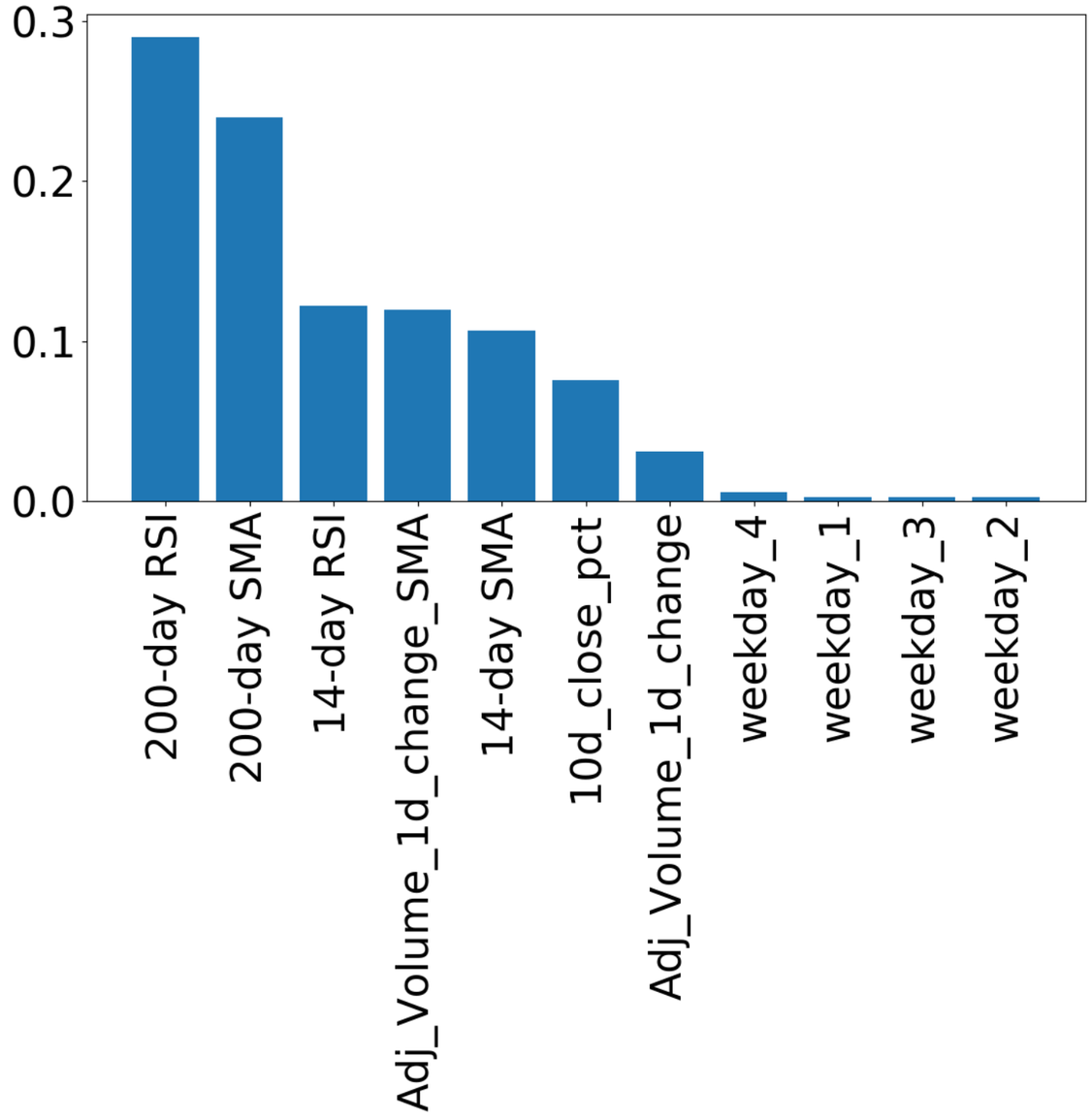
Scaling data and KNN Regression

MACHINE LEARNING FOR FINANCE IN PYTHON

Nathan George
Data Science Professor

datacamp


0.3
0.2
0.1
0.0
200-day RSI
200-day SMA
14-day RSI
Adj_Volume_1d_change_SMA
14-day SMA
10d_close_pct
Adj_Volume_1d_change
weekday_4
weekday_1
weekday_3
weekday_2

Feature selection: remove weekdays

```
print(feature_names)
```

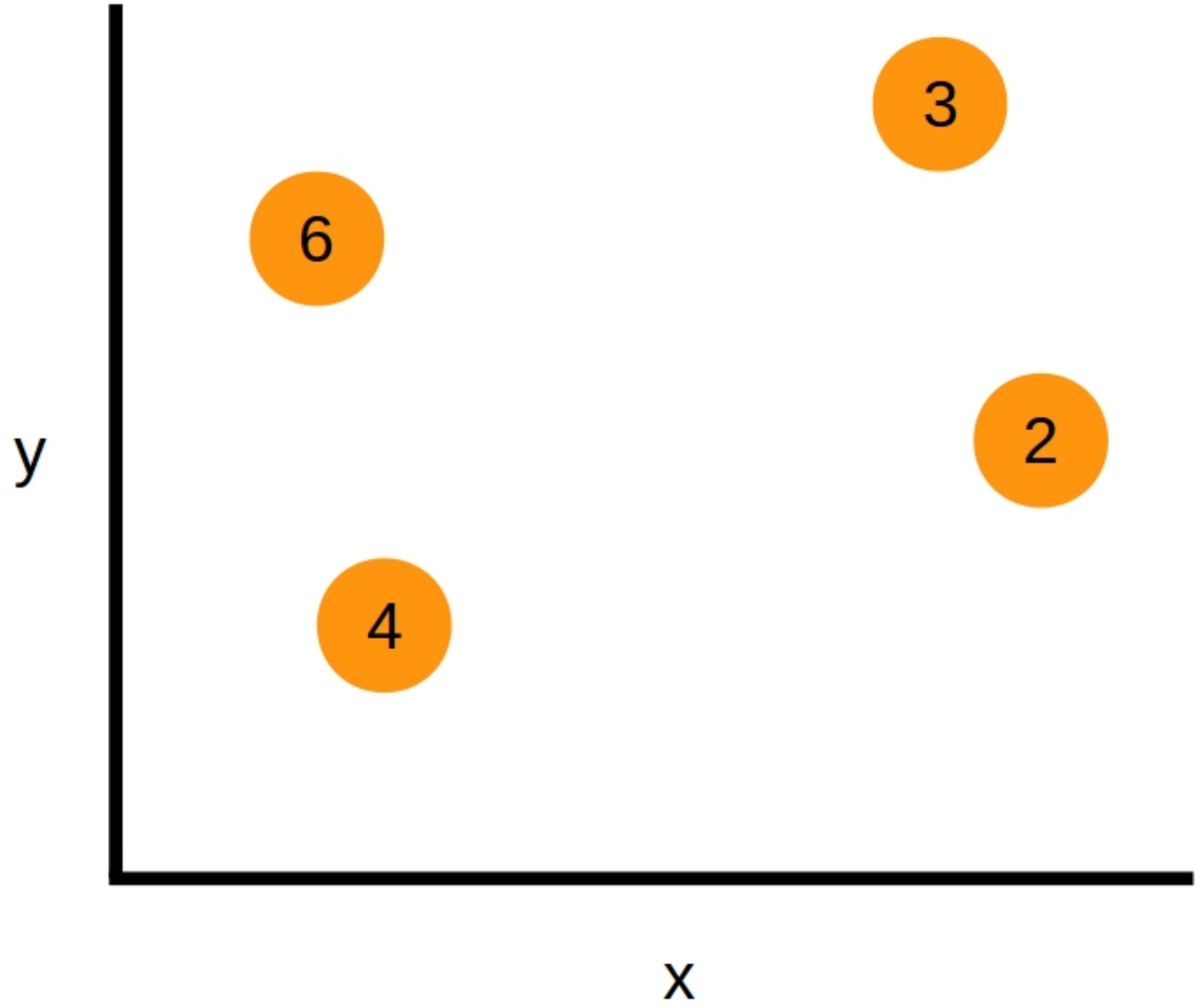
```
['10d_close_pct',
 '14-day SMA',
 '14-day RSI',
 '200-day SMA',
 '200-day RSI',
 'Adj_Volume_1d_change',
 'Adj_Volume_1d_change_SMA',
 'weekday_1',
 'weekday_2',
 'weekday_3',
 'weekday_4']
```

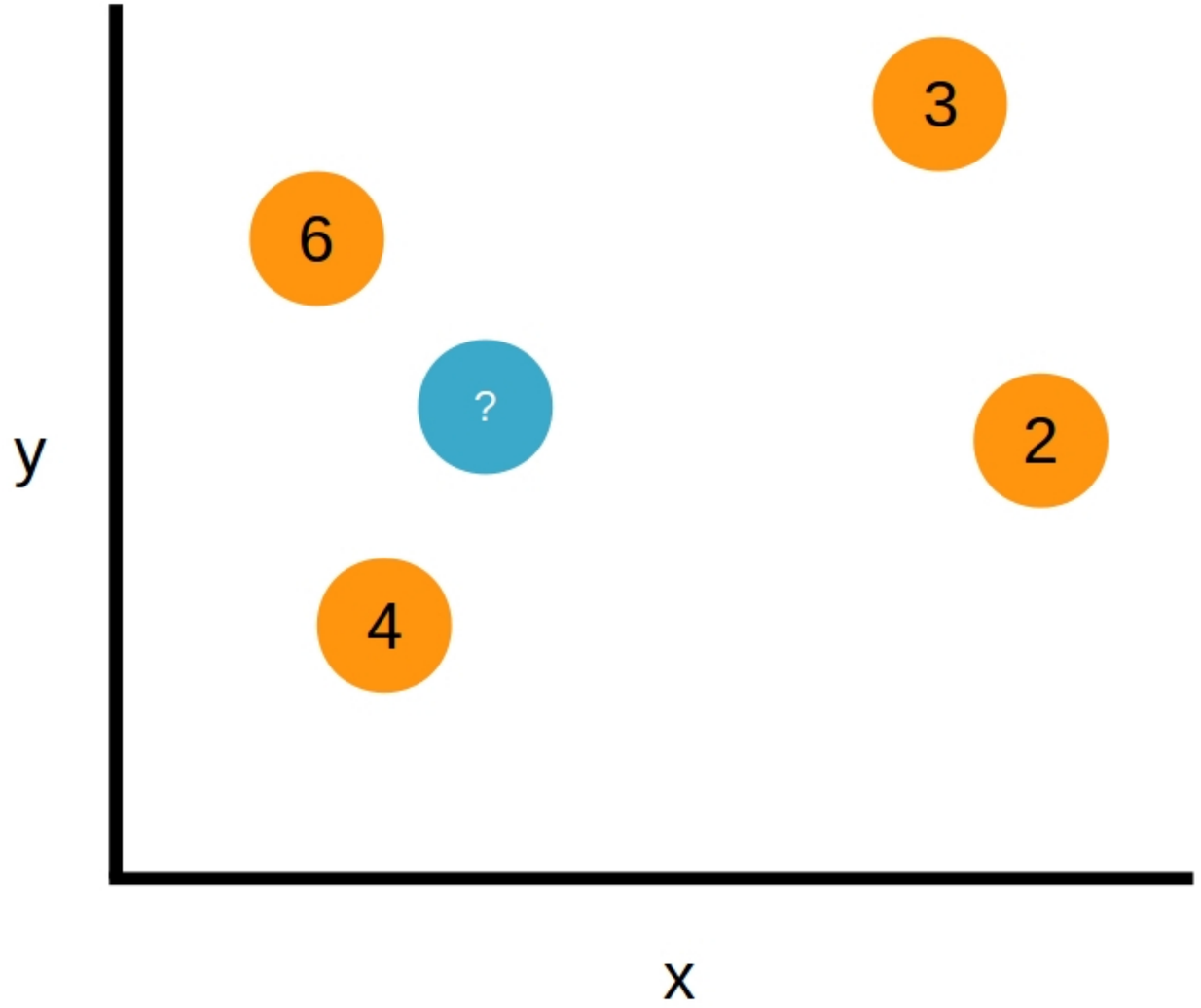
```
print(feature_names[:-4])
```

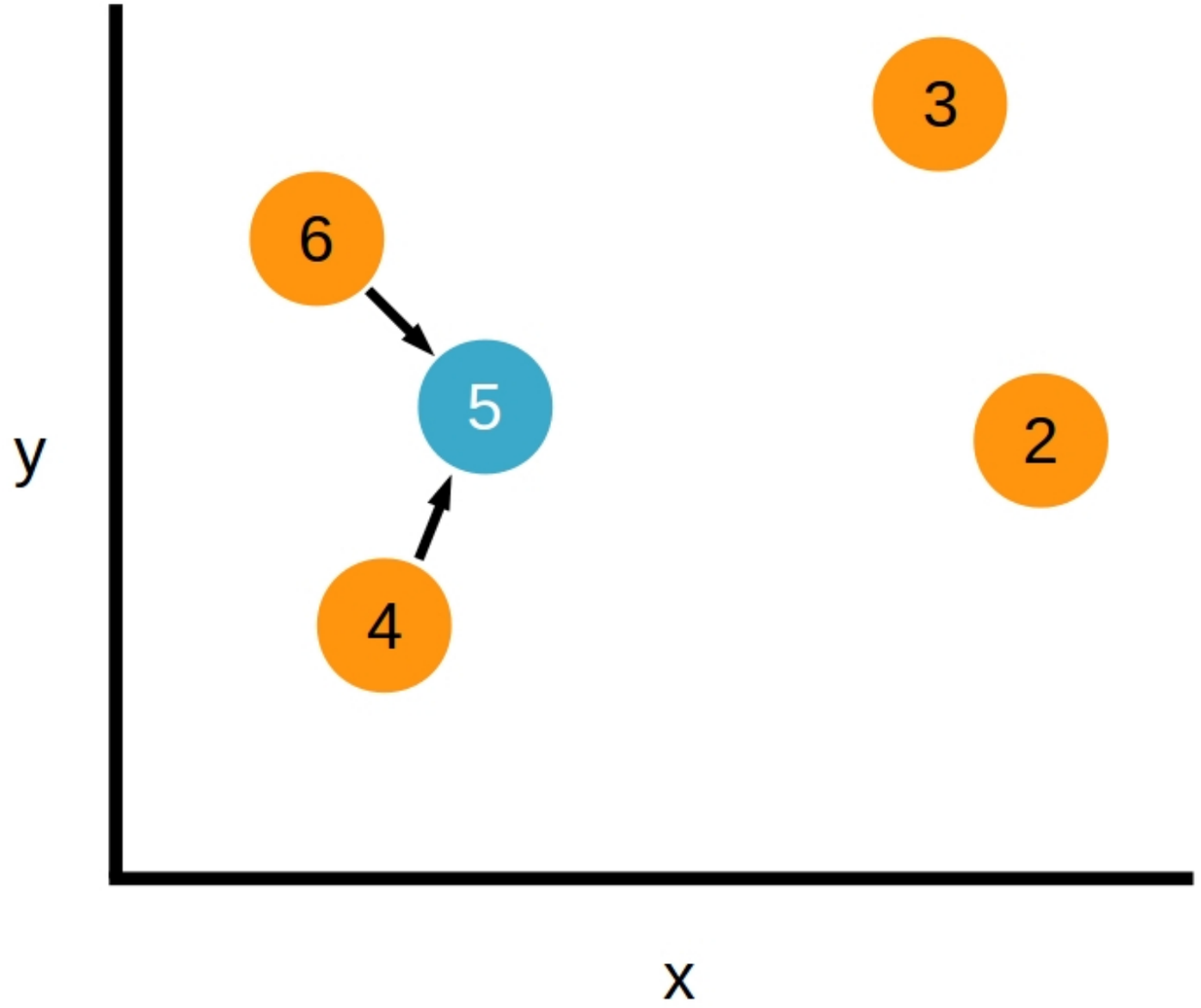
```
['10d_close_pct',
 '14-day SMA',
 '14-day RSI',
 '200-day SMA',
 '200-day RSI',
 'Adj_Volume_1d_change',
 'Adj_Volume_1d_change_SMA']
```

Remove weekdays

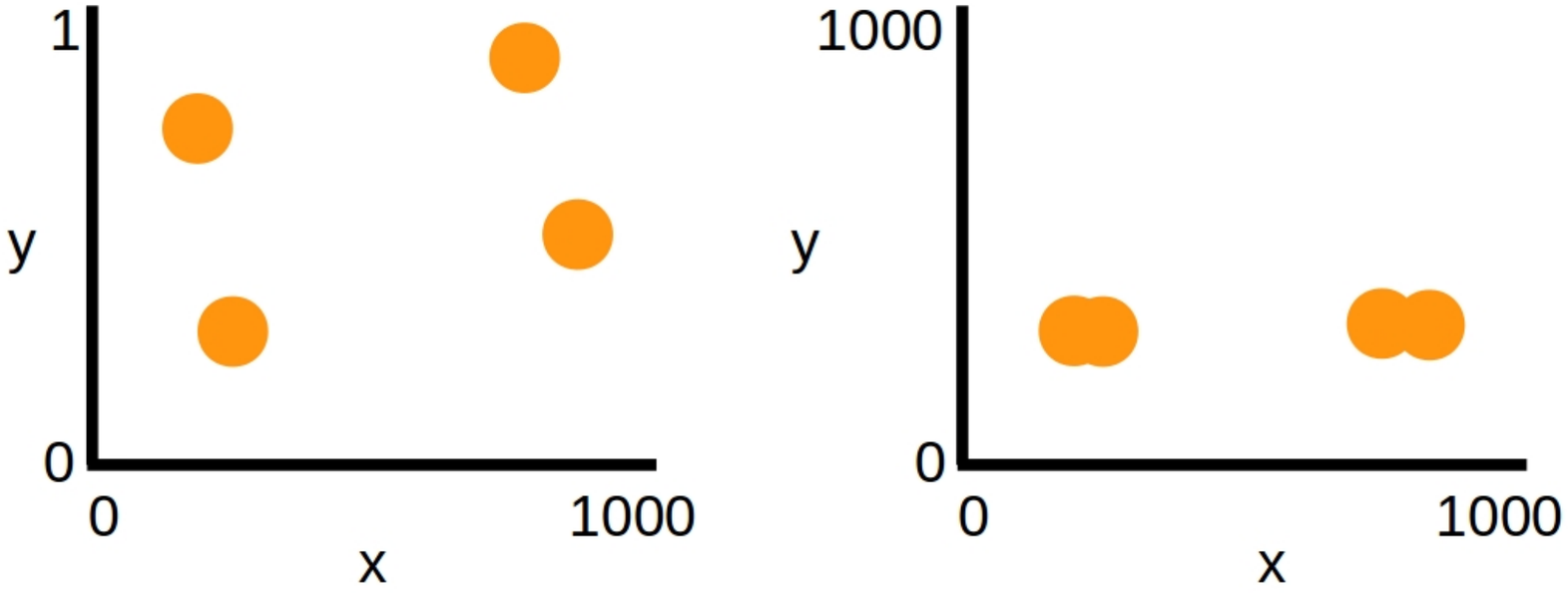
```
train_features = train_features.iloc[:, :-4]
test_features = test_features.iloc[:, :-4]
```


3
6
2
y
4
x


3
6
?
2
4
y
x


3
6
5
2
4
y
x

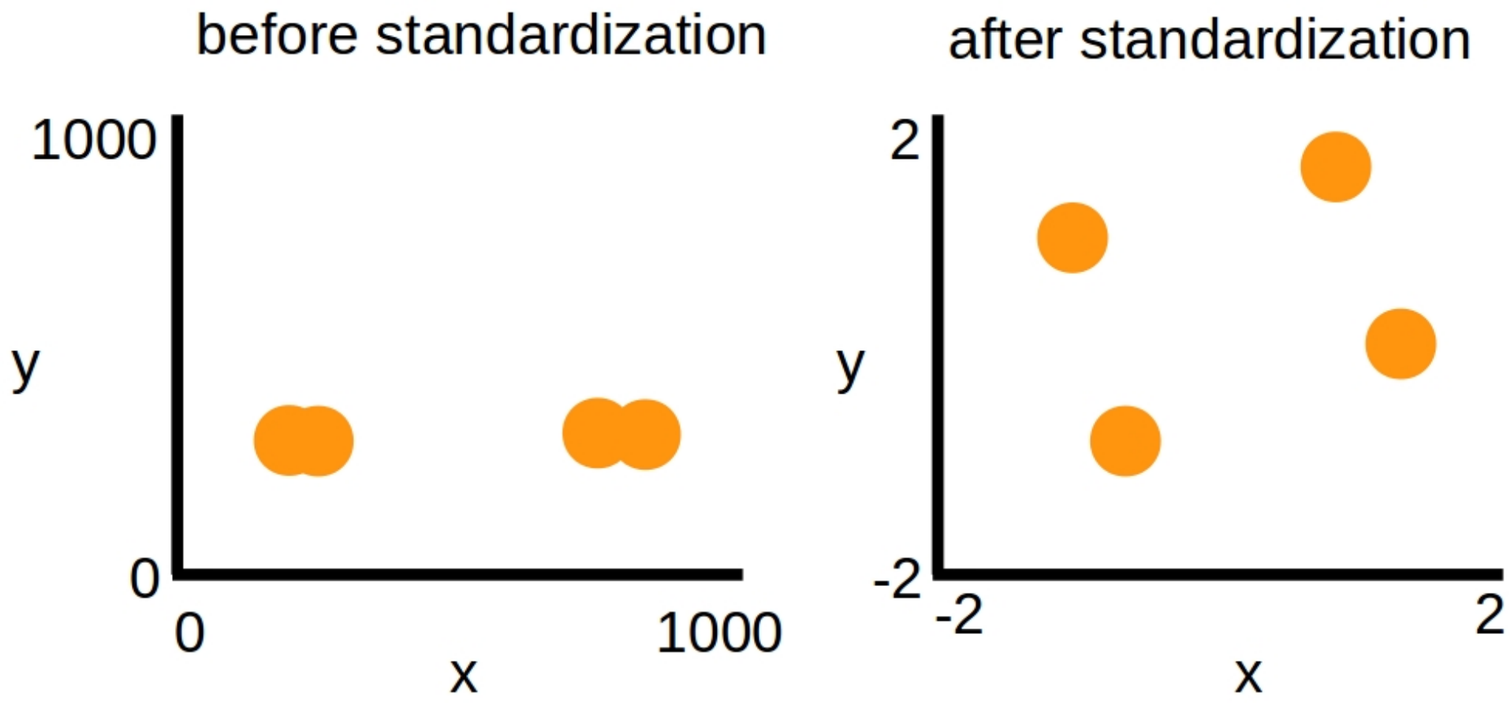
$$D(A,B) = \sum_i \left(\left| (a_i - b_i) \right| \right)^{(1/p)}$$


1
y
0
0
1000
x
1000
y
0
0
1000
x

Scaling options

Scaling options:

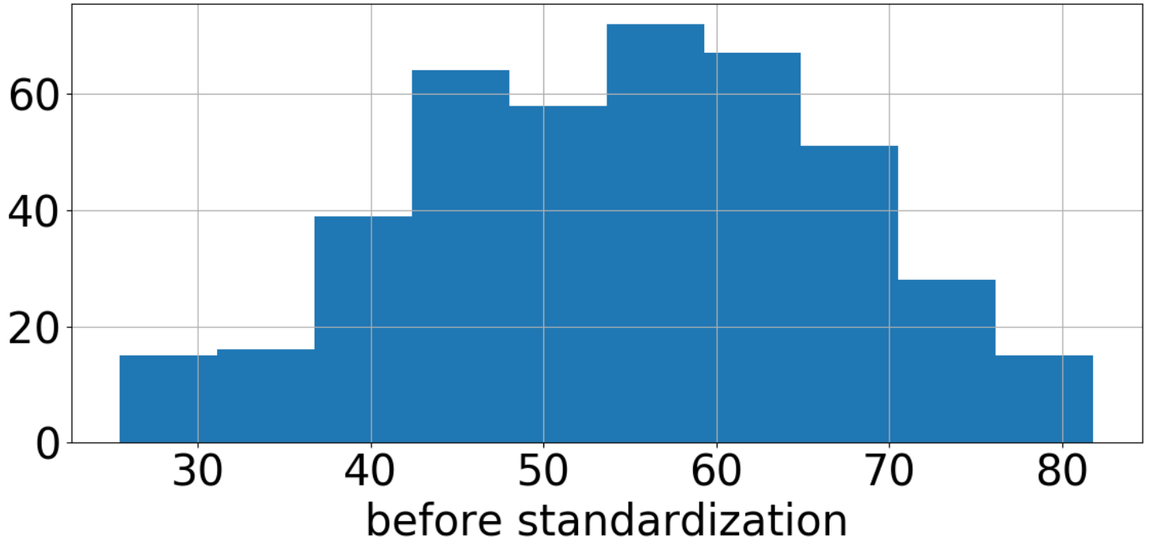
- min-max
- standardization
- median-MAD
- map to arbitrary function (e.g. sigmoid, tanh)

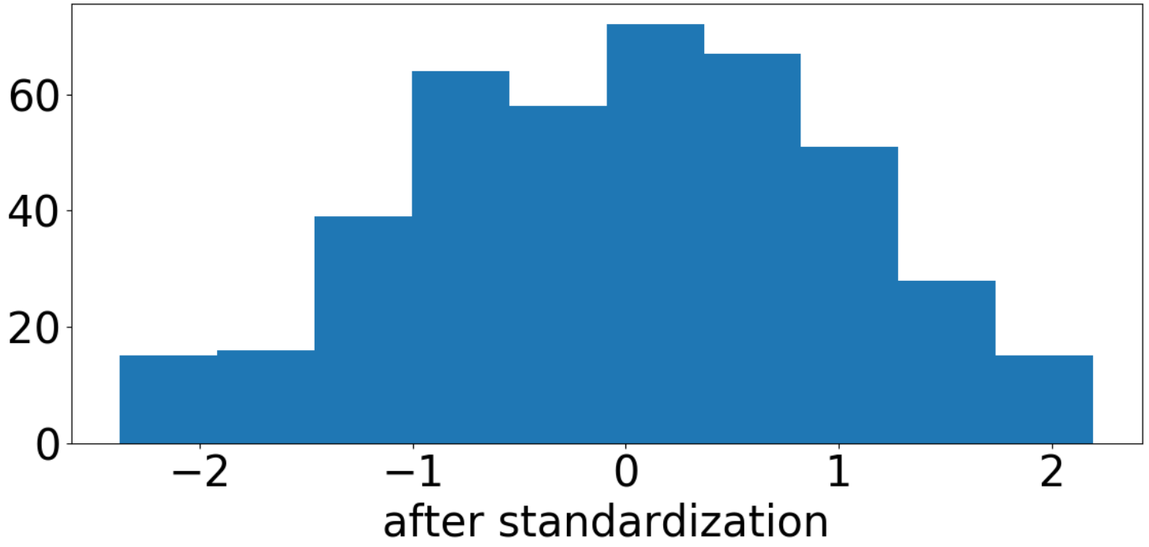

before standardization
1000
y
0
0
1000
x
after standardization
2
y
-2
-2
2
x

sklearn's scale

```
from sklearn.preprocessing import scale

sc = scale()
scaled_train_features = sc.fit_transform(train_features)
scaled_test_features = sc.transform(test_features)
```


60
40
20
0
30
40
50
60
70
80
before standardization


60
40
20
0
−2
−1
0
1
2
after standardization

Making subplots

```
# create figure and list containing axes
f, ax = plt.subplots(nrows=2, ncols=1)
# plot histograms of before and after scaling
train_features.iloc[:, 2].hist(ax=ax[0])
ax[1].hist(scaled_train_features[:, 2])
plt.show()
```

Scale data and use KNN!

MACHINE LEARNING FOR FINANCE IN PYTHON

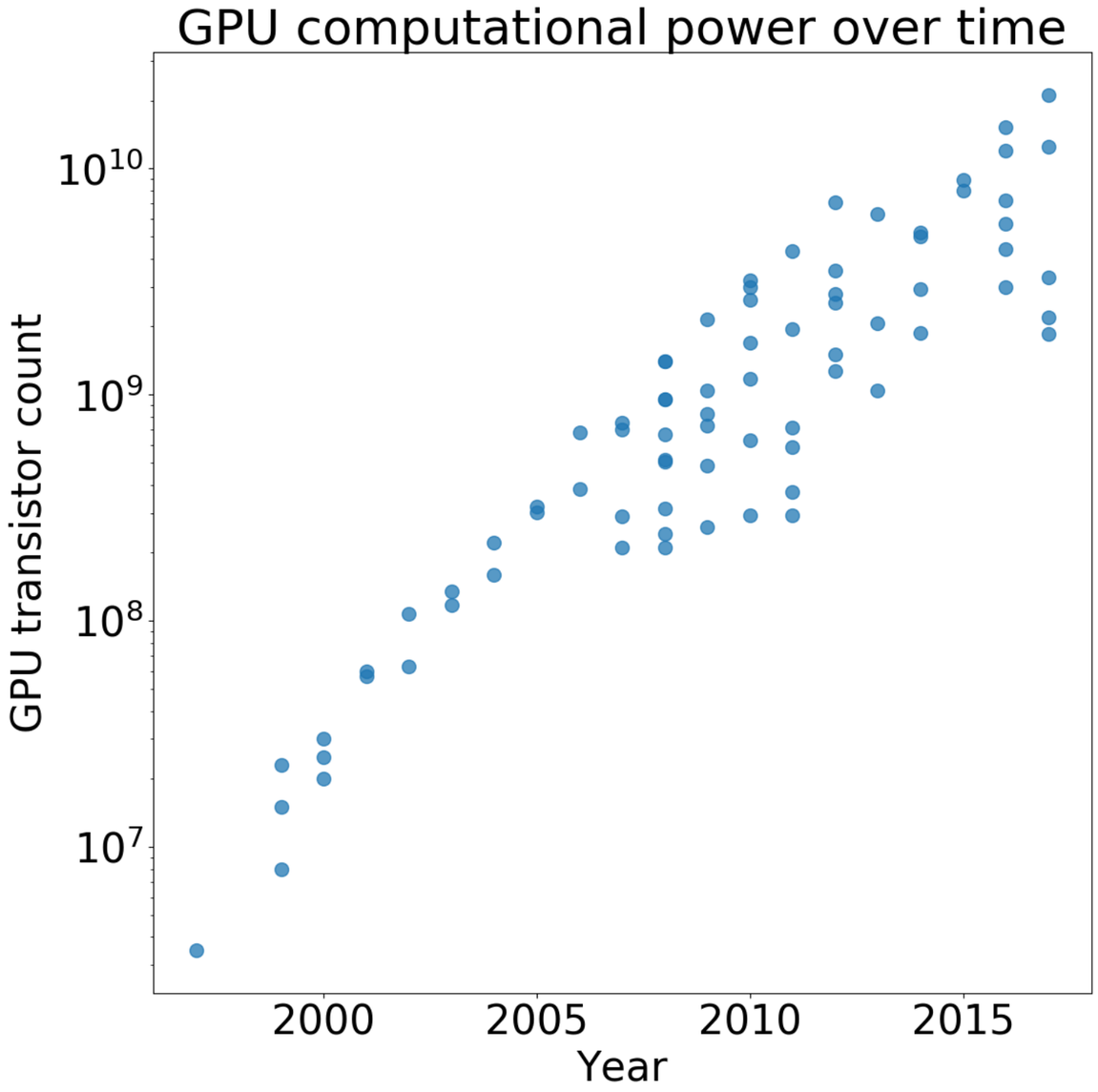
datacamp

Neural Networks

MACHINE LEARNING FOR FINANCE IN PYTHON

Nathan George
Data Science Professor

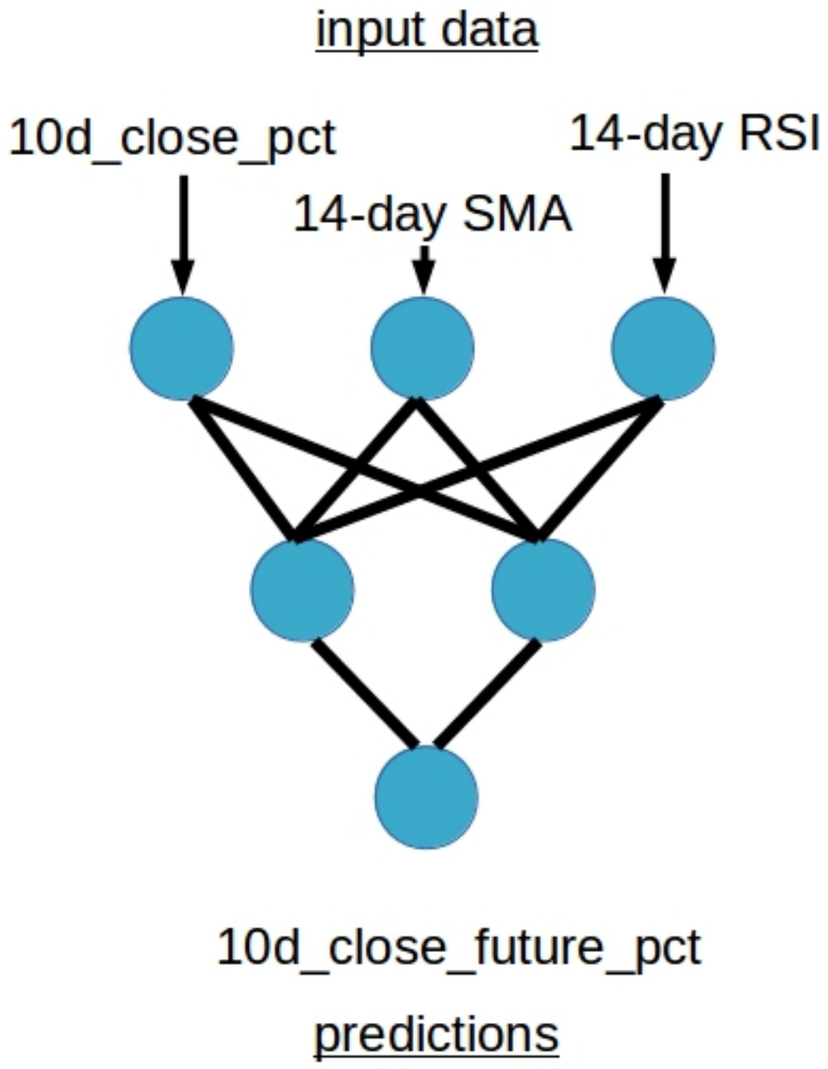
datacamp

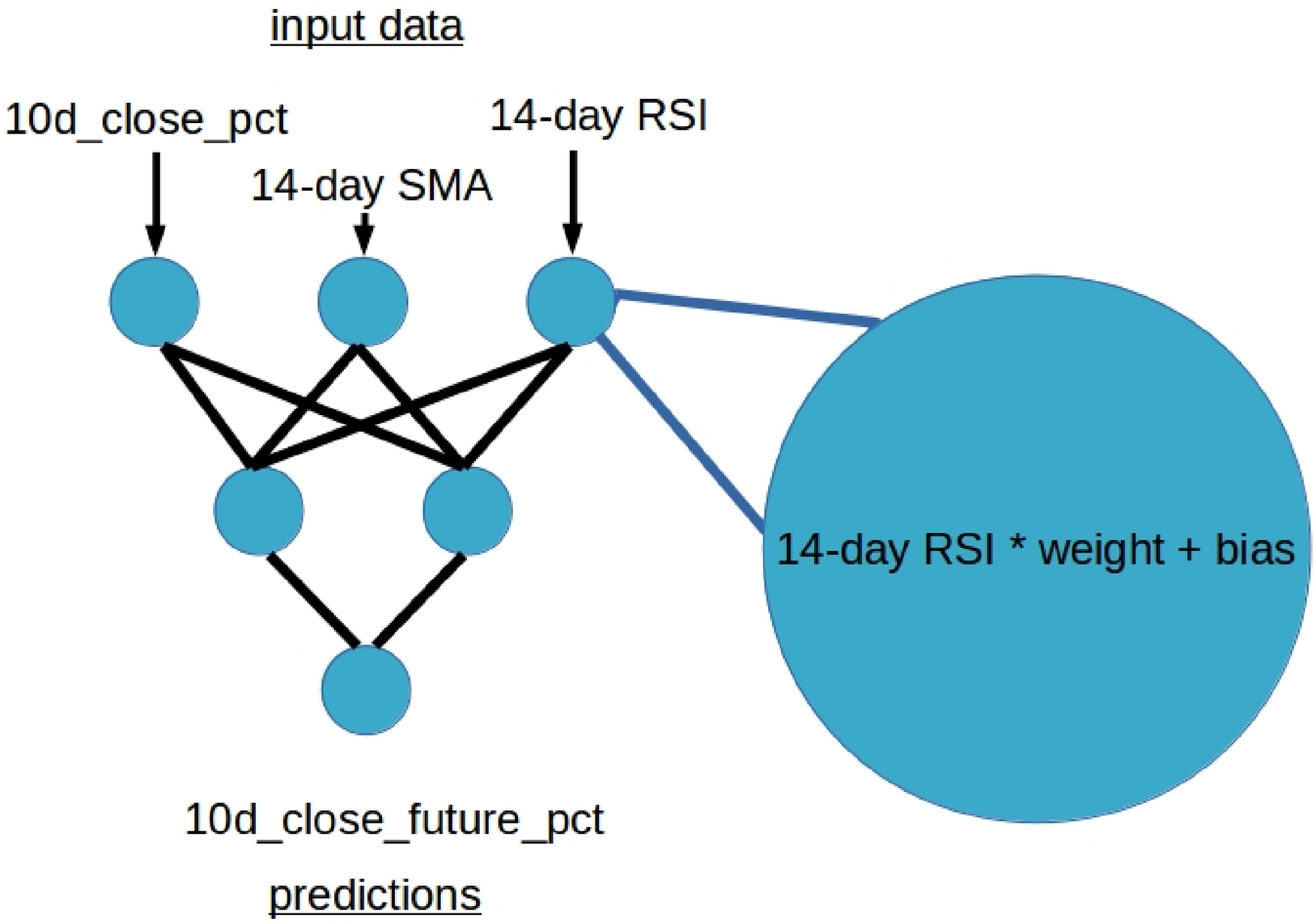

GPU computational power over time
GPU transistor count
10^{10}
10^{9}
10^{8}
10^{7}
2000
2005
2010
2015
Year

Neural networks have potential

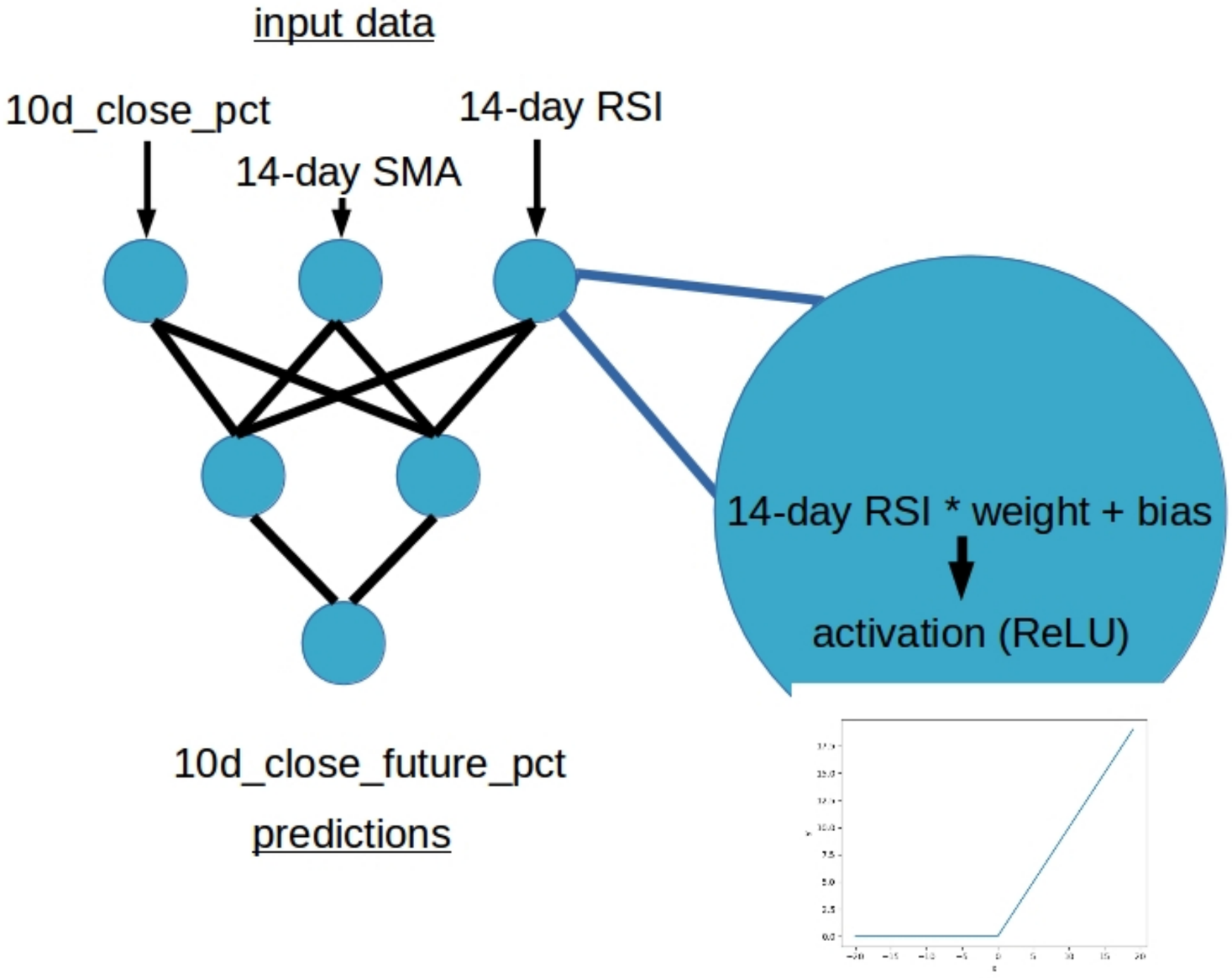
Neural nets have:

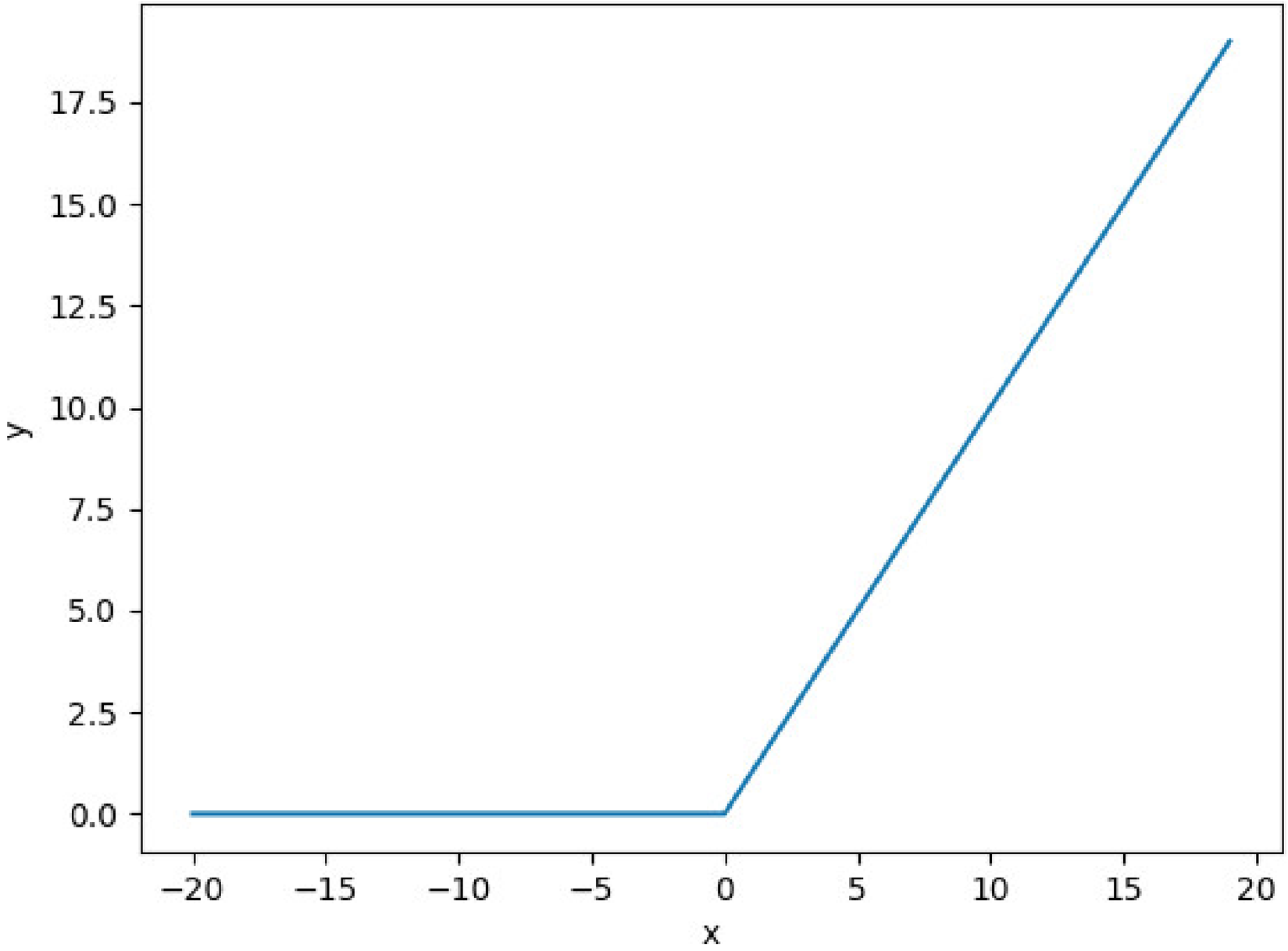
- non-linearity
- variable interactions
- customizability

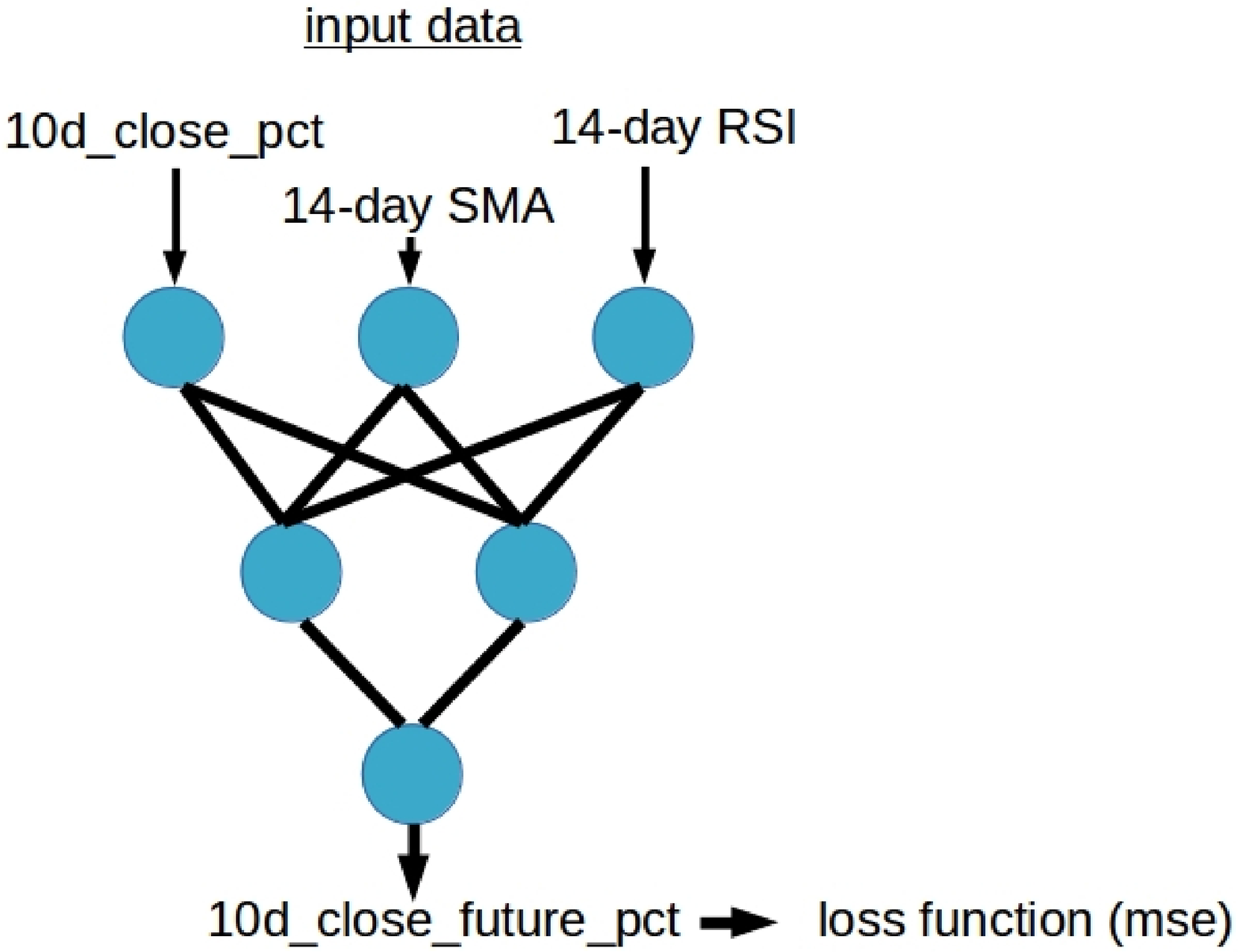

input data
10d_close_pct
14-day SMA
14-day RSI
10d_close_future_pct
predictions

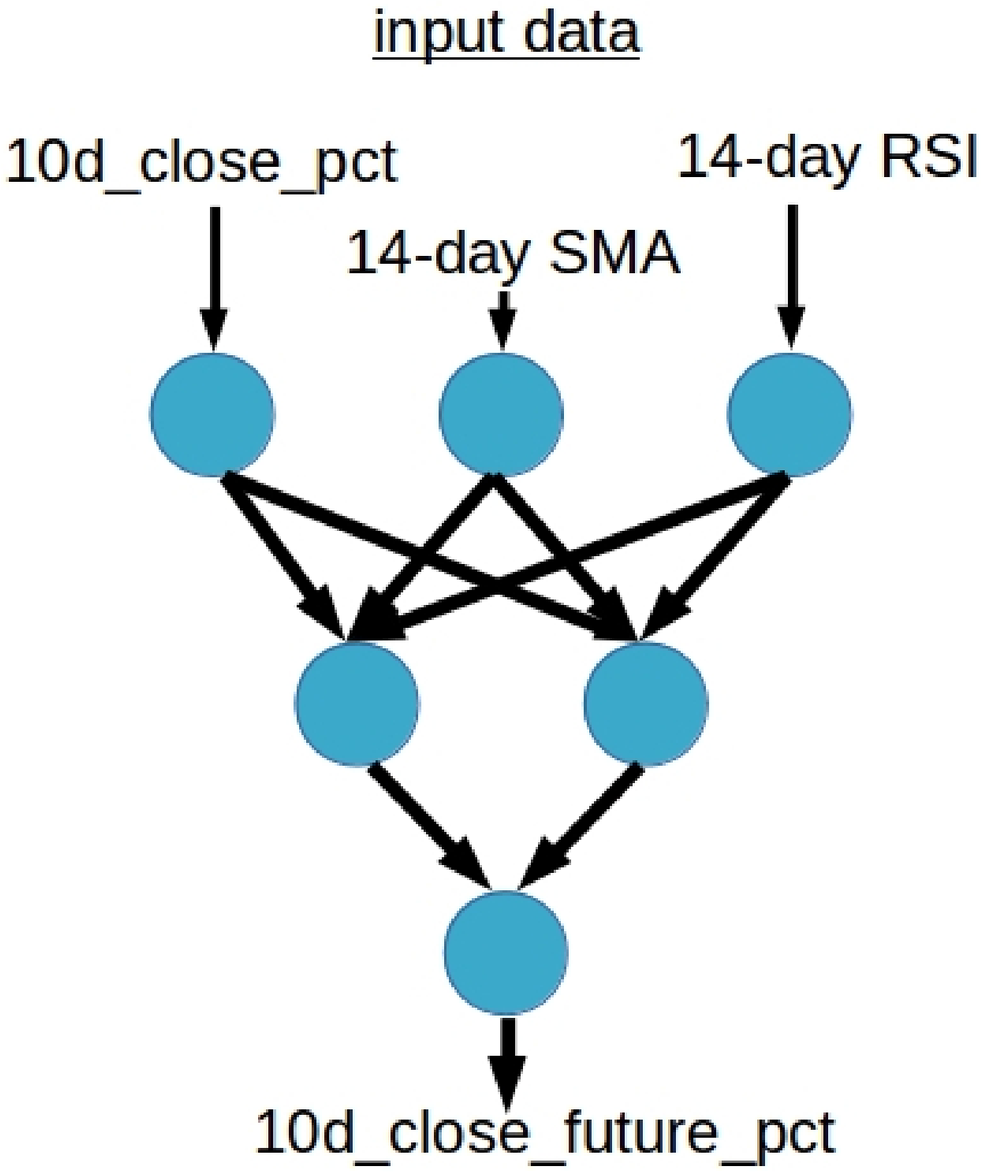

input data
10d_close_pct
14-day SMA
14-day RSI
14-day RSI * weight + bias
10d_close_future_pct
predictions

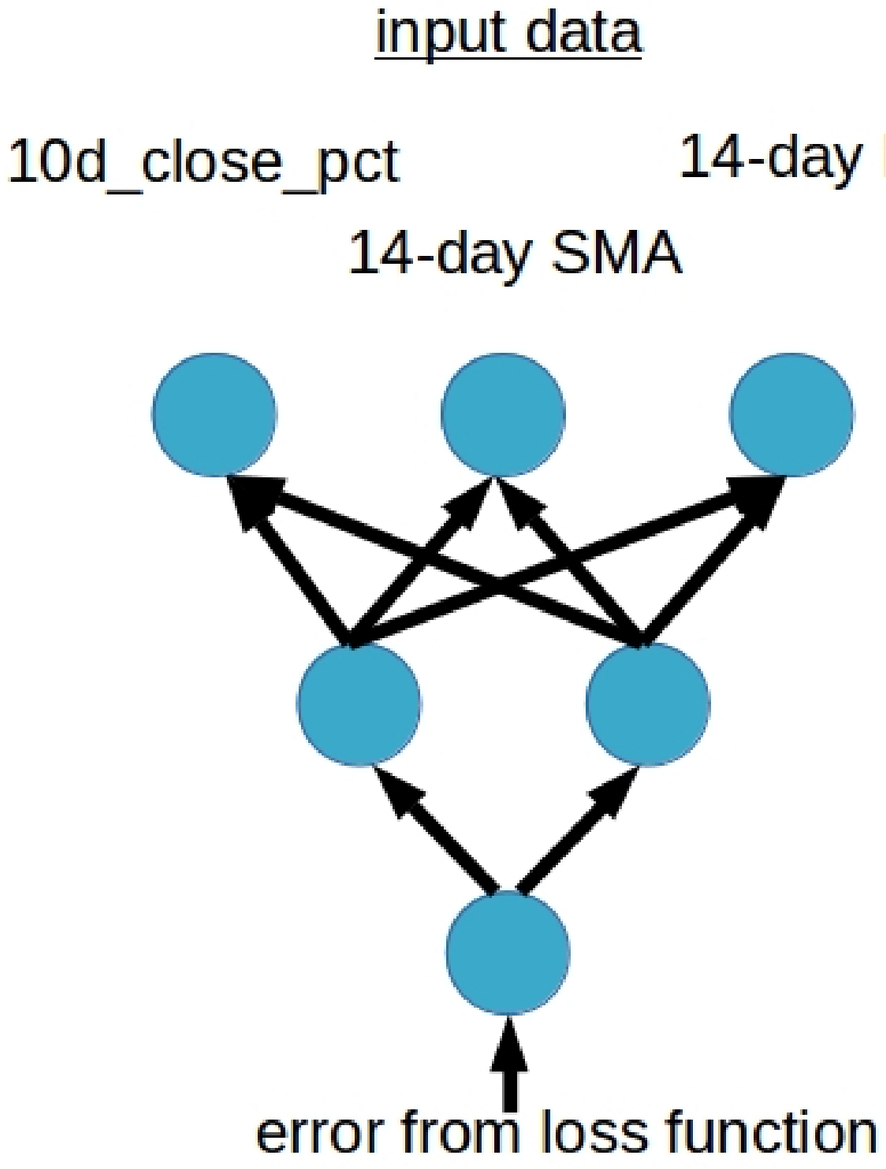
$$\sum_i w_i x_i + b$$

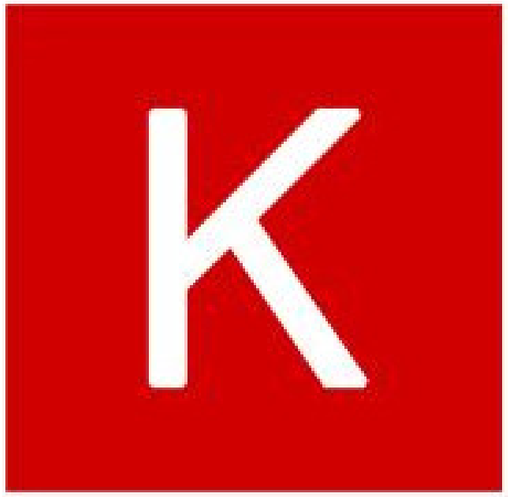

input data
10d_close_pct
14-day SMA
14-day RSI
14-day RSI * weight + bias
activation (ReLU)
10d_close_future_pct
predictions


17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
y
−20
−15
−10
−5
0
5
10
15
20
x


input data
10d_close_pct
14-day SMA
14-day RSI
10d_close_future_pct
loss function (mse)


input data
10d_close_pct
14-day SMA
14-day RSI
10d_close_future_pct


input data
10d_close_pct
14-day RSI
14-day SMA
error from loss function



Keras

Implementing a neural net with keras

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Implementing a neural net with keras

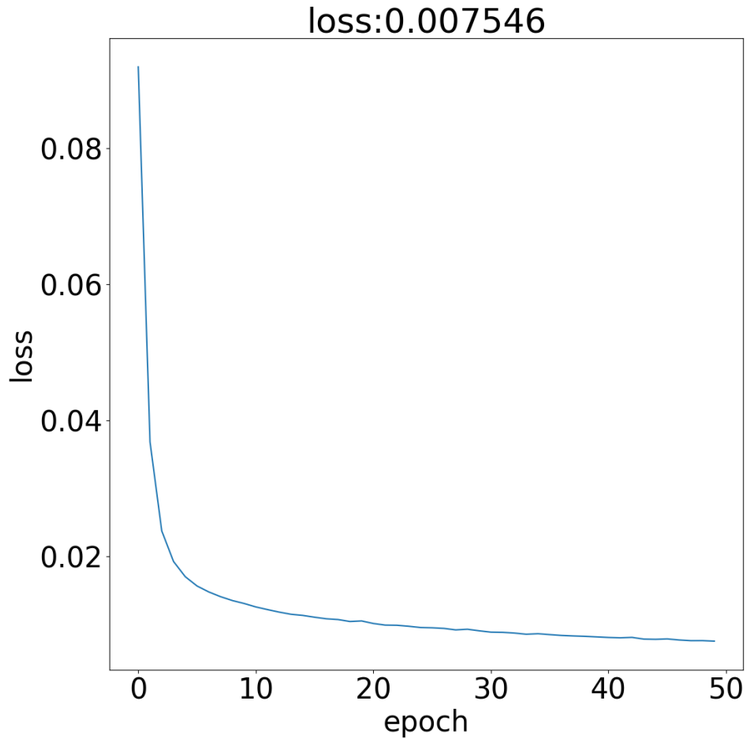
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(50,
                input_dim=scaled_train_features.shape[1],
                activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Fitting the model

```
model.compile(optimizer='adam', loss='mse')
history = model.fit(scaled_train_features,
                    train_targets,
                    epochs=50)
```

```
plt.plot(history.history['loss'])
plt.title('loss:' + str(round(history.history['loss'][-1], 6)))
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



Checking out performance

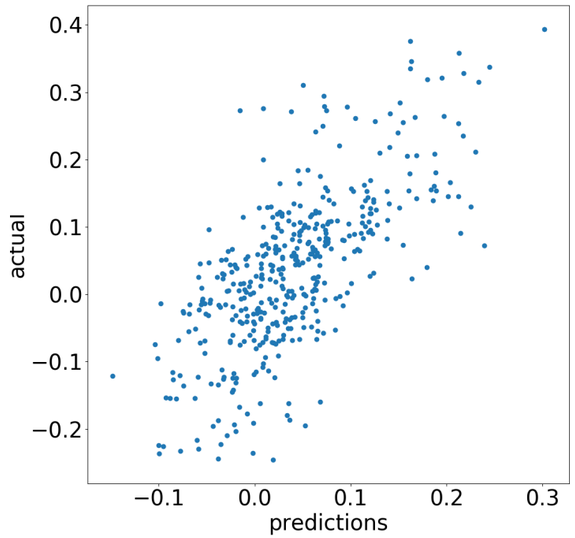
```
from sklearn.metrics import r2_score


# calculate R^2 score
train_preds = model.predict(scaled_train_features)
print(r2_score(train_targets, train_preds))
```

```
0.4771387560719418
```

Plot performance

```
# plot predictions vs actual
plt.scatter(train_preds, train_targets)
plt.xlabel('predictions')
plt.ylabel('actual')
plt.show()
```



Make a neural net!

MACHINE LEARNING FOR FINANCE IN PYTHON

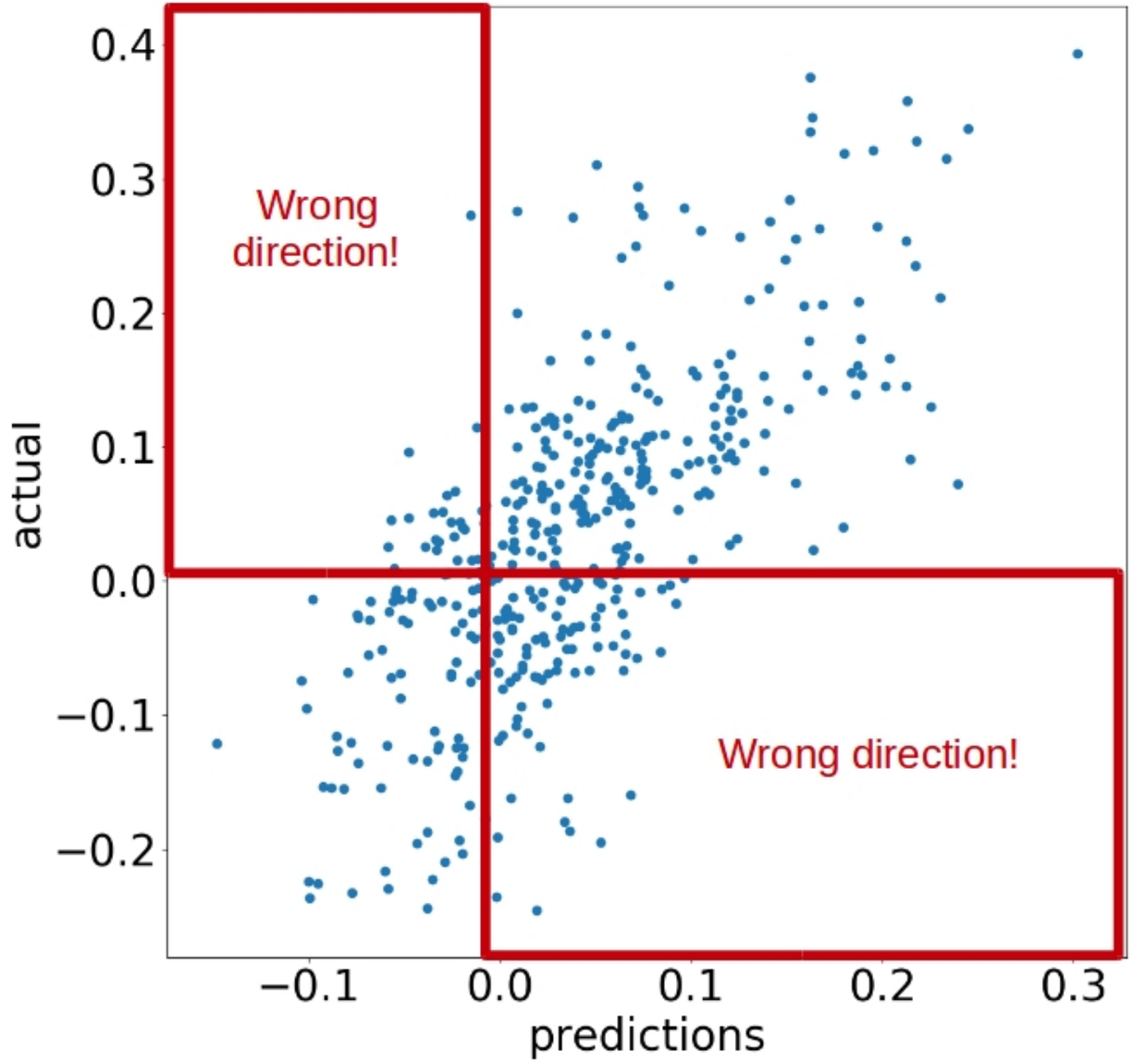
datacamp

Custom loss functions

MACHINE LEARNING FOR FINANCE IN PYTHON

Nathan George
Data Science Professor

datacamp


Wrong direction!
Wrong direction!
actual
predictions
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.1
0.0
0.1
0.2
0.3

MSE with directional penalty

If prediction and target direction match:

- $\sum(y - \hat{y})^2$

If not:

- $\sum(y - \hat{y})^2 * \text{penalty}$

Implementing custom loss functions

```
import tensorflow as tf
```

Creating a function

```
import tensorflow as tf
# create loss function
def mean_squared_error(y_true, y_pred):
```

Mean squared error loss

```
import tensorflow as tf


# create loss function
def mean_squared_error(y_true, y_pred):
    loss = tf.square(y_true - y_pred)
    return tf.reduce_mean(loss, axis=-1)
```

Add custom loss to keras

```
import tensorflow as tf

# create loss function
def mean_squared_error(y_true, y_pred):
    loss = tf.square(y_true - y_pred)
    return tf.reduce_mean(loss, axis=-1)
# enable use of loss with keras
import keras.losses
keras.losses.mean_squared_error = mean_squared_error
```

```
# fit the model with our mse loss function
model.compile(optimizer='adam', loss=mean_squared_error)
history = model.fit(scaled_train_features, train_targets, epochs=50)
```

Checking for correct direction

```
tf.less(y_true * y_pred, 0)
```

Correct direction:

- neg * neg = pos
- pos * pos = pos

Wrong direction:

- neg * pos = neg
- pos * neg = neg

Using tf.where()

```
# create loss function
def sign_penalty(y_true, y_pred):
    penalty = 10.
    loss = tf.where(tf.less(y_true * y_pred, 0),
                    penalty * tf.square(y_true - y_pred),
                    tf.square(y_true - y_pred))
```

Tying it together

```
# create loss function
def sign_penalty(y_true, y_pred):
    penalty = 100.
    loss = tf.where(tf.less(y_true * y_pred, 0),
                    penalty * tf.square(y_true - y_pred),
                    tf.square(y_true - y_pred))

    return tf.reduce_mean(loss, axis=-1)
# enable use of loss with keras
keras.losses.sign_penalty = sign_penalty
```

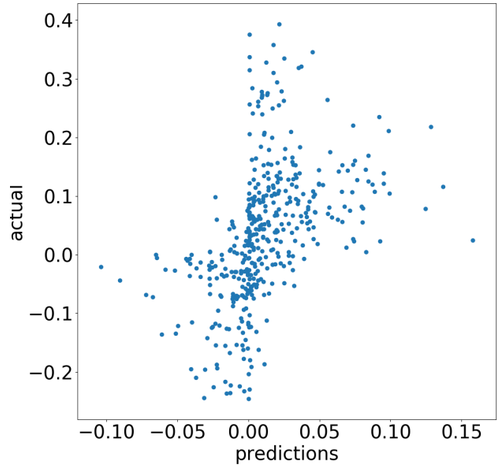
Using the custom loss

```
# create the model
model = Sequential()
model.add(Dense(50,
                input_dim=scaled_train_features.shape[1],
                activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='linear'))
```

```
# fit the model with our custom 'sign_penalty' loss function
model.compile(optimizer='adam', loss=sign_penalty)
history = model.fit(scaled_train_features, train_targets, epochs=50)
```

The bow-tie shape

```
train_preds = model.predict(scaled_train_features)
# scatter the predictions vs actual
plt.scatter(train_preds, train_targets)
plt.xlabel('predictions')
plt.ylabel('actual')
plt.show()
```



Create your own loss function!

MACHINE LEARNING FOR FINANCE IN PYTHON

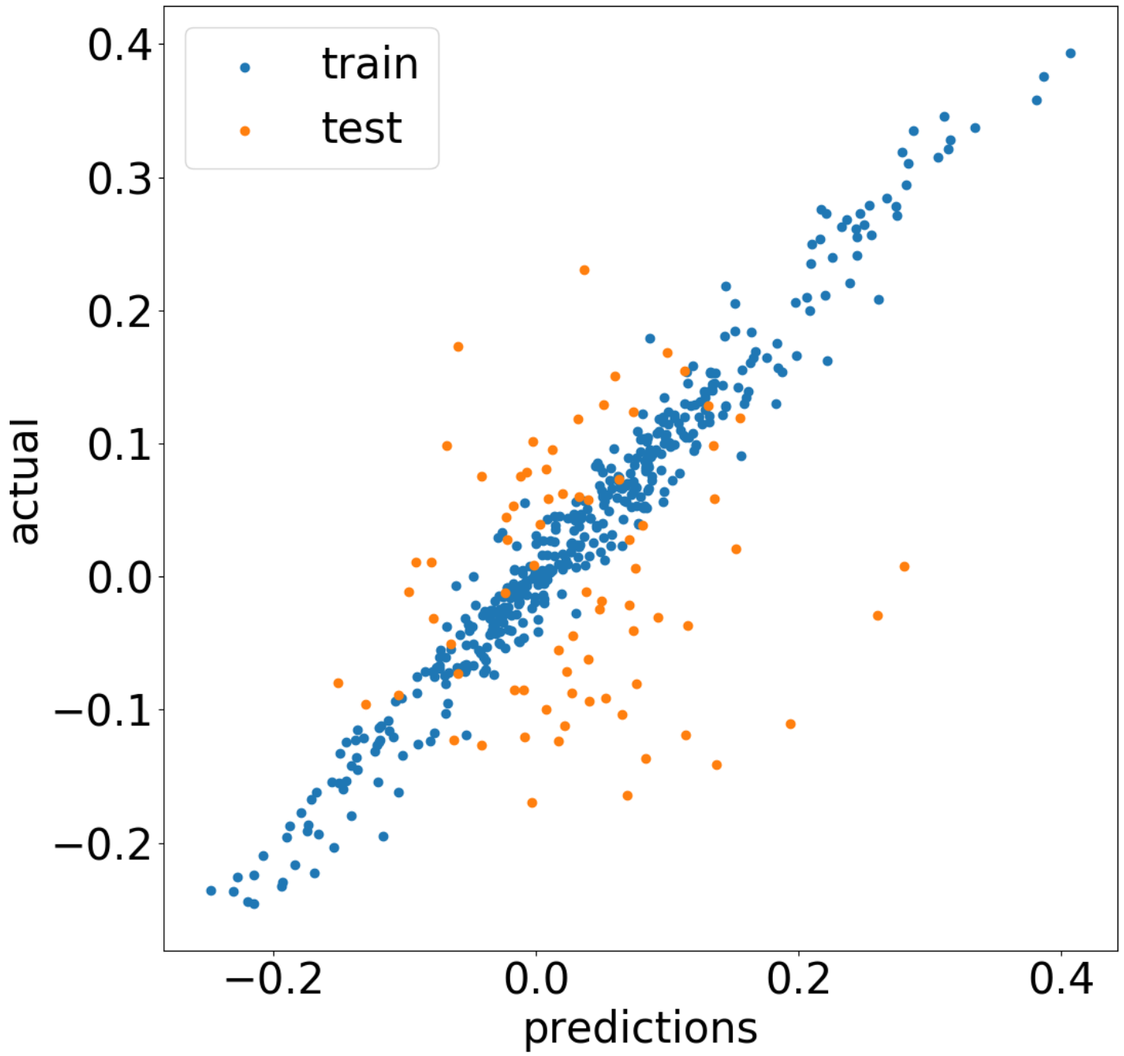
datacamp

Overfitting and ensembling

MACHINE LEARNING FOR FINANCE IN PYTHON

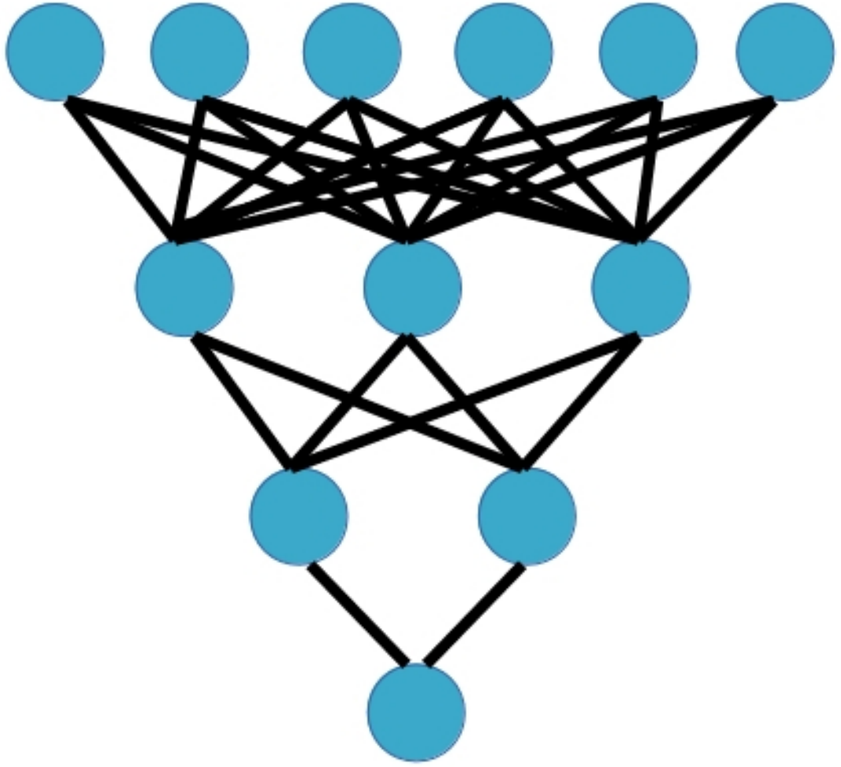
Nathan George
Data Science Professor

datacamp

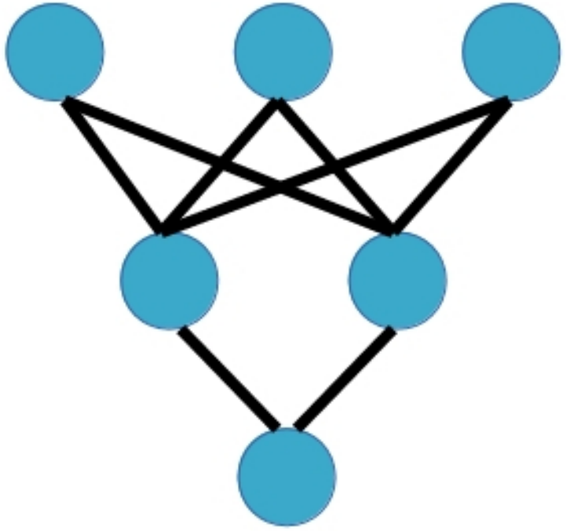

train
test
actual
predictions
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.2
0.0
0.2
0.4

Simplify your model

Complex net overfits



Simpler net prevents overfitting



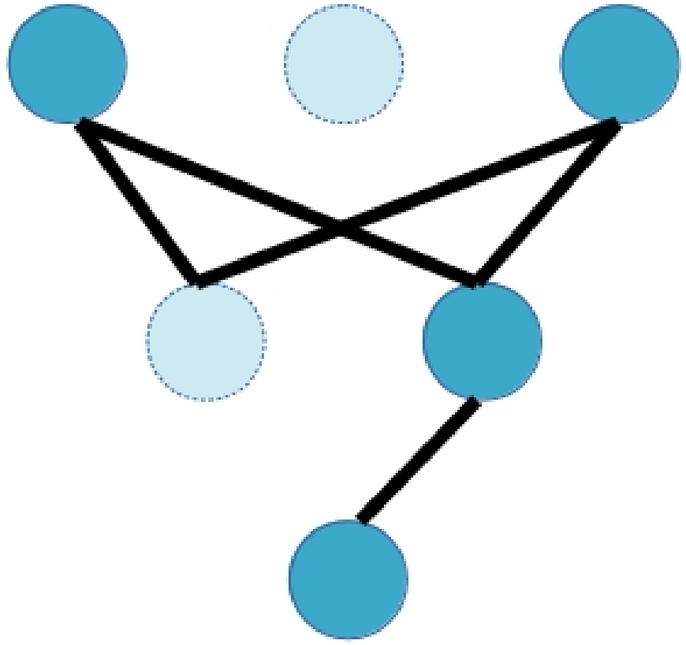
Neural network options

Options to combat overfitting:

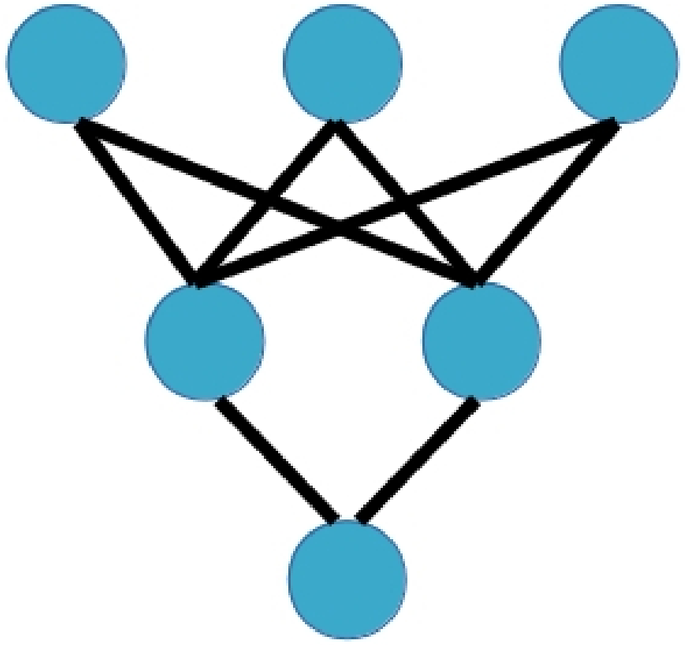
- Decrease number of nodes
- Use L1/L2 regulariation
- Dropout
- Autoencoder architecture
- Early stopping
- Adding noise to data
- Max norm constraints
- Ensembling

Dropout

33% dropout



no dropout



Dropout in keras

```
from keras.layers import Dense, Dropout
model = Sequential()
model.add(Dense(500,
                input_dim=scaled_train_features.shape[1],
                activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Test set comparison

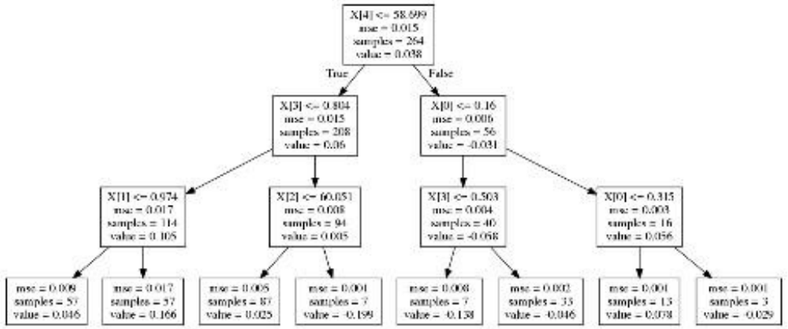
R^2 values on AMD without dropout:

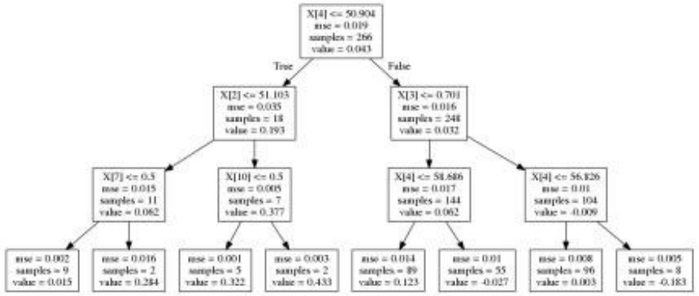
- train: 0.91
- test: -0.72

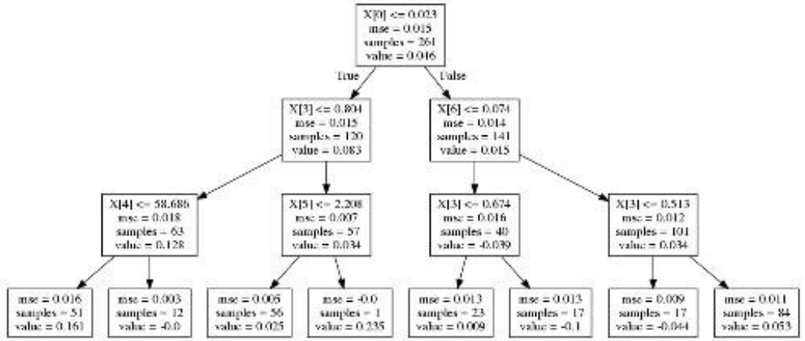
With dropout:

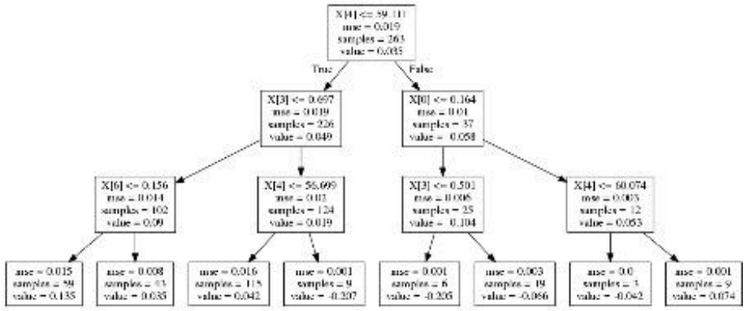
- train: 0.46
- test: -0.22

Ensembling


X[4] <= 58.699
mse = 0.015
samples = 264
value = 0.038
True
False
X[3] <= 0.804
mse = 0.015
samples = 208
value = 0.06
X[0] <= 0.16
mse = 0.006
samples = 56
value = -0.031
X[1] <= 0.974
mse = 0.017
samples = 114
value = 0.105
X[2] <= 60.051
mse = 0.008
samples = 94
value = 0.005
X[3] <= 0.503
mse = 0.004
samples = 40
value = -0.068
X[6] <= 0.315
mse = 0.003
samples = 16
value = 0.056
mse = 0.008
samples = 57
value = 0.046
mse = 0.017
samples = 57
value = 0.166
mse = 0.005
samples = 87
value = 0.025
mse = 0.001
samples = 7
value = -0.199
mse = 0.008
samples = 7
value = -0.138
mse = 0.002
samples = 33
value = -0.046
mse = 0.001
samples = 13
value = 0.038
mse = 0.001
samples = 3
value = -0.029


X[4] <= 50.904
mse = 0.019
samples = 266
value = 0.043
True
False
X[2] <= 51.103
mse = 0.035
samples = 18
value = 0.193
X[3] <= 0.701
mse = 0.016
samples = 248
value = 0.032
X[7] <= 0.3
mse = 0.015
samples = 11
value = 0.062
X[10] <= 0.5
mse = 0.005
samples = 7
value = 0.377
X[4] <= 58.686
mse = 0.017
samples = 144
value = 0.062
X[4] <= 56.826
mse = 0.01
samples = 104
value = -0.009
mse = 0.002
samples = 9
value = 0.015
mse = 0.016
samples = 2
value = 0.284
mse = 0.001
samples = 5
value = 0.322
mse = 0.003
samples = 2
value = 0.433
mse = 0.014
samples = 89
value = 0.123
mse = 0.01
samples = 55
value = -0.027
mse = 0.008
samples = 96
value = 0.003
mse = 0.005
samples = 8
value = -0.183


X[0] <= 0.023
mse = 0.015
samples = 261
value = 0.046
True
False
X[3] <= 0.804
mse = 0.015
samples = 120
value = 0.083
X[6] <= 0.074
mse = 0.014
samples = 141
value = 0.015
X[4] <= 58.686
mse = 0.018
samples = 63
value = 0.128
X[5] <= 2.208
mse = 0.007
samples = 57
value = 0.034
X[3] <= 0.674
mse = 0.016
samples = 40
value = -0.039
X[3] <= 0.513
mse = 0.012
samples = 101
value = 0.034
mse = 0.016
samples = 51
value = 0.161
mse = 0.003
samples = 12
value = -0.0
mse = 0.005
samples = 56
value = 0.025
mse = -0.0
samples = 1
value = 0.235
mse = 0.013
samples = 23
value = 0.009
mse = 0.013
samples = 17
value = -0.1
mse = 0.009
samples = 17
value = -0.044
mse = 0.011
samples = 84
value = 0.053


X[4] <= 59.711
mse = 0.019
samples = 263
value = 0.035
True
False
X[3] <= 0.697
mse = 0.019
samples = 226
value = 0.049
X[0] <= 0.164
mse = 0.01
samples = 37
value = -0.058
X[6] <= 0.456
mse = 0.014
samples = 102
value = 0.09
X[4] <= 56.699
mse = 0.02
samples = 124
value = 0.019
X[3] <= 0.501
mse = 0.006
samples = 25
value = -0.104
X[4] <= 60.074
mse = 0.003
samples = 12
value = 0.053
mse = 0.015
samples = 98
value = 0.135
mse = 0.008
samples = 53
value = 0.035
mse = 0.016
samples = 115
value = 0.042
mse = 0.001
samples = 9
value = -0.207
mse = 0.001
samples = 6
value = -0.205
mse = 0.003
samples = 19
value = -0.066
mse = 0.0
samples = 3
value = -0.042
mse = 0.001
samples = 9
value = 0.074

Implementing ensembling

```
# make predictions from 2 neural net models
test_pred1 = model_1.predict(scaled_test_features)
test_pred2 = model_2.predict(scaled_test_features)
# horizontally stack predictions and take the average across rows
test_preds = np.mean(np.hstack((test_pred1, test_pred2)), axis=1)
```

Comparing the ensemble

Model 1 R^2 score on test set:

- -0.179

model 2:

- -0.148

ensemble (averaged predictions):

- -0.146

Dropout and ensemble!

MACHINE LEARNING FOR FINANCE IN PYTHON

datacamp