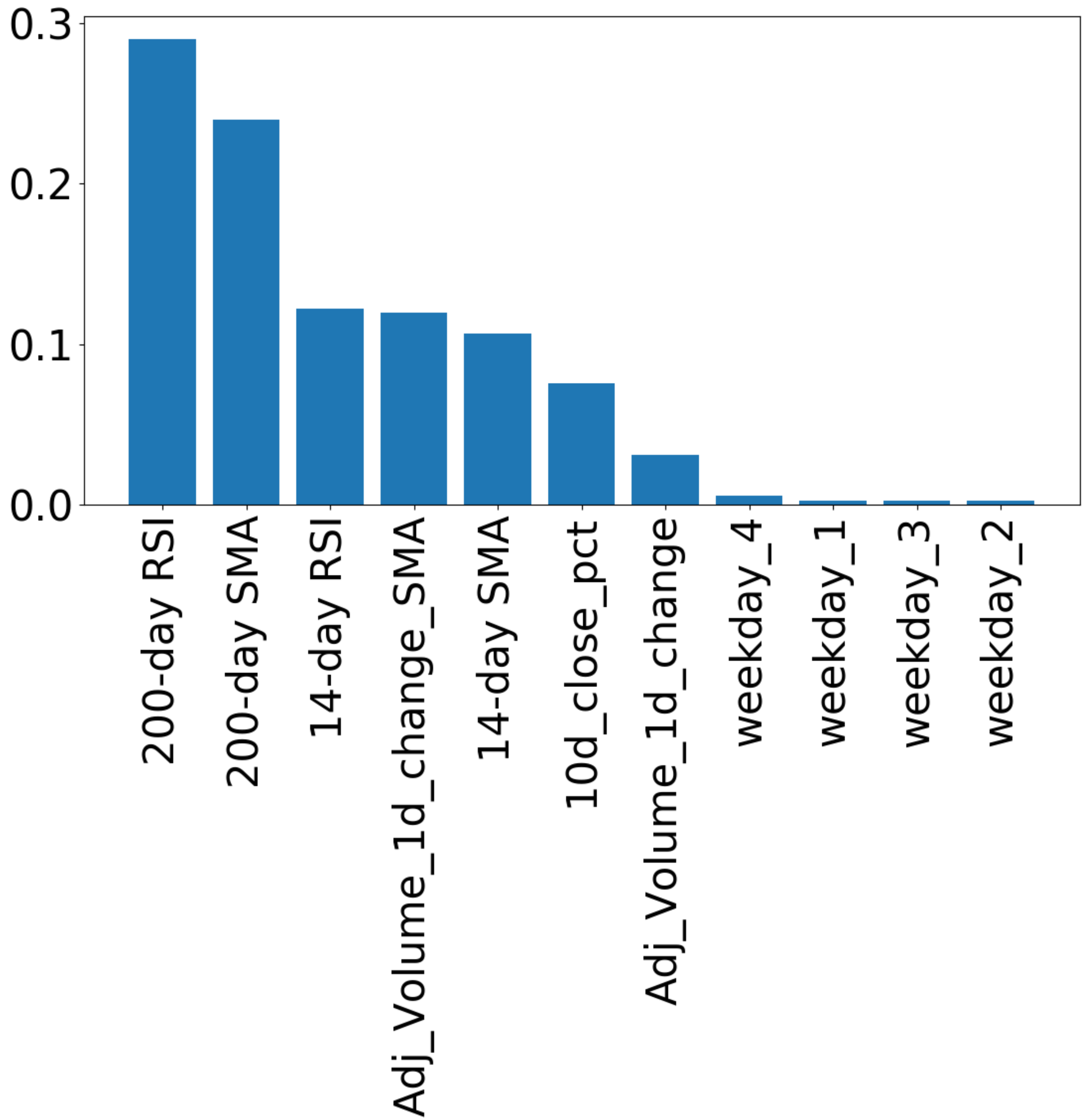
MACHINE LEARNING FOR FINANCE IN PYTHON

Scaling data and KNN Regression

Nathan George
Data Science Professor

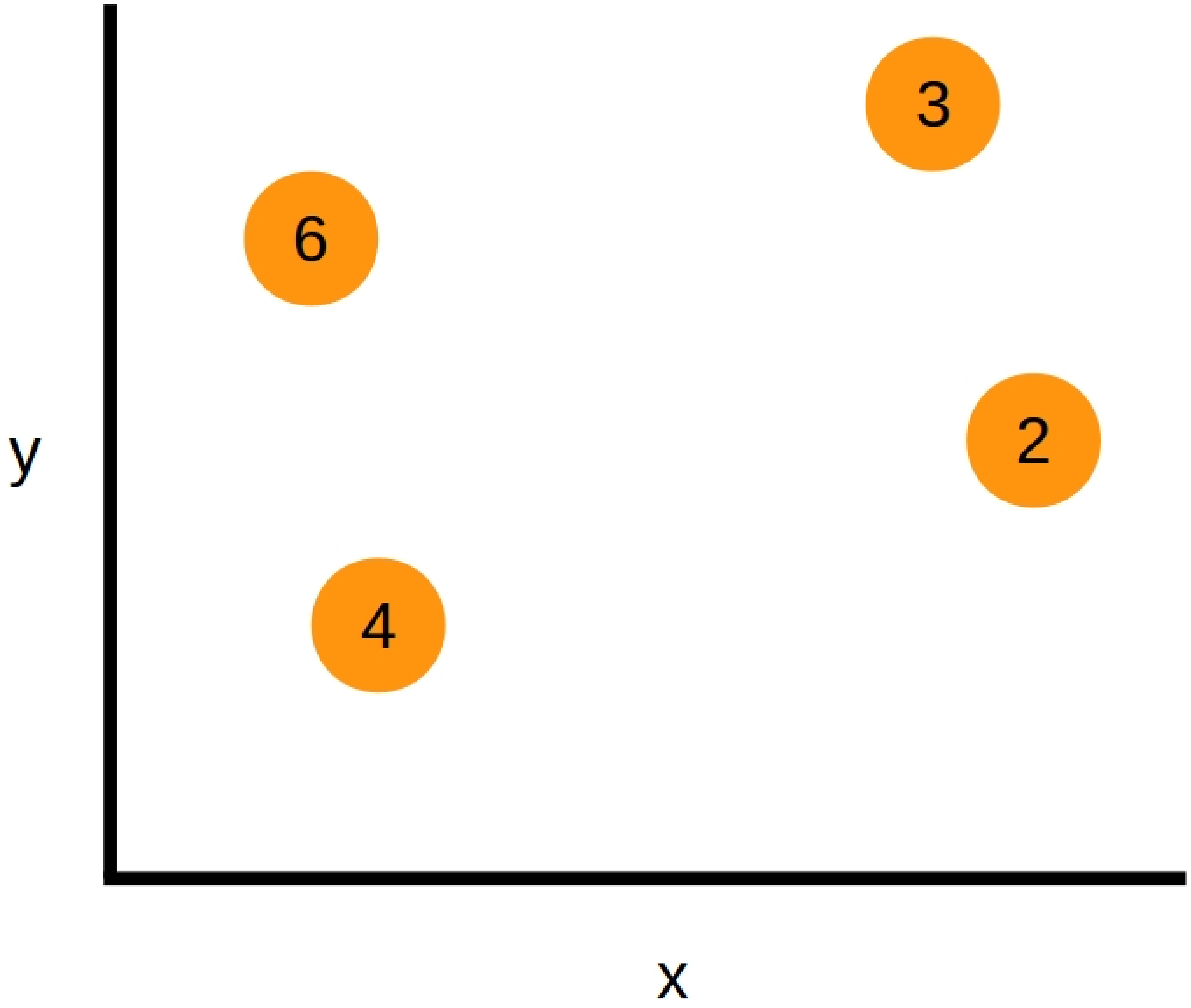

0.3
0.2
0.1
0.0
200-day RSI
200-day SMA
14-day RSI
Adj_Volume_1d_change_SMA
14-day SMA
10d_close_pct
Adj_Volume_1d_change
weekday_4
weekday_1
weekday_3
weekday_2

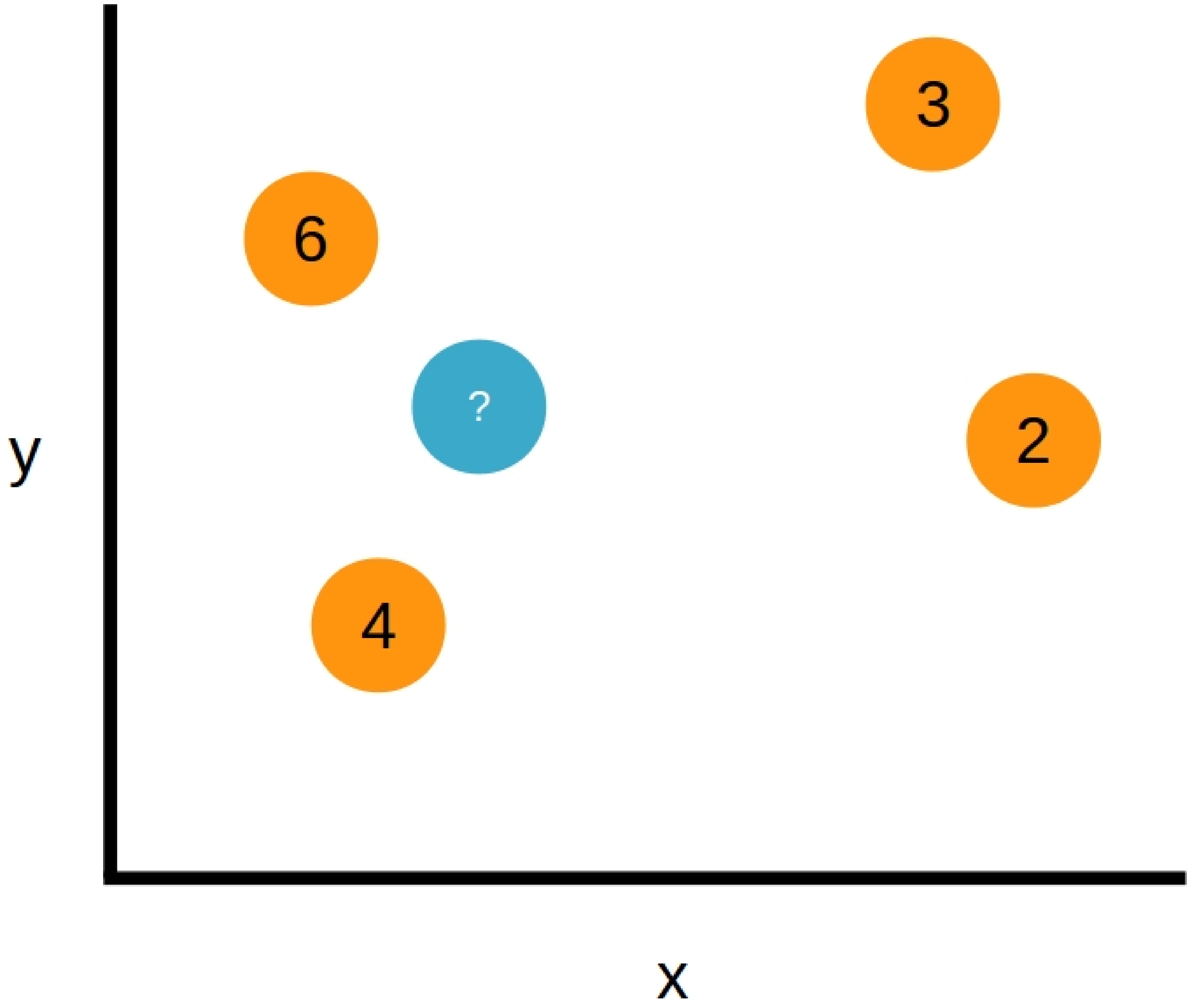
Feature selection: remove weekdays

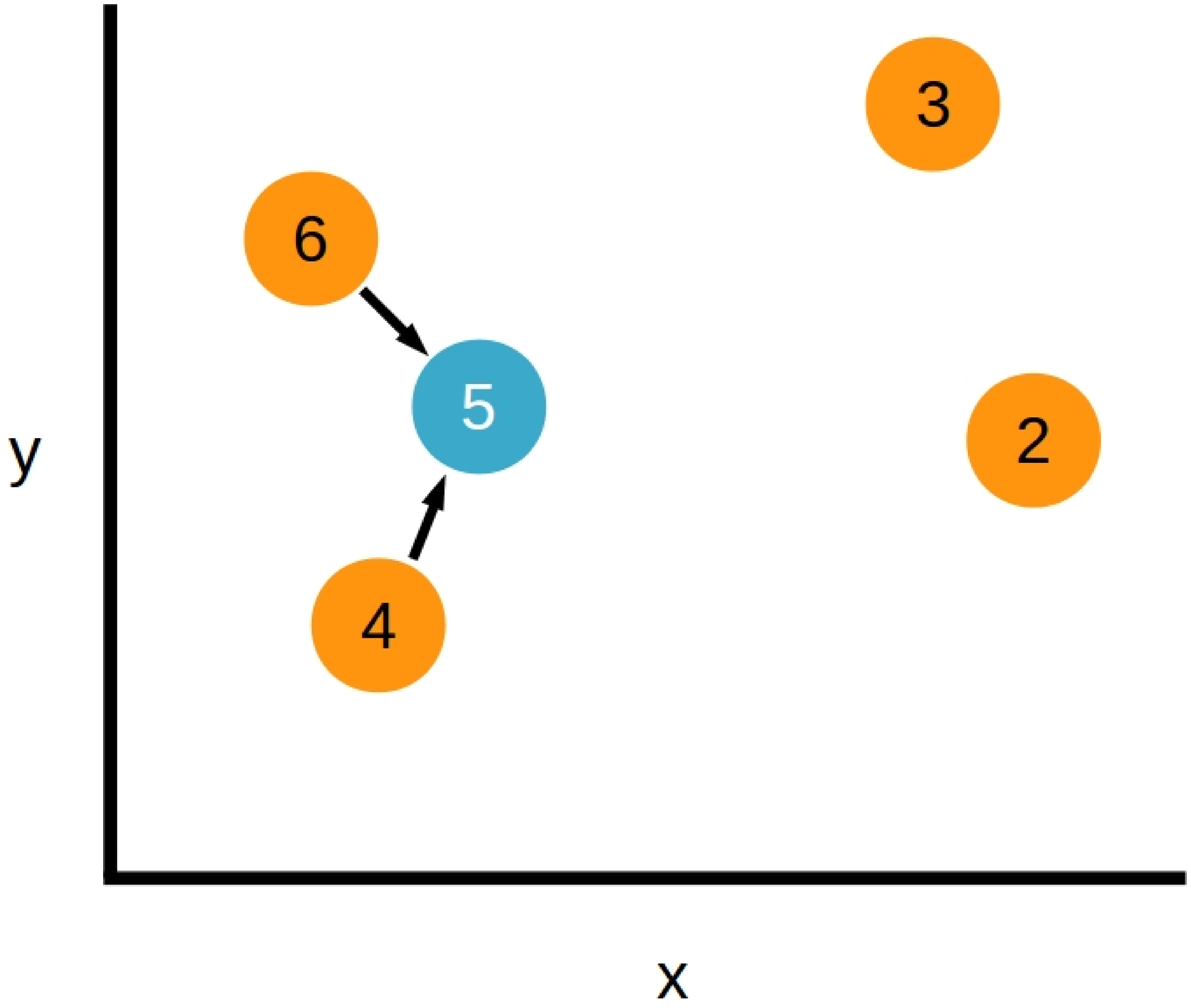
```
print(feature_names)

['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

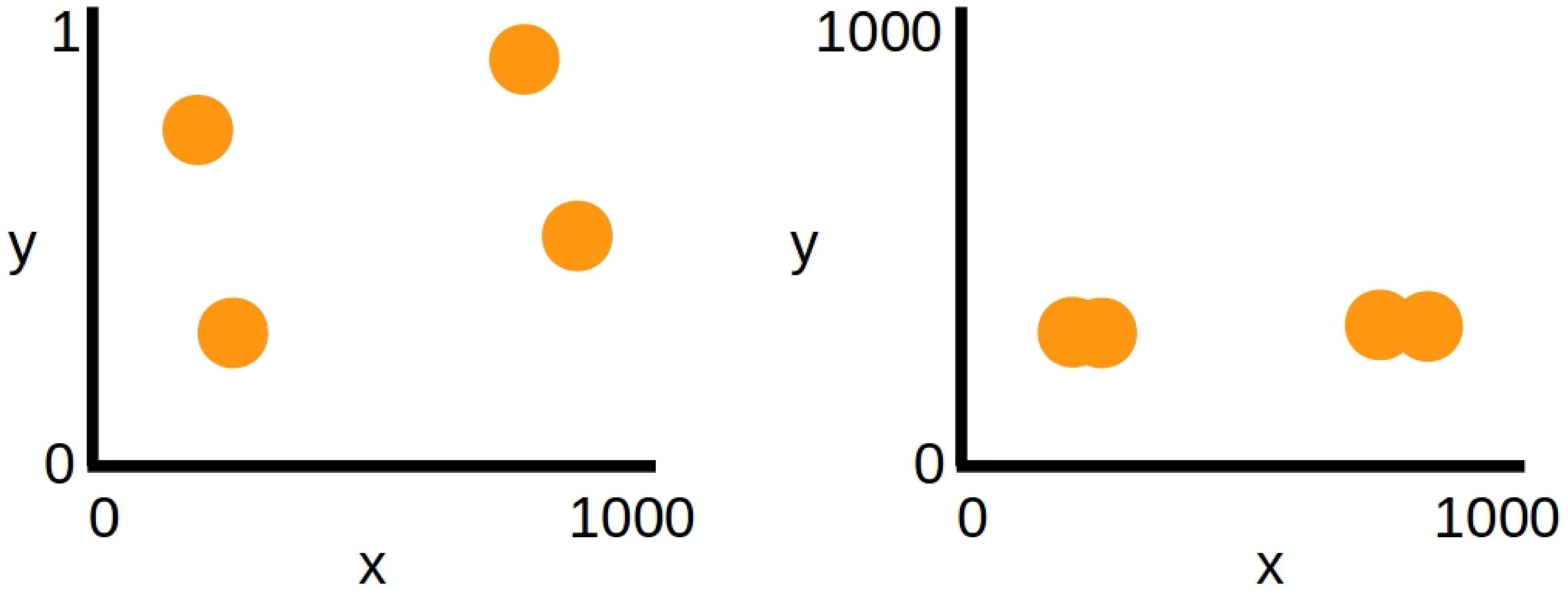
```
train_features = train_features.iloc[:, :-4]
test_features = test_features.iloc[:, :-4]
```


3
6
y
2
4
x


3
6
?
y
2
4
x


3
6
5
2
y
4
x

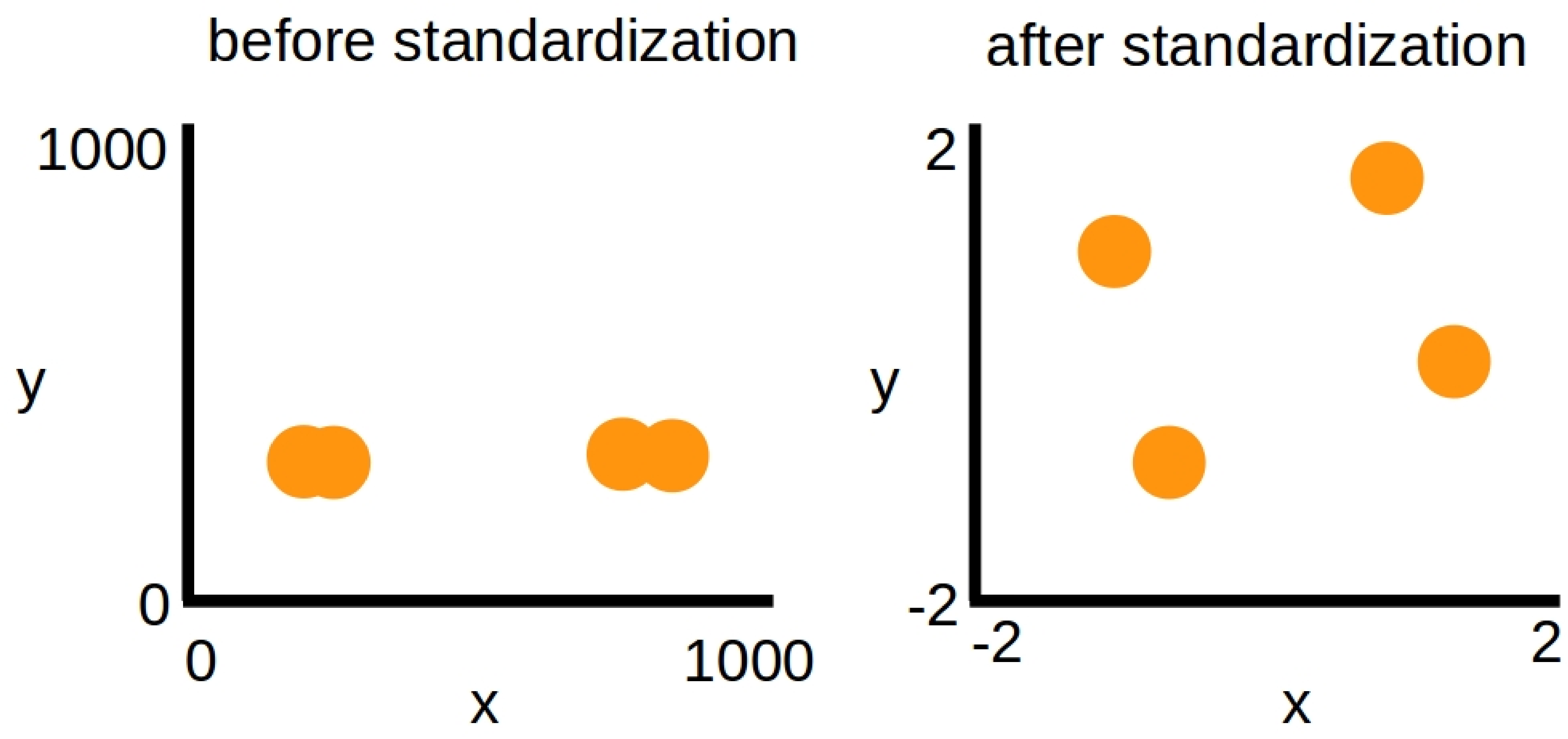
$$D(A,B)=\sum_{i}\left(\left|(a_i-b_i)\right|\right)^{(1/p)}$$


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

Scaling options

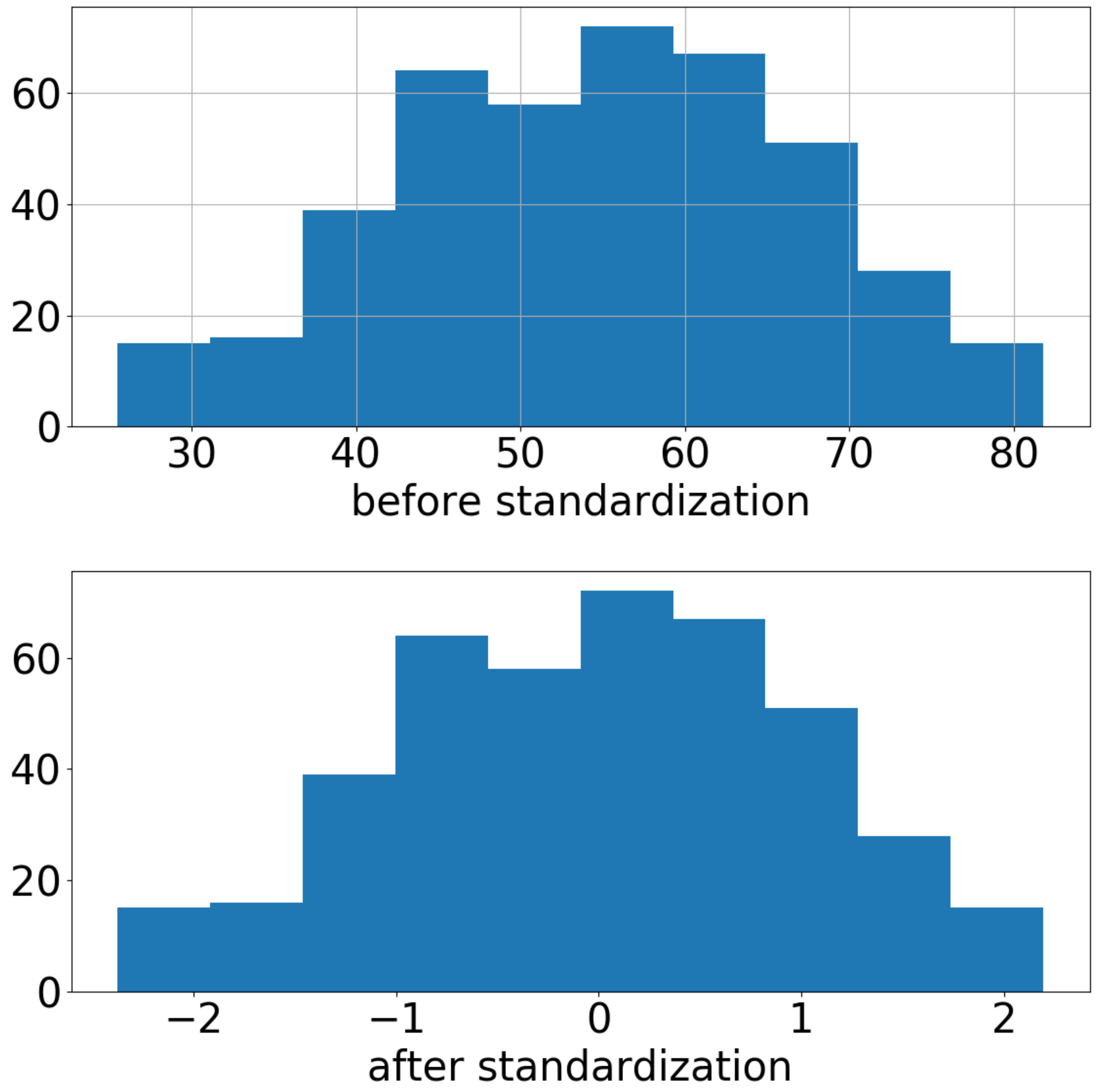
Scaling options:

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


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

sklearn's scaler

```
from sklearn.preprocessing import scaler

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


0
20
40
60
30
40
50
60
70
80
before standardization
0
20
40
60
−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()
```

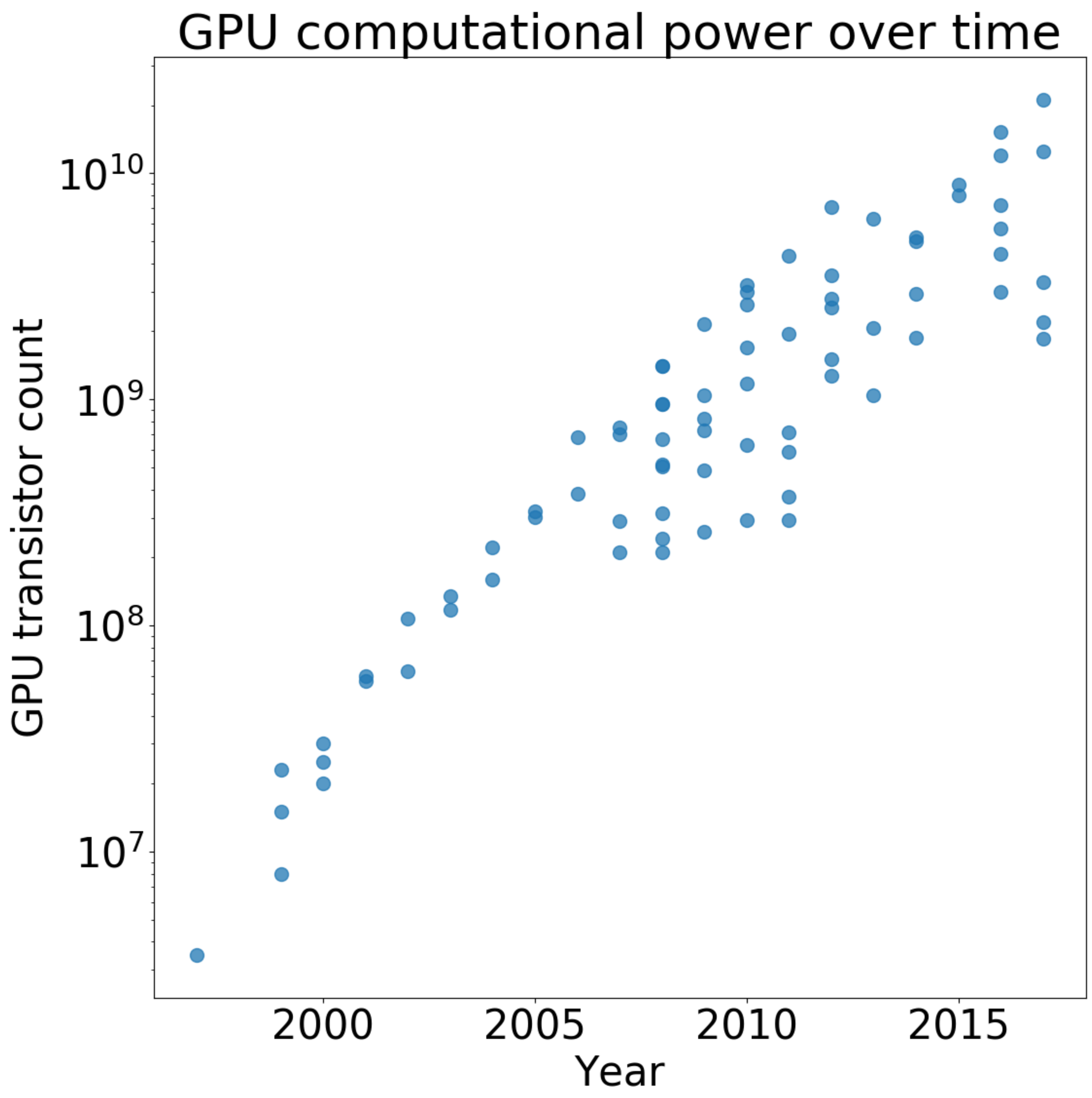
MACHINE LEARNING FOR FINANCE IN PYTHON

Scale data and use KNN!

MACHINE LEARNING FOR FINANCE IN PYTHON

Neural Networks

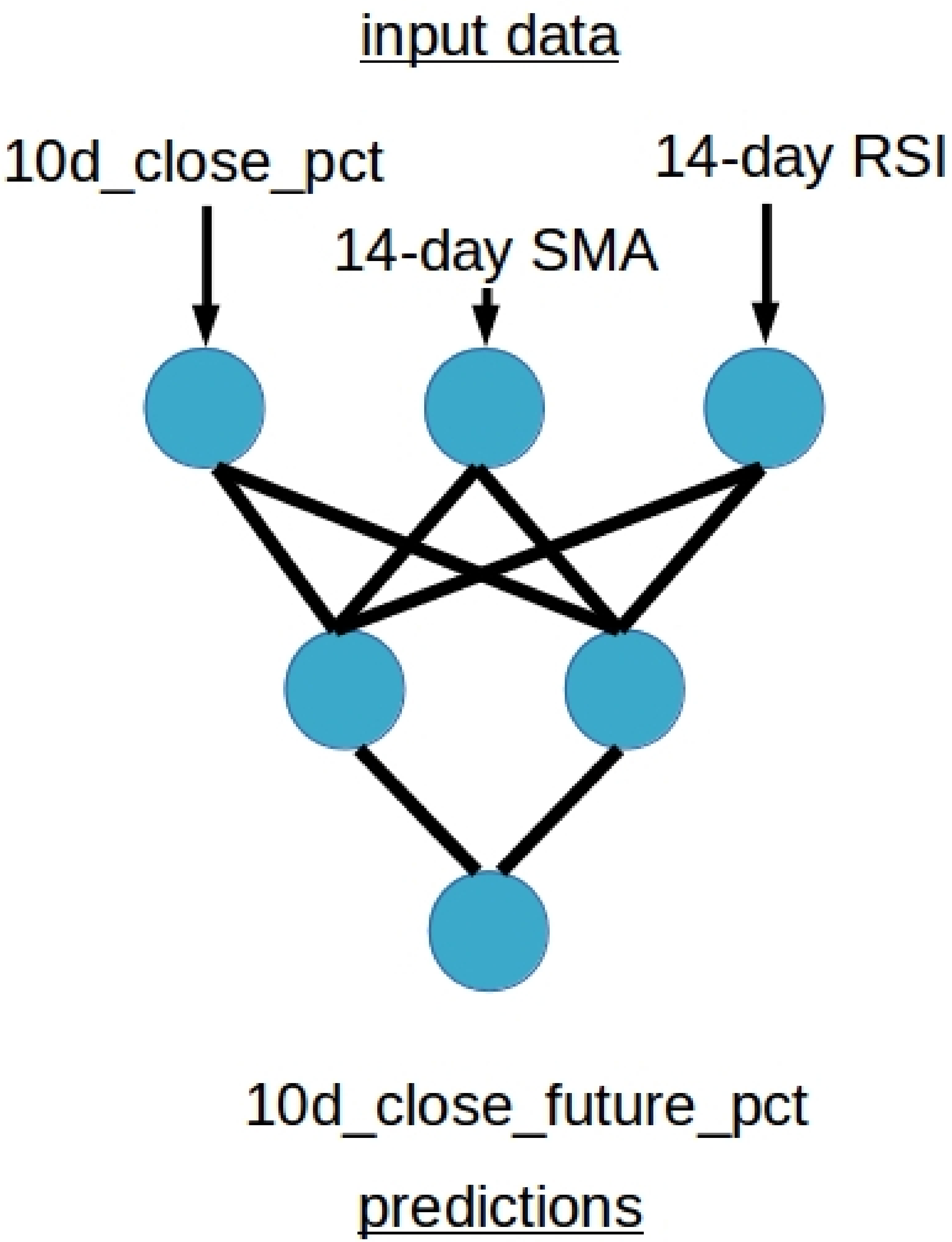
Nathan George
Data Science Professor

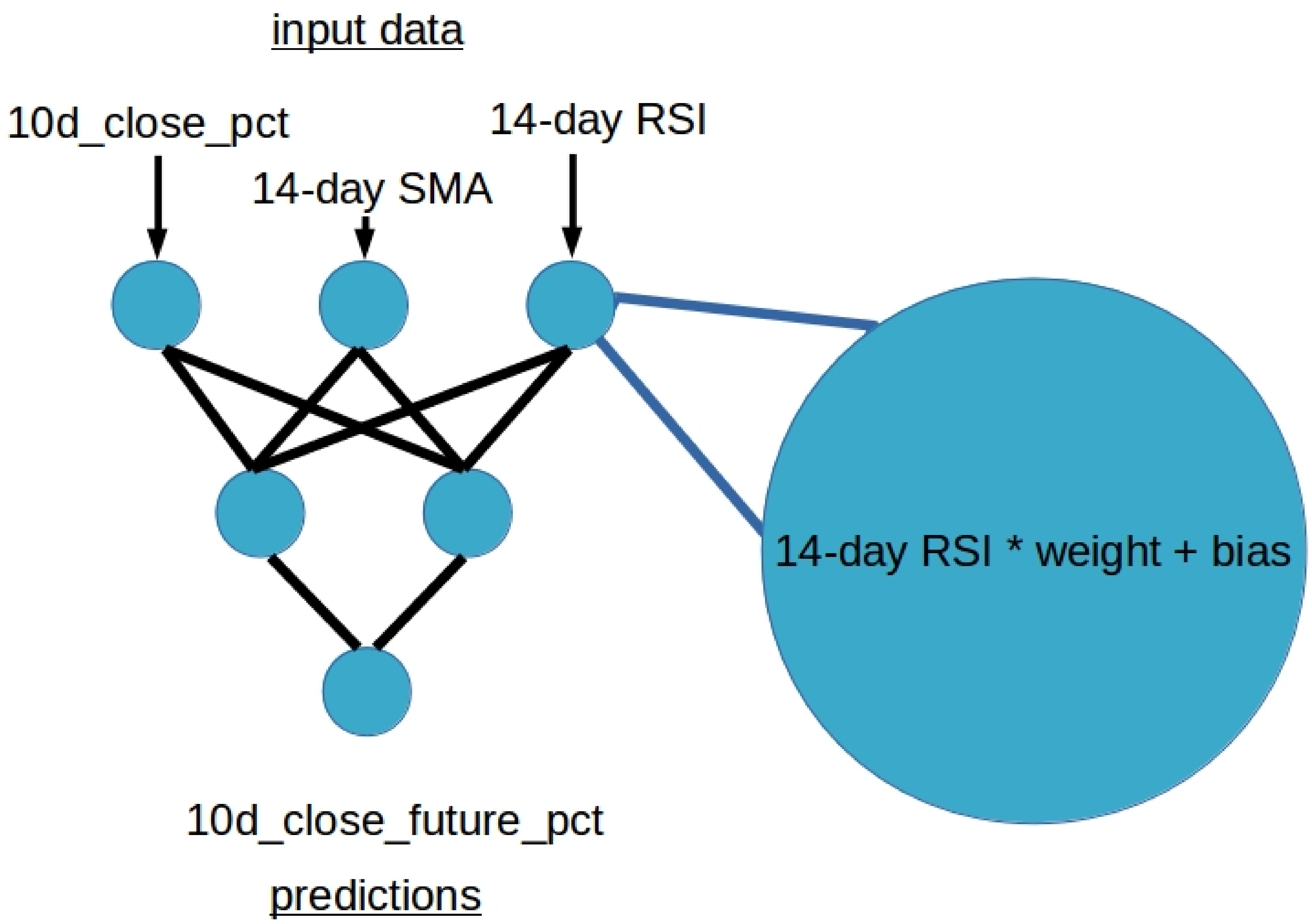

GPU computational power over time
GPU transistor count
10^{10}
10^{9}
10^{8}
10^{7}
2000
2005
2010
2015
Year

Neural networks have potential

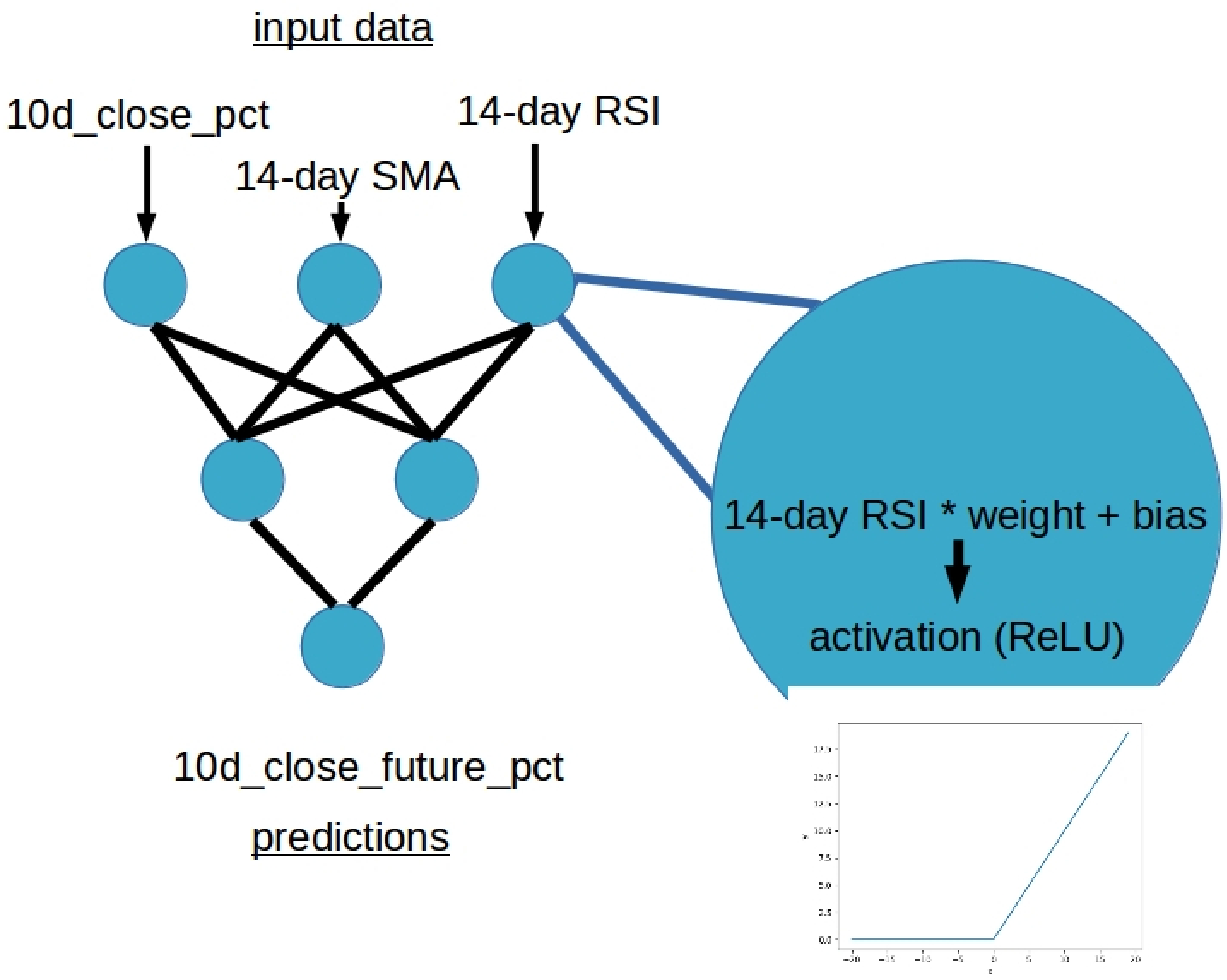
Neural nets have:

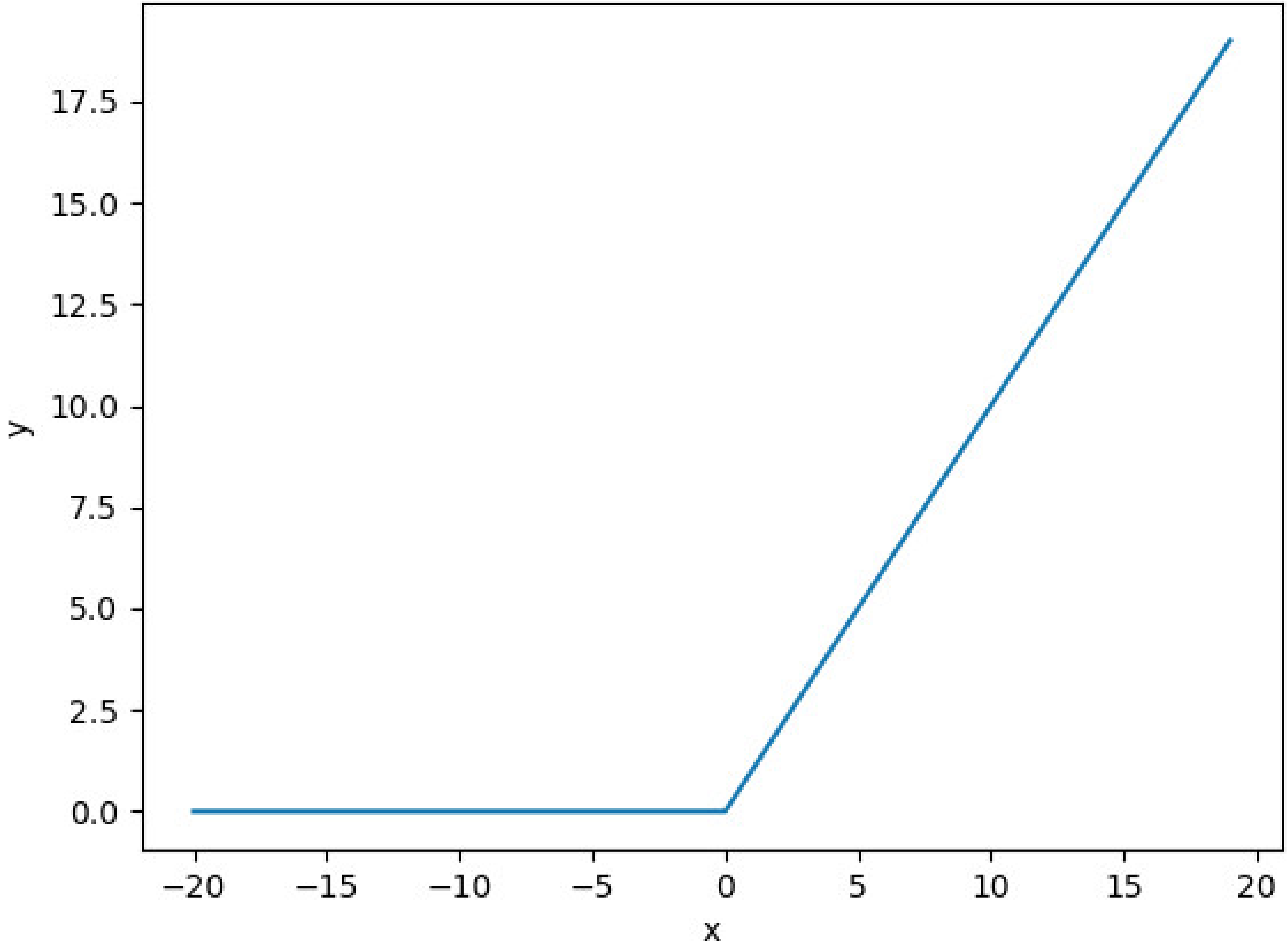
- non-linearity
- variable interactions
- customizability

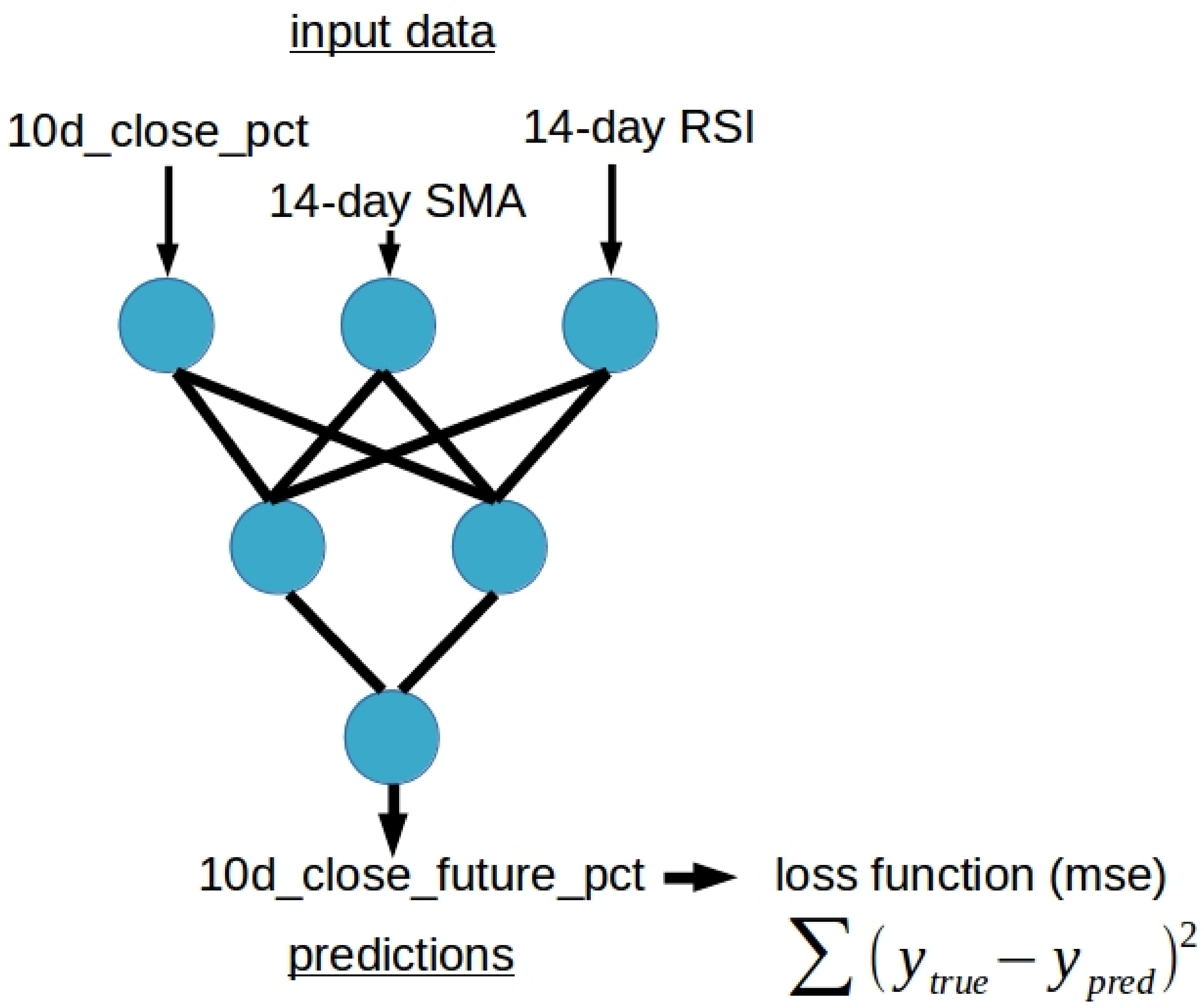

input data
10d_close_pct
14-day SMA
14-day RSI
10d_close_future_pct
predictions

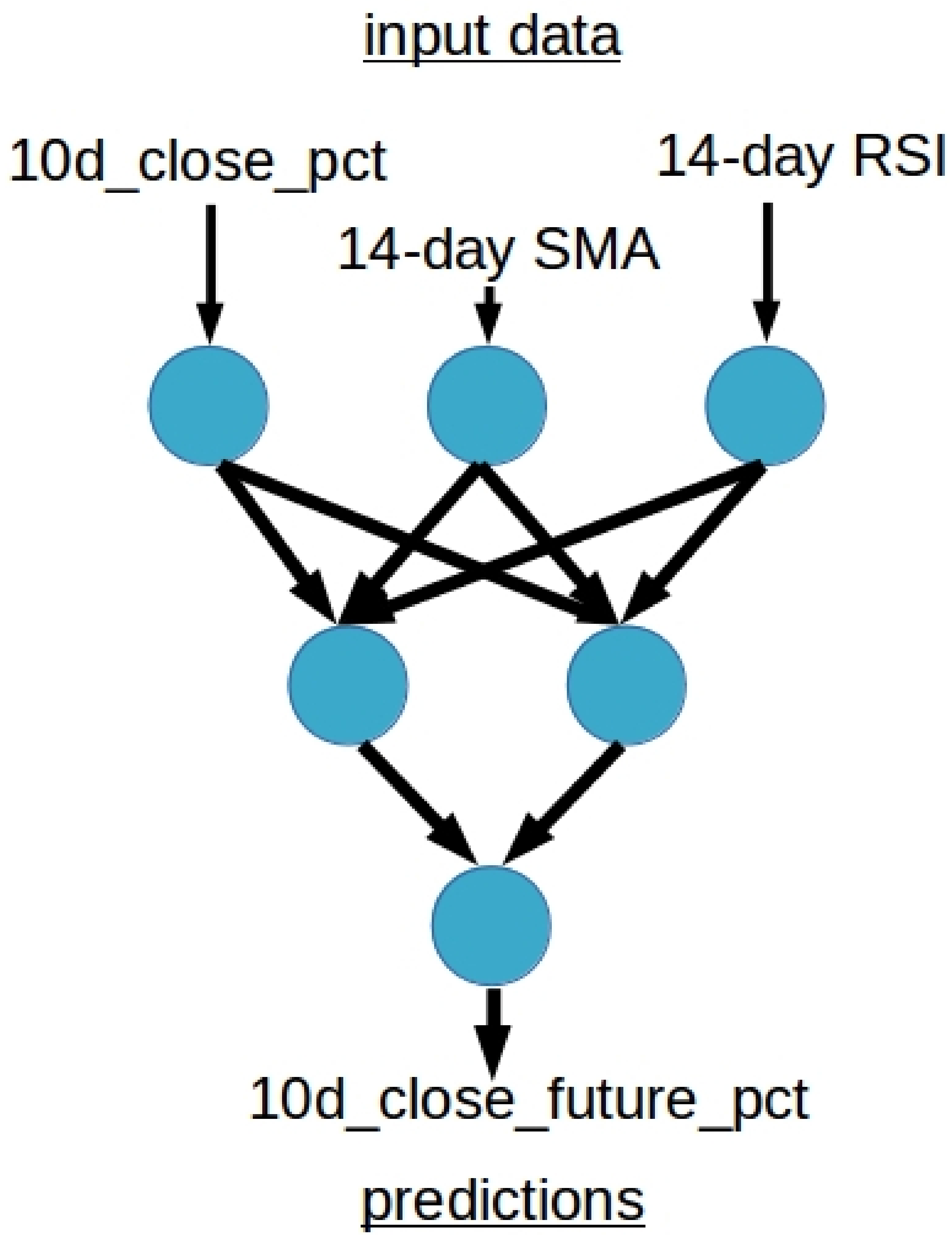

input data
10d_close_pct
14-day SMA
14-day RSI
14-day RSI * weight + bias
10d_close_future_pct
predictions

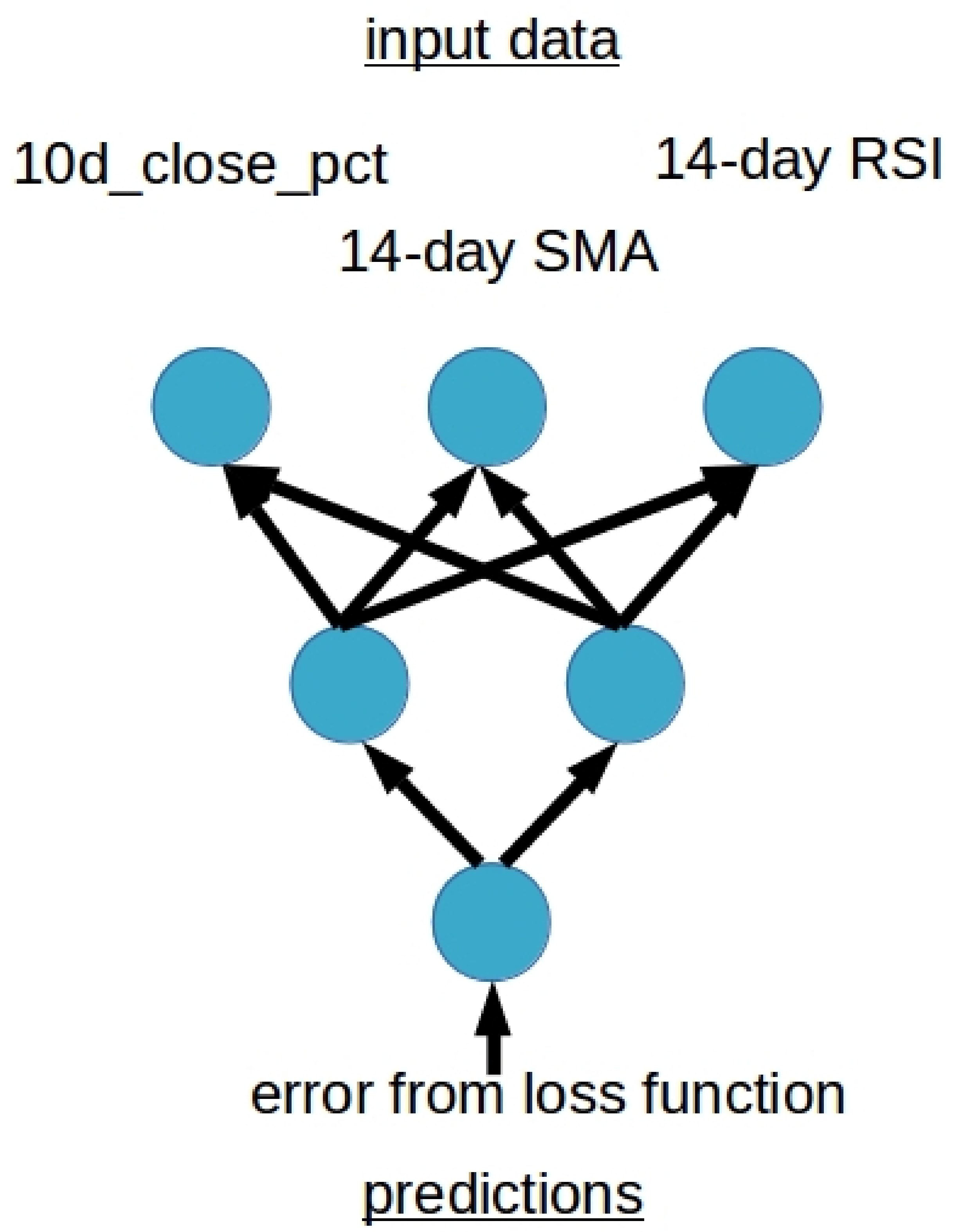
$$\sum_i w_i x_i + b$$


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
predictions
loss function (mse)
$\sum (y_{true} - y_{pred})^2$


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


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

Keras

TensorFlow

Implementing a neural net with keras

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

Implementing a neural net with keras

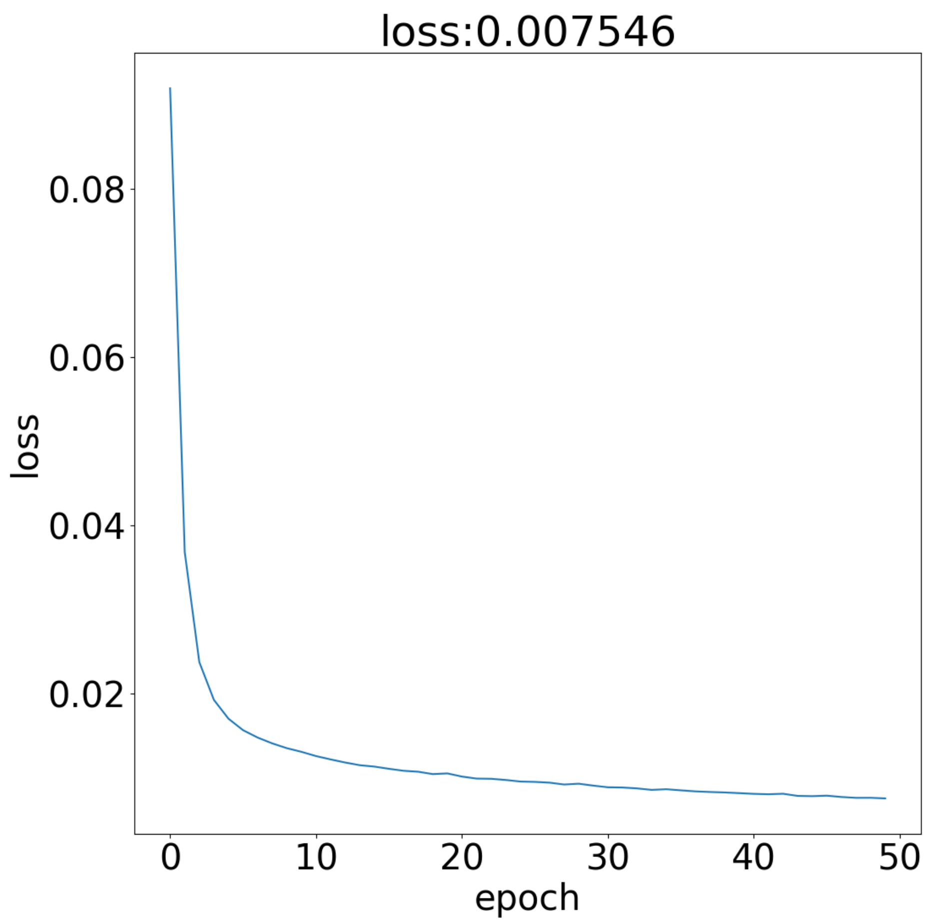
```
from keras.models import Sequential
from keras.layers import Dense

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)
```

Examining the loss

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



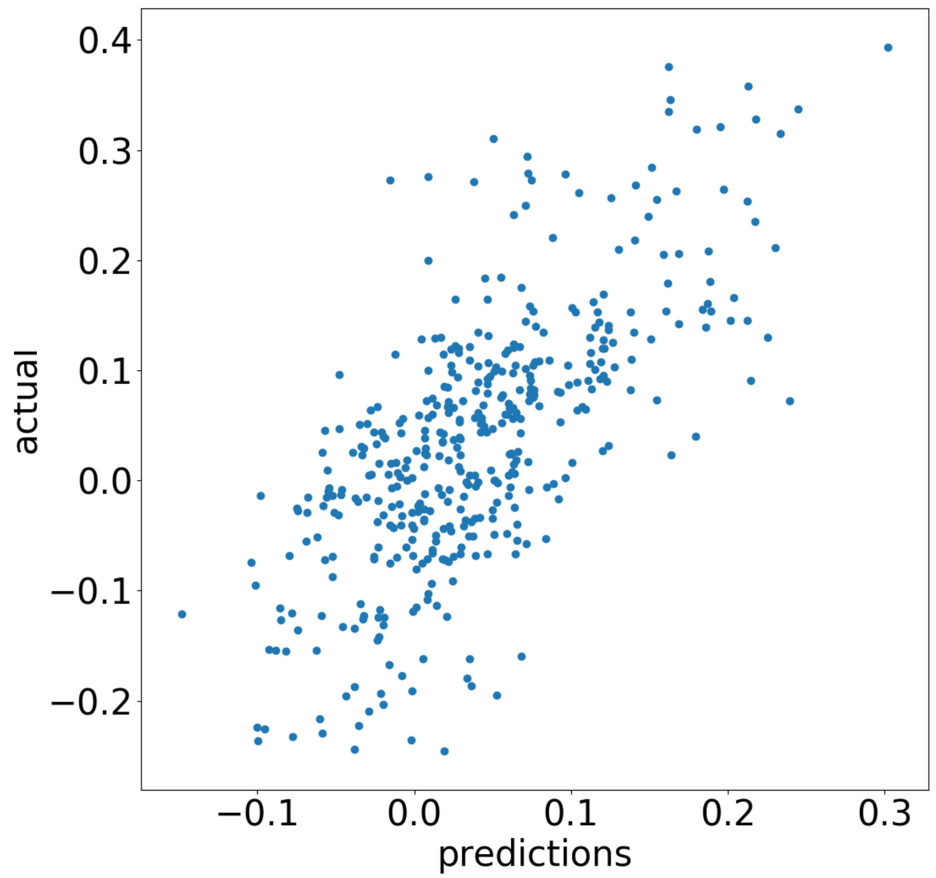
Checking out performance

```
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()
```



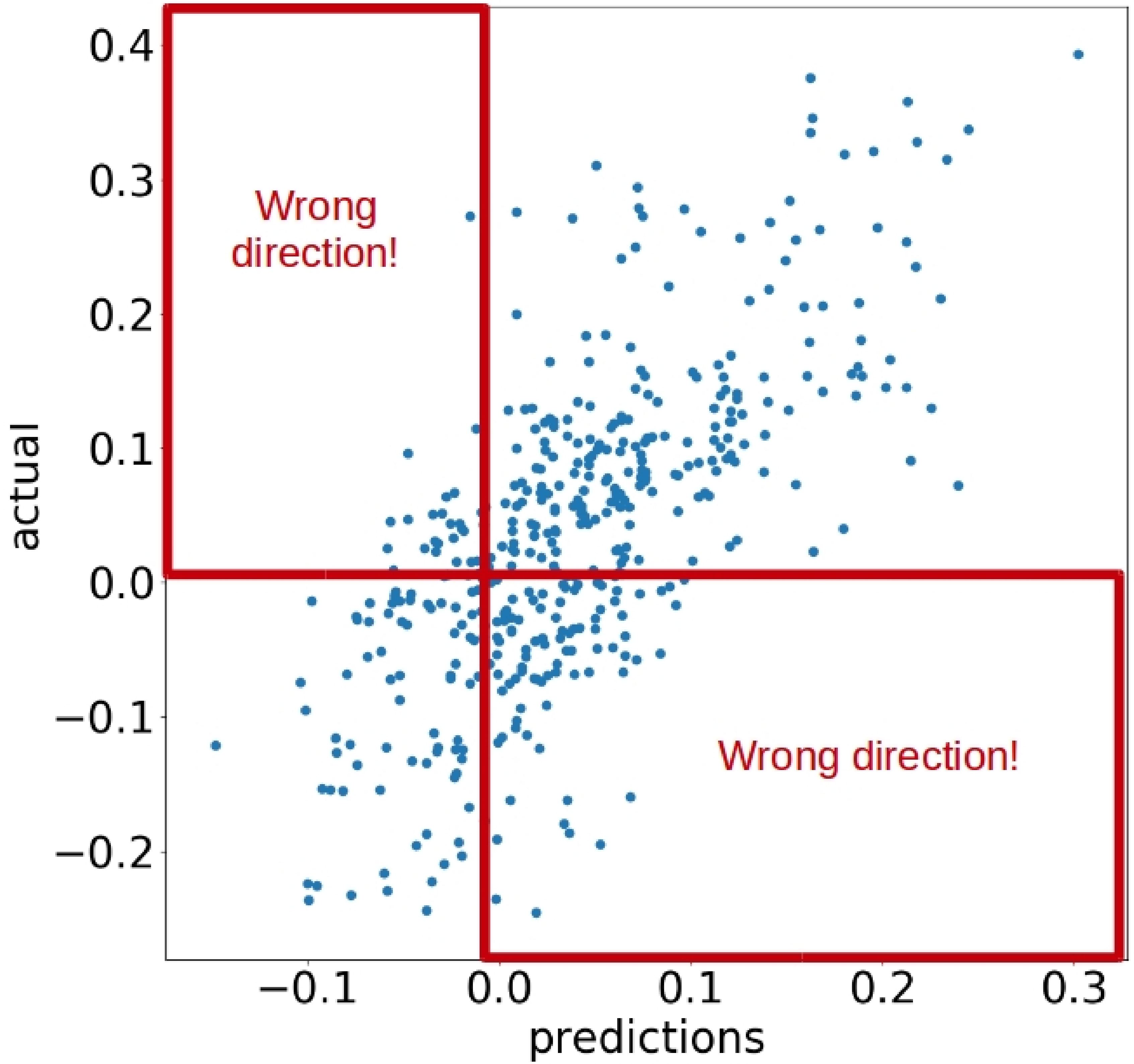
MACHINE LEARNING FOR FINANCE IN PYTHON

Make a neural net!

MACHINE LEARNING FOR FINANCE IN PYTHON

Custom loss functions

Nathan George
Data Science Professor


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

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)

keras.losses.sign_penalty = sign_penalty  # enable use of loss with keras
```

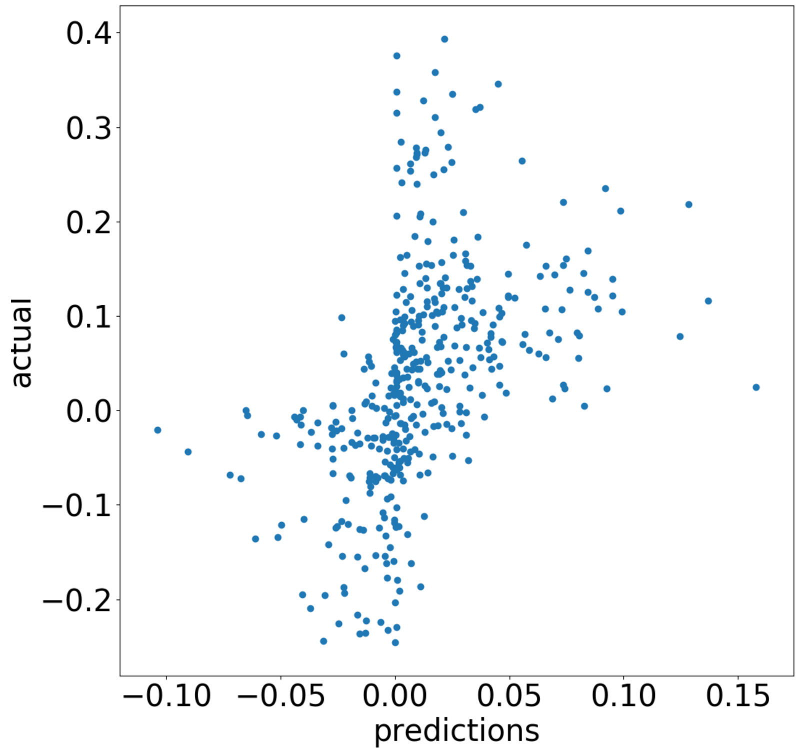
Using the custom loss

```
# 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()
```



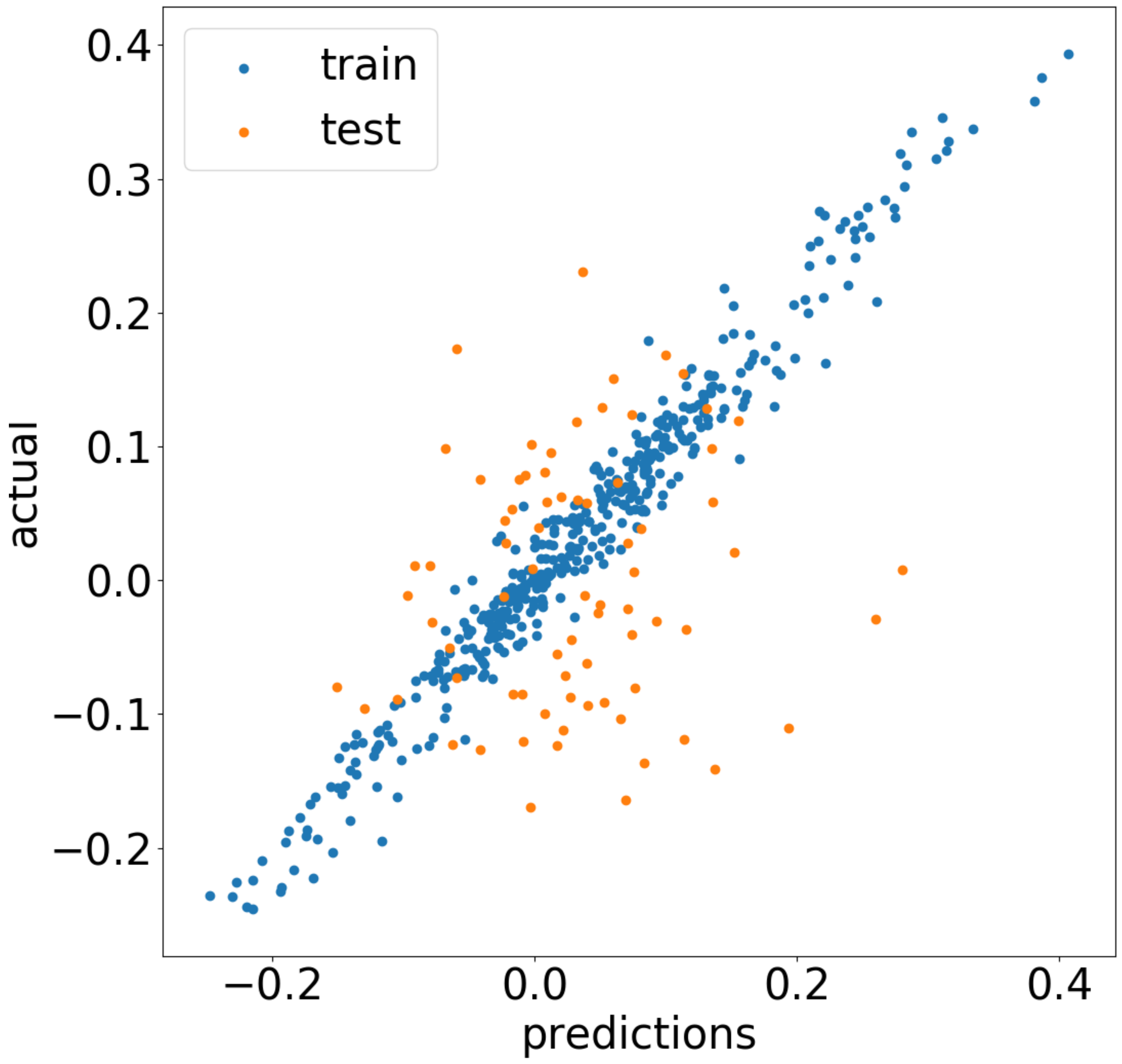
MACHINE LEARNING FOR FINANCE IN PYTHON

Create your own loss function!

MACHINE LEARNING FOR FINANCE IN PYTHON

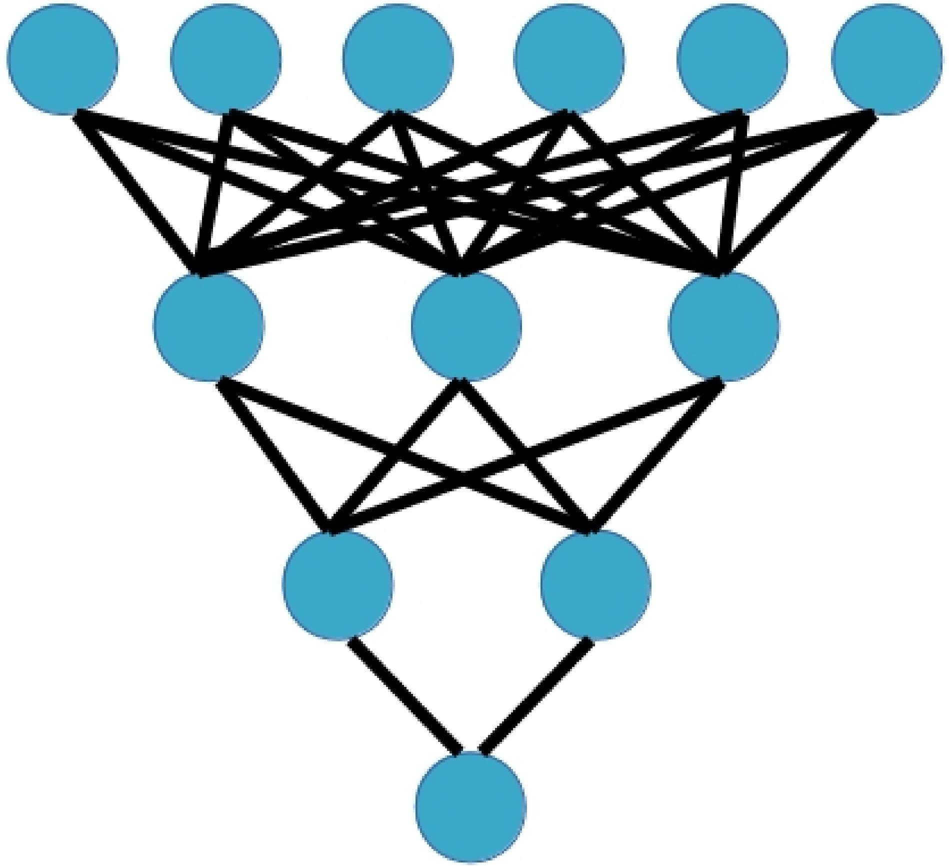
Overfitting and ensembling

Nathan George
Data Science Professor

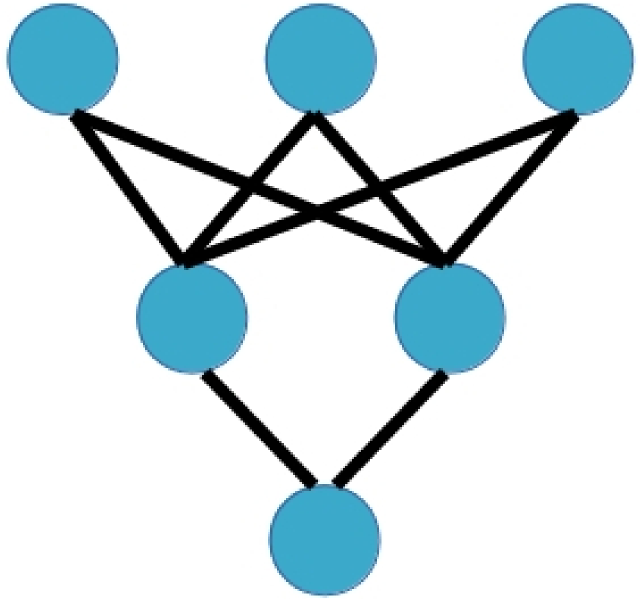

train
test
actual
predictions
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.2
0.0
0.2
0.4

Simplify your model

Complex net overfits



Simpler net prevents overfitting

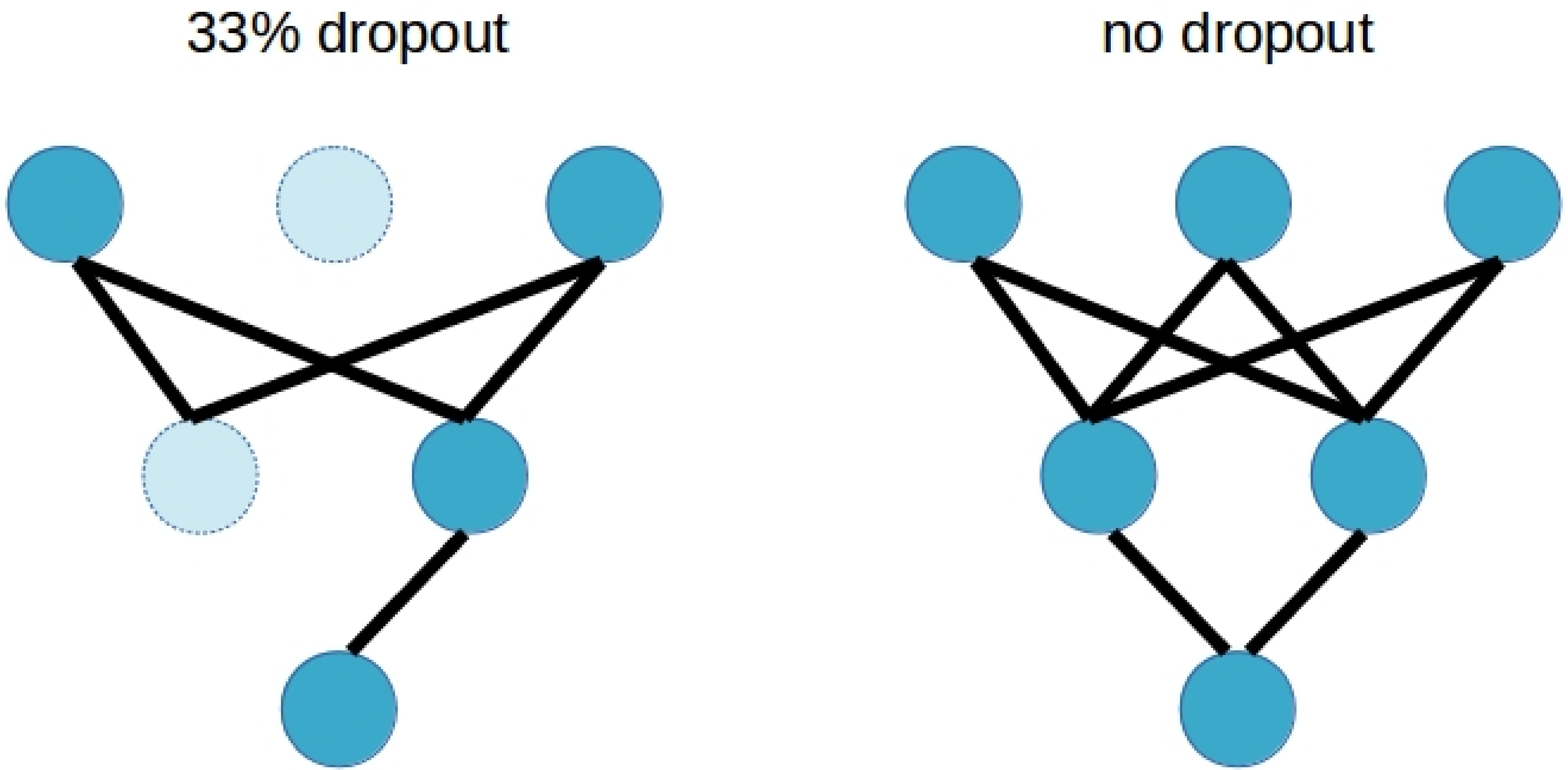


Neural network options

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



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

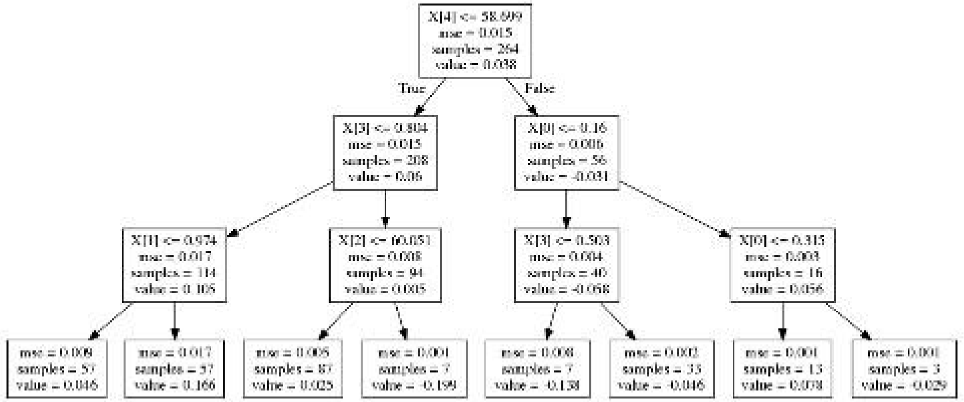
R^2 values on AMD without dropout:

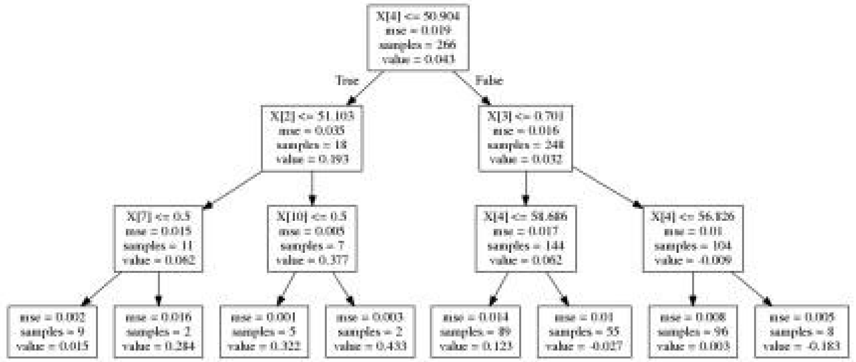
- train: 0.91
- test: -0.72

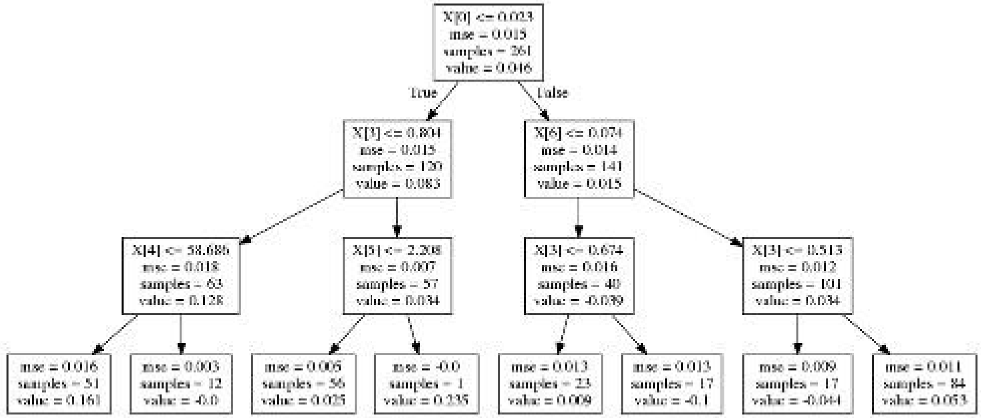
With dropout:

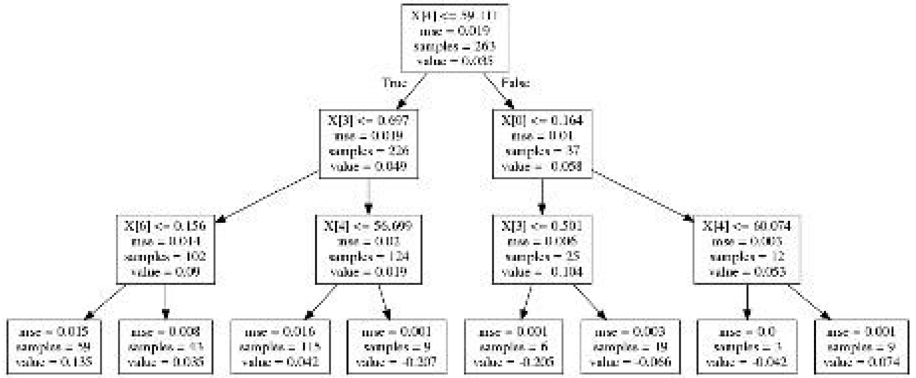
- train: 0.46
- test: -0.22

Ensembling









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

MACHINE LEARNING FOR FINANCE IN PYTHON

Dropout and ensemble!