Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Best Practices in Machine Learning

CAS Artificial Intelligence - Machine Learning

Tobias Mérinat
Prof. Dr. Marc Pouly

tobias.merinat@hslu.ch

FH Zentralschweiz

Motivation

1. You carefully split your data into training, validation and test set
2. You train your favorite learning algorithm on the training set
3. You perform hyperparameter optimization on the validation set
4. You get 20% error on the test set which seems rather high

What can you do ?

Some Ideas …

- You could try getting more or better training data
- You could try a smaller set of features
- You could try a larger set of features
- You could try entirely different features
- You could try more iterations of gradient descent or adapt the learning rate
- You could try other optimization algorithms than gradient descent
- You could try different hyper-parameters (e.g. regularization parameters)
- You could try other machine learning algorithms

Just trying random modifications is not the way to go!
You need to know what is going on first.

Bias and Variance

Different sources of error can prevent supervised learning algorithms from generalizing beyond their training set

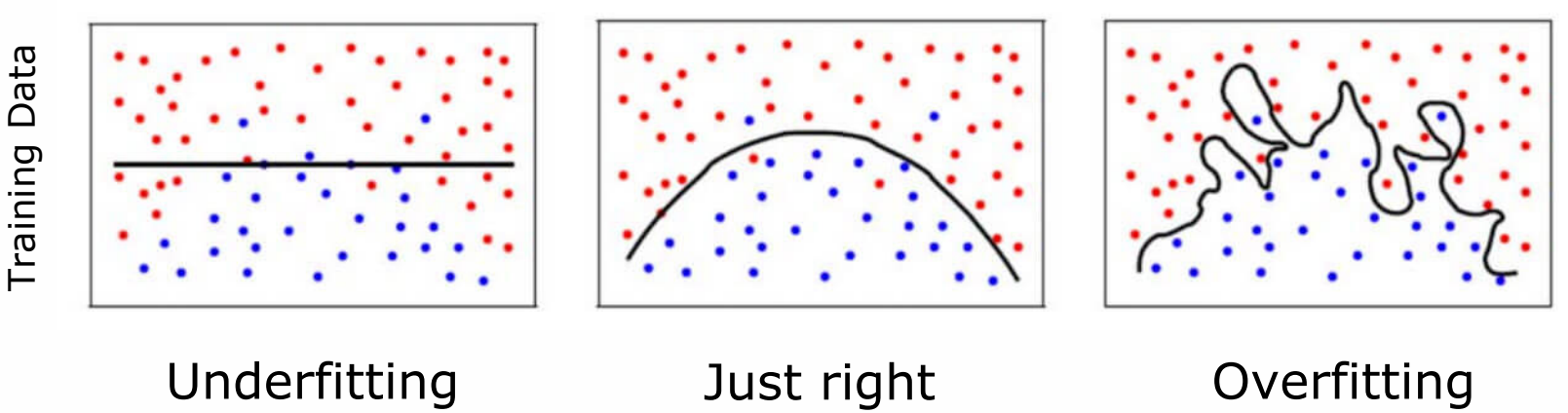
- The bias results from false assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant structure between features and target

- The variance results from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data rather than the intended outputs

- A model with high bias and low variance underfits the data

- A model with low bias and high variance overfits the data

Underfitting vs Overfitting for Classification



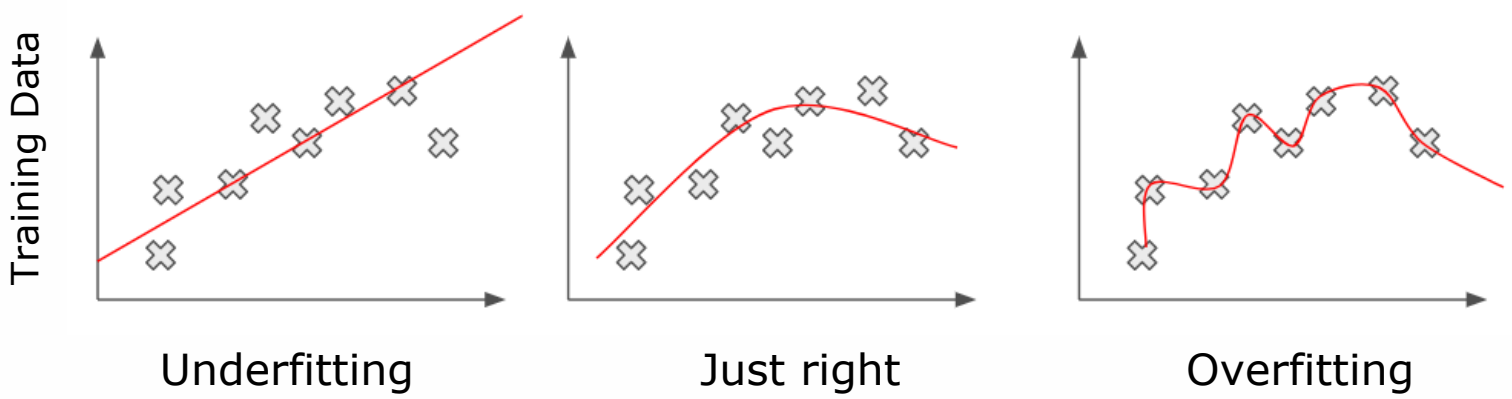
Underfitting:

- A straight line is just not complex enough to separate the classes
- This model performs badly on both training and test data

Overfitting:

- This model optimizes to much with respect to the training data
- It shows high performance on training but low performance on test data

Underfitting vs Overfitting for Regression



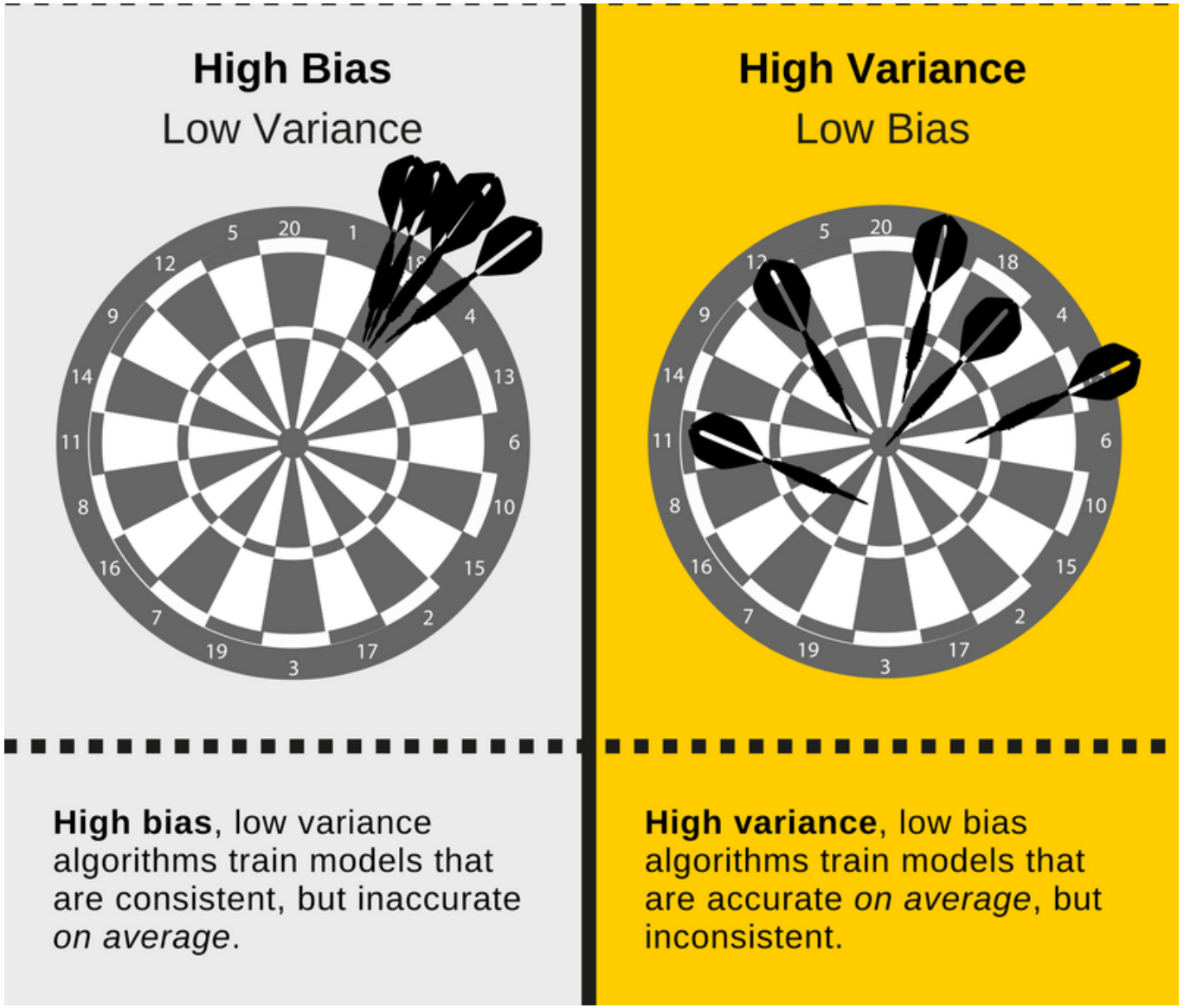
Underfitting:

- A straight line is just not complex enough to approximate the data
- This model performs badly on both training and test data

Overfitting:

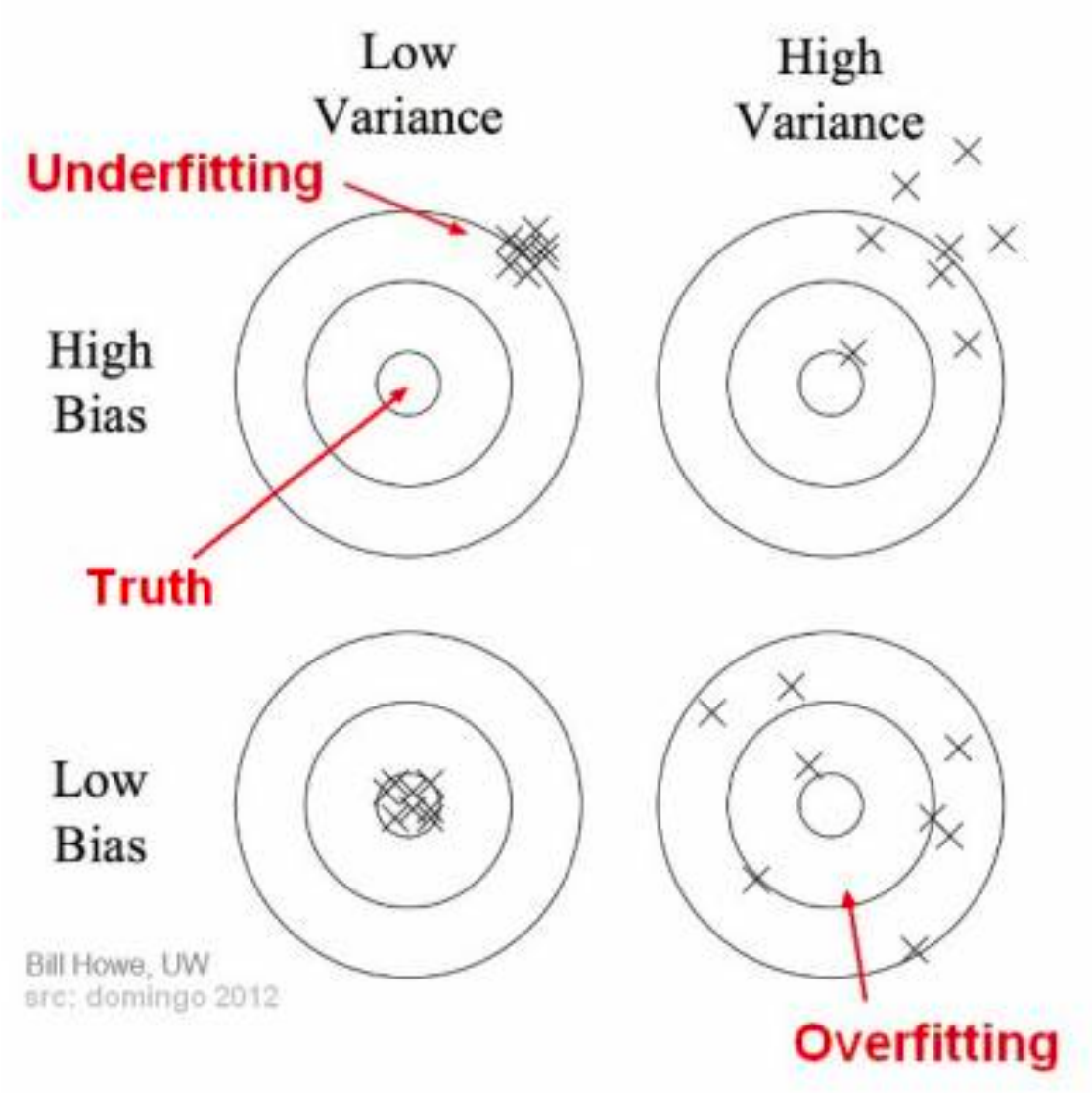
- This model optimizes to much with respect to the training data
- It shows high performance on training but low performance on test data

Underfitting



Overfitting

Image Source: https://elitedatascience.com

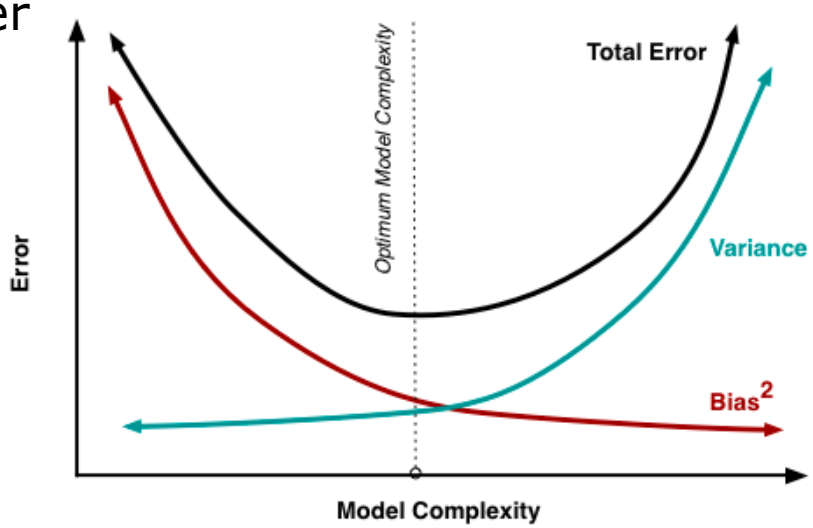

Low
Variance
High
Variance
Underfitting
High
Bias
Truth
Low
Bias
Bill Howe, UW
src: domingo 2012
Overfitting

Bias-Variance Tradeoff

- Bias and Variance go hand-in-hand. It is often easy to improve one but simultaneously worsen the other

- This is called the
  Bias-Variance Tradeoff

- But out goal is to reach an overall improvement!



Note that the Bias-Variance Tradeoff is less severe with large deep neural networks. There, simply adding more and more and more data often improves variance while not affecting bias negatively.

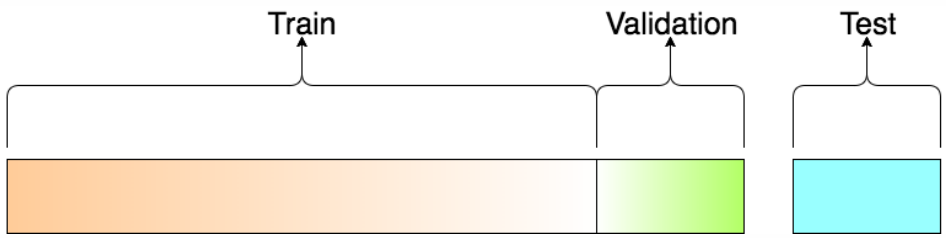
Image source: dataquest.io/

Our List of Ideas revisited…

- You could try getting more or better training data
- You could try a smaller set of features
- You could try a larger set of features
- You could try more iterations of gradient descent or adapt the learning rate
- You could try other optimization algorithms than gradient descent
- You could try different hyper-parameters (e.g. regularization parameters)
- You could try other machine learning algorithms
-

Which action would be good when underfitting?
When overfitting?

Recap: More Complex Evaluation Workflows



- A recommended split is 60% / 20% / 20%
- How to evaluate a machine learning classifier properly:
    1. Loop over all hyperparameter combinations you would like to try
    2. Train model on selected hyperparameter setting and training set
    3. Evaluate model on validation set and measure performance
    4. Select model with best performance as candidate
    5. Evaluate model on test set for final performance estimate
- This workflow requires a lot of data

Recap: Cross Validation

A train-validation-test split of 60% / 20% / 20% is common*, but this only works when there is enough data. If not, split you data into 80% training and 20% test data (lock your test data away) and perform k-fold cross-validation. Even in the presence of enough data, cross-validations may bring in additional trustworthiness

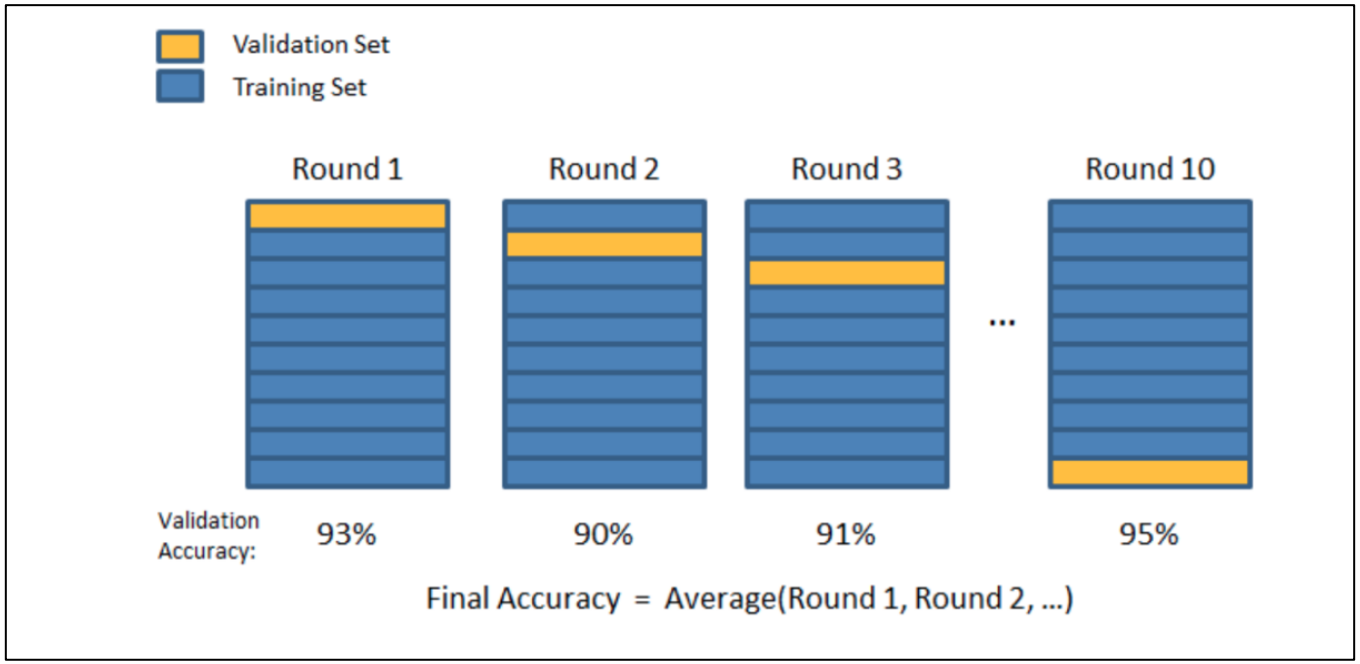
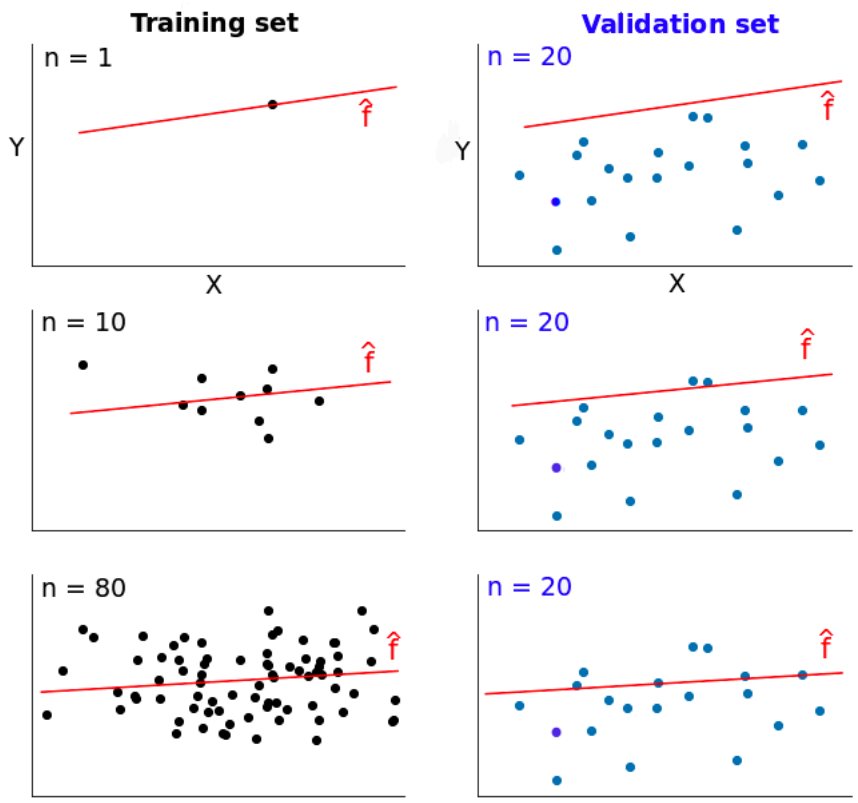


Illustration of 10-fold cross-validation

*If you have a huge amount of data, as you should when training large deep NN, it is OK to make your validation and test sets a lot smaller than 20%.

Error in respect to number of samples



- Training error increases first and gets better over time

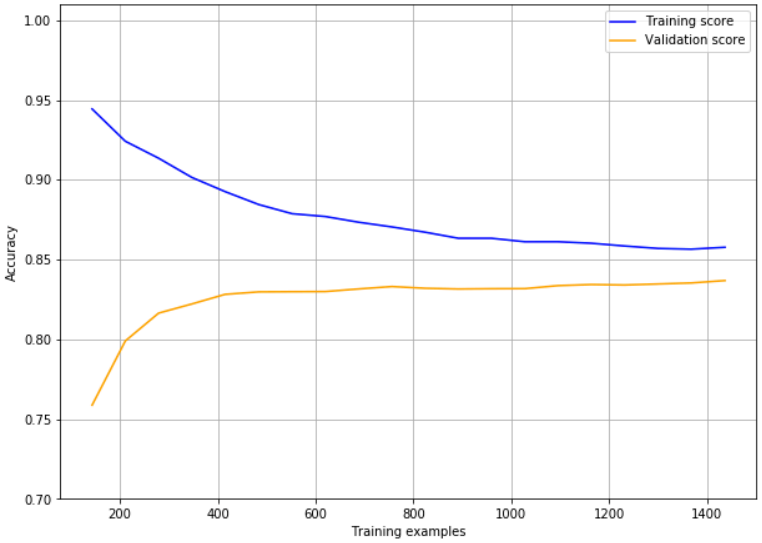
- Validation error is off in the beginning but approaches training error over time

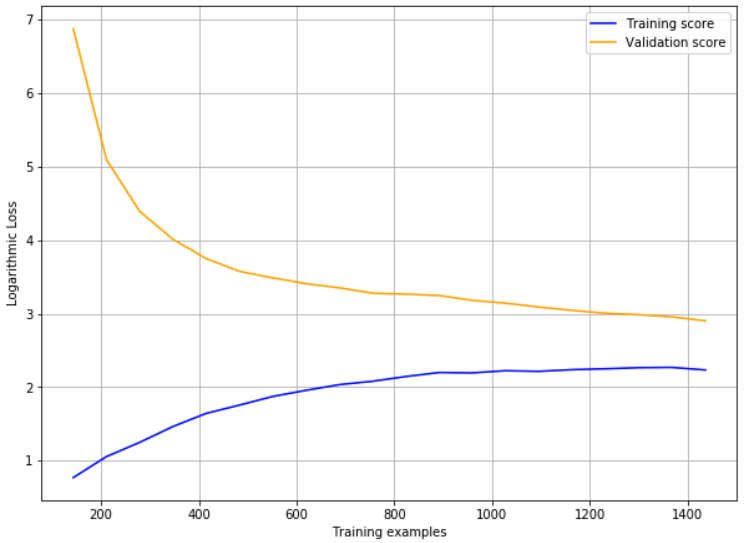
Learning Curves

- Learning curves are a diagnostic tool

- The x-axis normally shows either «experience» or sometimes «model complexity»

- They usually show the «amount of learning» (in a chosen metric) on the y-axis



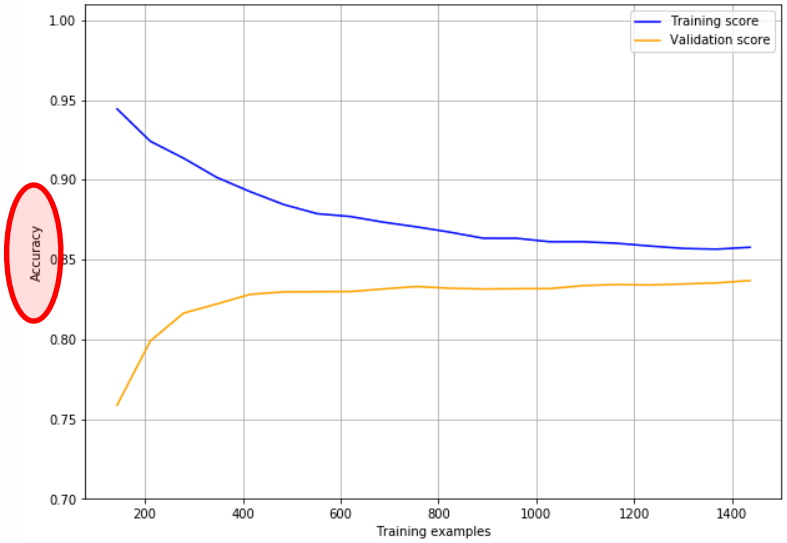


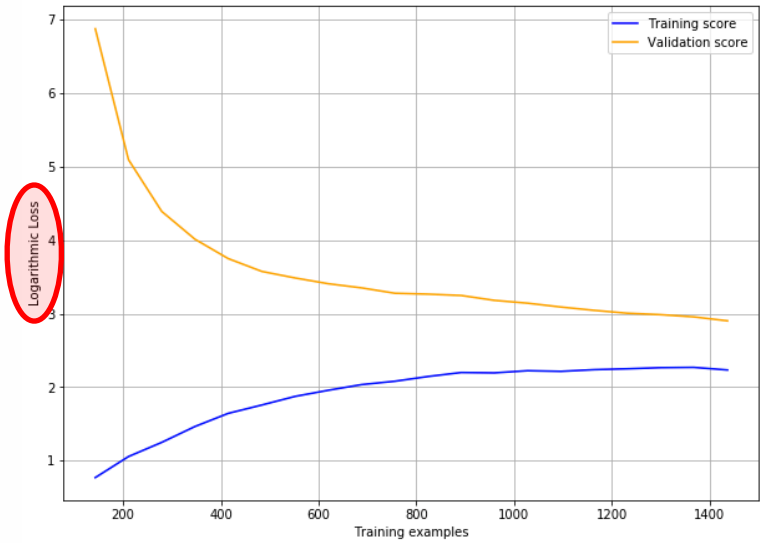
Learning Curves

- Learning curves are a diagnostic tool

- The x-axis normally shows either «experience» or sometimes «model complexity»

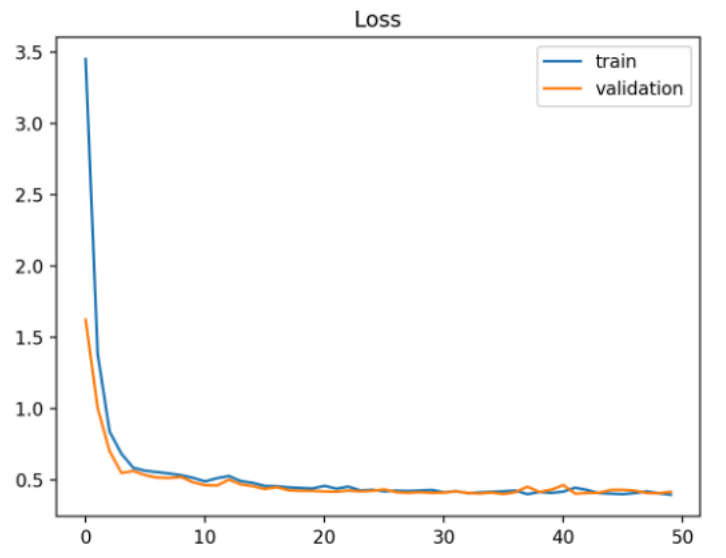
- They usually show the «amount of learning» (in a chosen metric) on the y-axis





Learning Curves in Neural Networks

- With Neural Networks, we often plot one data point per epoch (where above, we plotted one data point per batch)
- The x-axis still shows «experience», as we train longer (many epochs)
- But the effect of an increasing training error is much less pronounced, if it is there at all
- This makes the curves a bit easier to understand



A rather nice-looking set of learning curves

Irreducible error

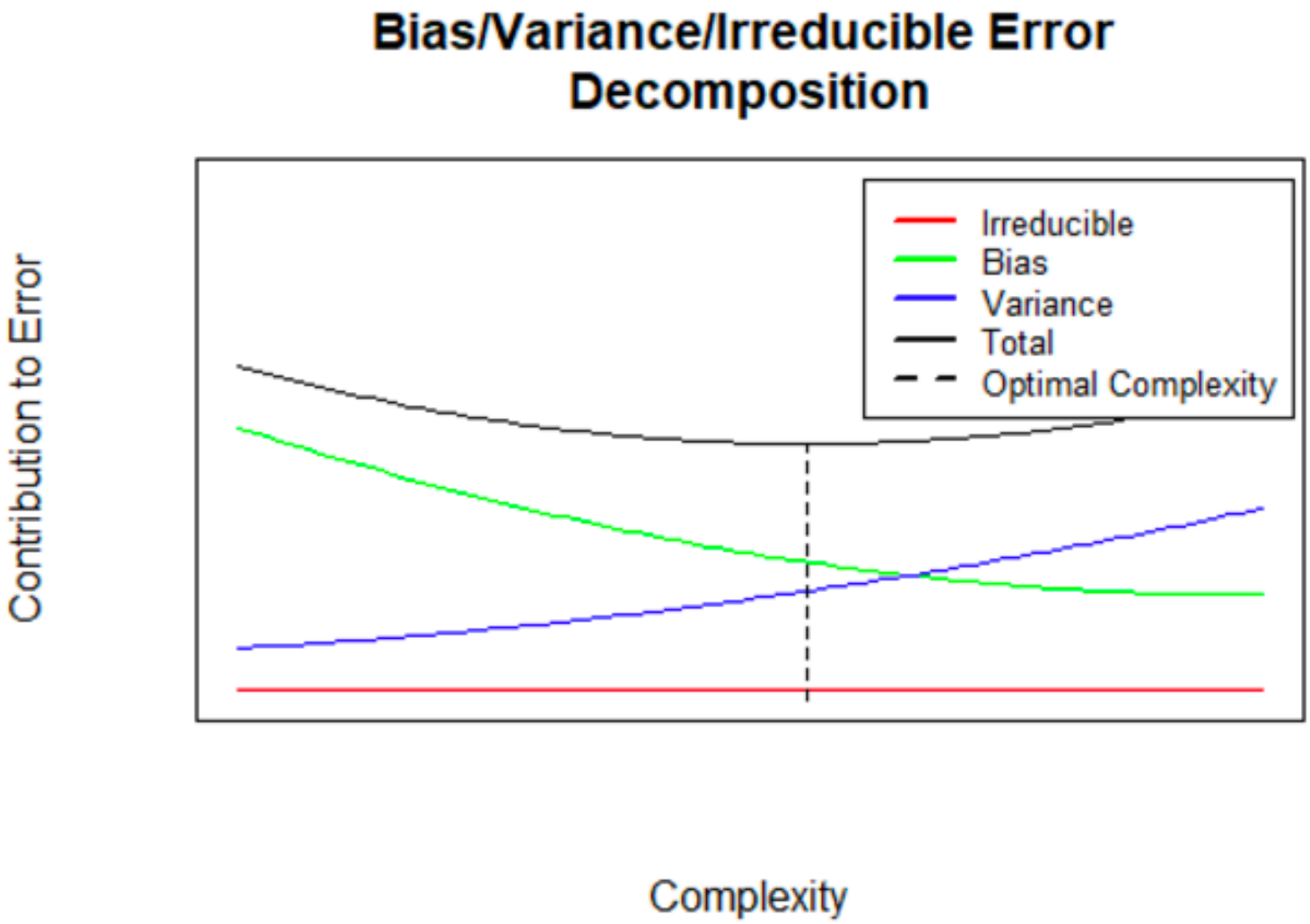
- With our two curves, do we have enough information to judge the situation?
- Not if the achieved training error is far from 0.

- There is a third source of error next to bias and variance:

  The irreducible error $Y = f(X) + irreducible\ error$

- It stems from noise in the data or lack of information (things that influence the outcome but are not represented in your data).

- We try to minimize the reducible error by lowering bias and variance, but we should also have a notion of what the irreducible error is, so we know what is achievable.


Bias/Variance/Irreducible Error Decomposition
Irreducible
Bias
Variance
Total
Optimal Complexity
Contribution to Error
Complexity

Analysis with Irreducible Error

- The theoretically achievable performance is often unknown.

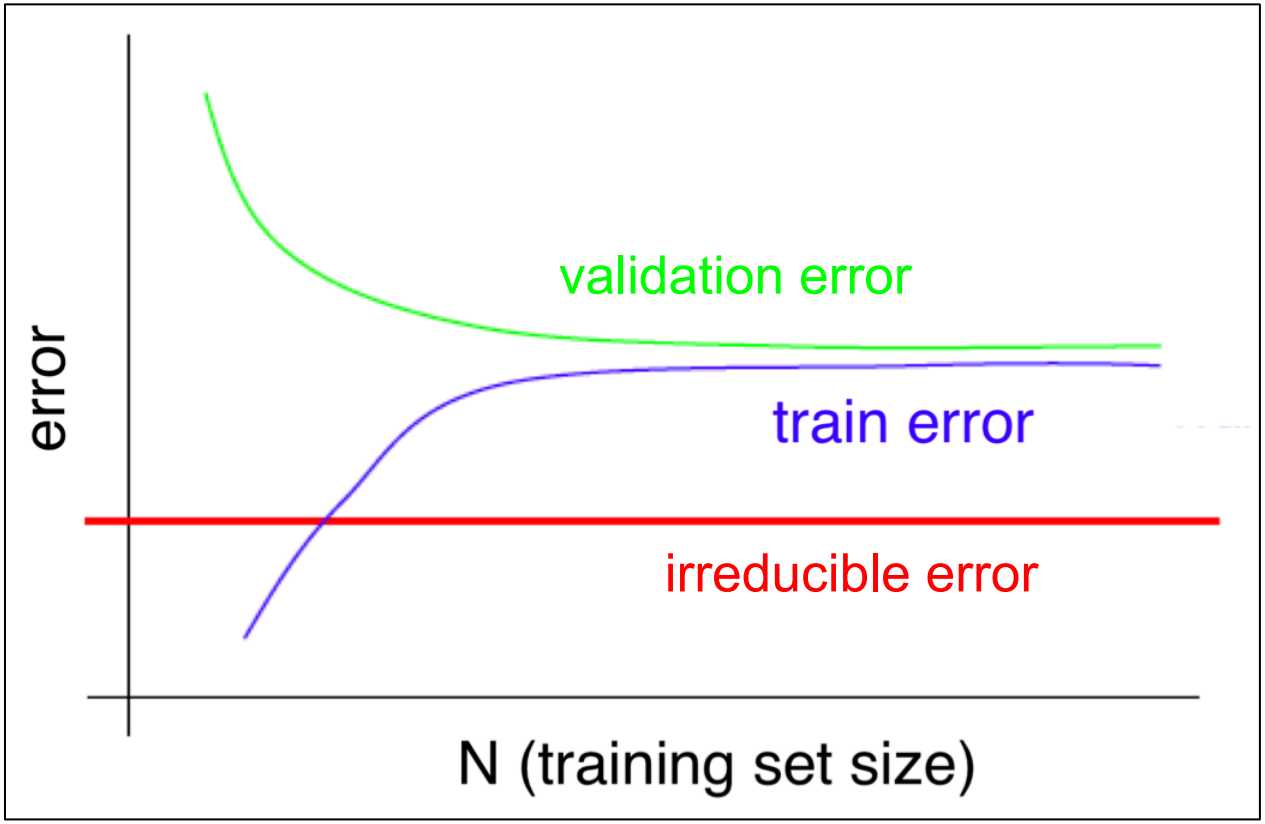
- Except when a human can solve all samples 100% accurately.

- But even if it is unknown, for tasks where humans are very good, their performance can be used to deduce the irreducible error.

- We now have two characteristic measures which we can read from our learning curves:

  1. Difference between Irreducible error and Training error
  2. Difference between Training error and Validation error

Learning Curve for Underfitting

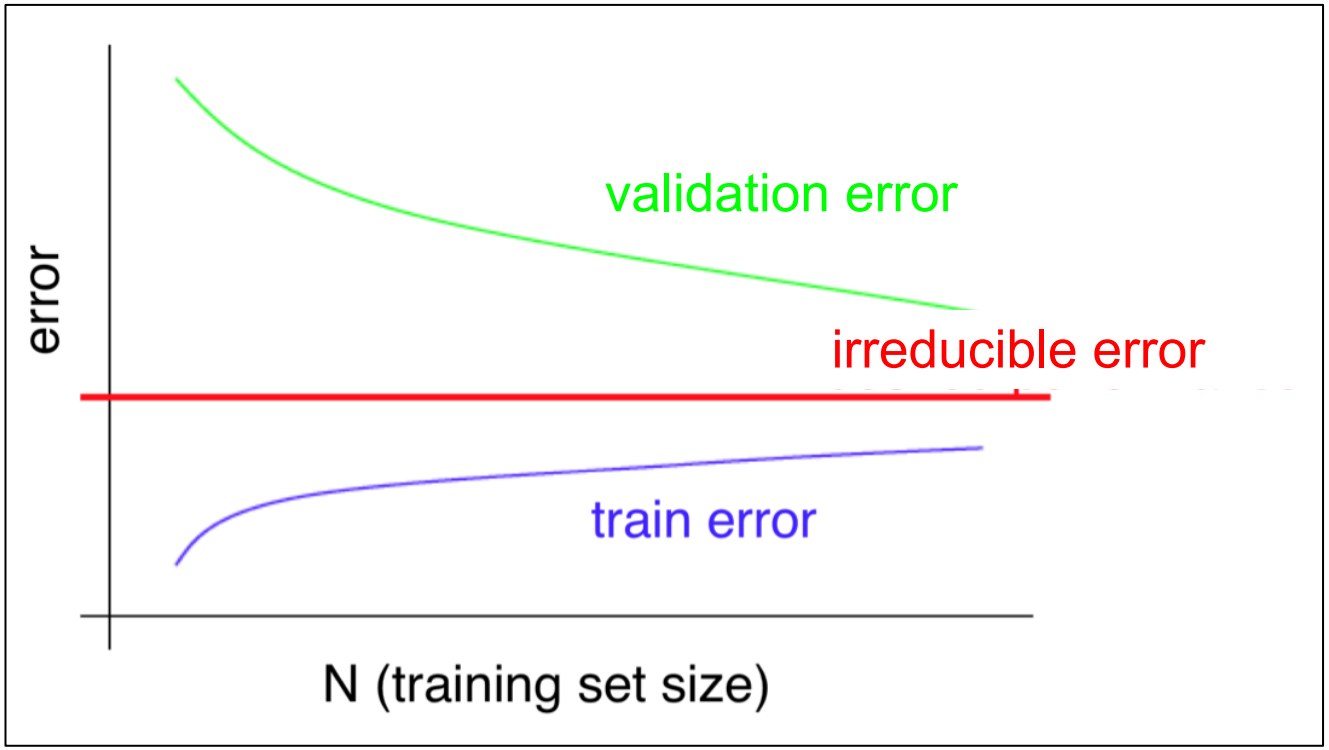


Bad performance on training and test data

Counteracting Underfitting

- Models that underfit the data are often easier to repair
- If your model misses the relevant structure, try make it more complex
  - Try a larger set of features
  - Try a different set of features
  - Try more complex machine learning algorithms
  - …

Learning Curve for Overfitting

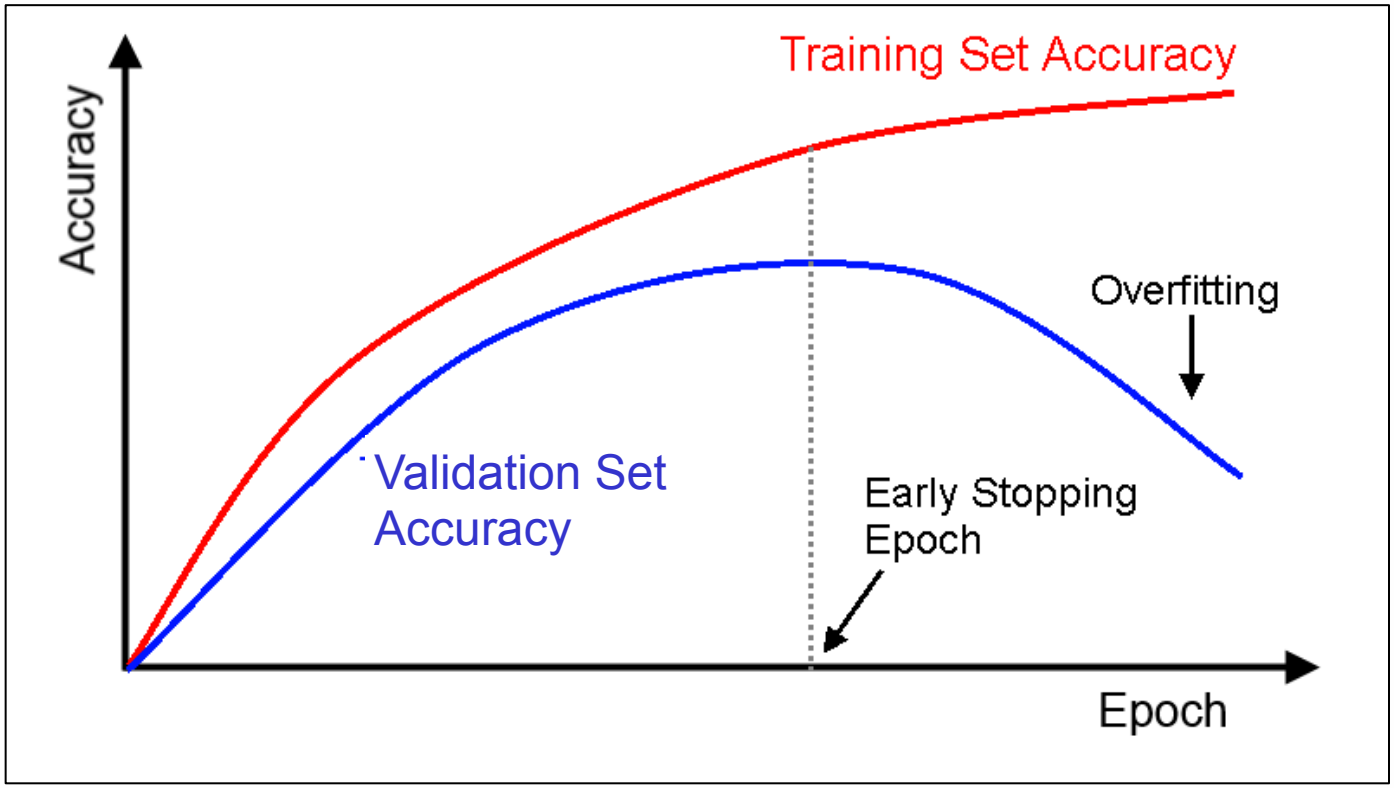


Good performance on training but bad performance on test data

Counteracting Overfitting

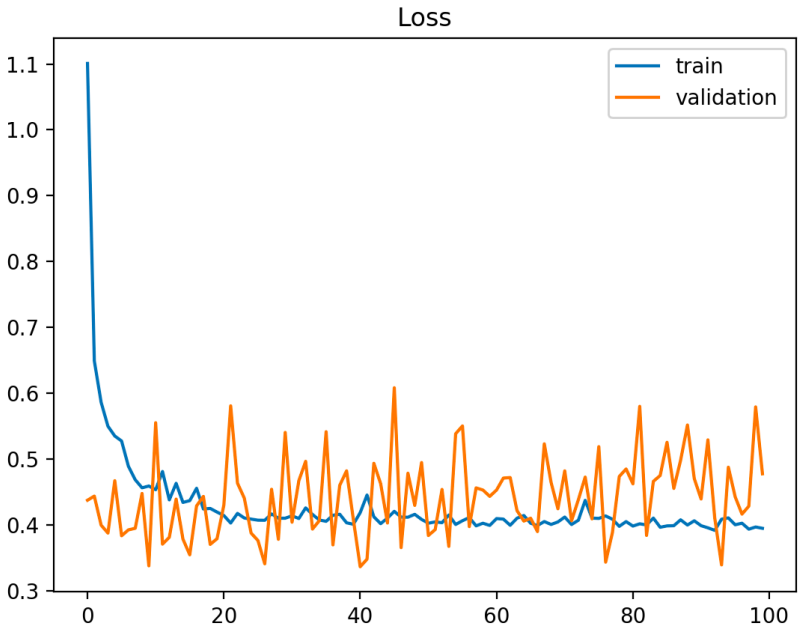
- This happens more frequently and is much harder to repair
- If your model is too sensitive to the training data
  - Try getting more training data
  - Try better data cleaning (e.g. eliminate outliers)
  - Try a smaller set of features through feature selection
  - Try early stopping during iterative training (see next slide)
  - Try tuning the regularization parameter
  - Try ensemble methods
  - …

Early Stopping



Especially important for Deep Learning !

What happened in these two Examples?



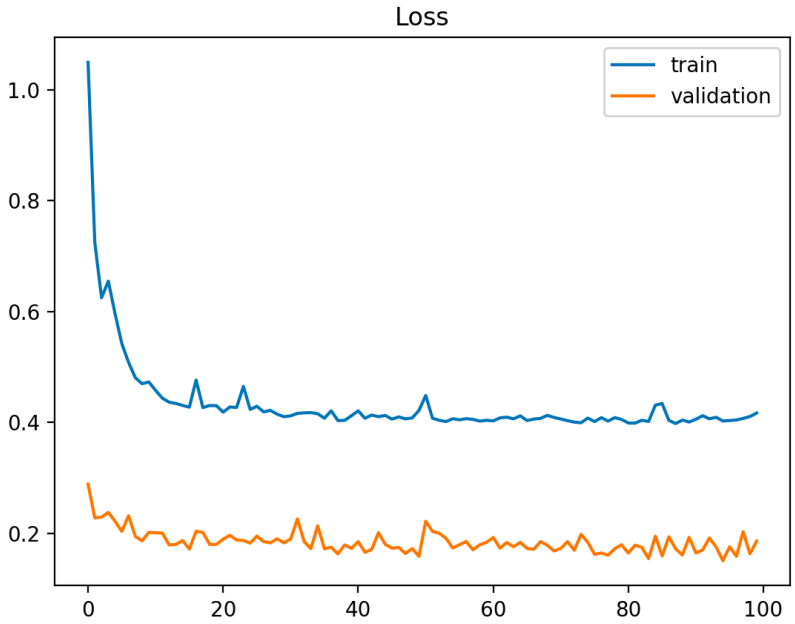


Image source: machinelearningmastery.com

Analyse Distribution Mismatch

- If it is unclear what part of your error is due to variance and what part due to a distribution mismatch, you can do the following:

- Take your training set only and split it into a training-train and a training-validation set, then build and evaluate your existing model on that.

- You then have eliminated the mismatch (provided your splitting method is correct) and can get a notion of how much your model overfits.

Example of How to Split

- You want to build a mobile app that recognizes cats
- You have a bunch of low-resolution cat/not cat pictures taken with mobile cameras, but not enough to train your model
- So you scrape the web and download millions of high-resolution cat/notcat images.

- How do you build your training, validation and test sets?

- Only use mobile camera data: Does not work, not enough data
- Split randomly: Does not work, your score is too optimistic, since your end goal is to classify mobile camera pictures with the app
- Better: Exclusively use mobile pictures for validation and test sets, use all high-res pictures and a few mobile pictures for training.

Error Analysis in Classification

- Pick 100 mislabeled items and analyse them:
  Make a table of the n most frequently occuring problem types (100 x n)

- A problem type might be: Small dogs are mislabeled as cats
- Another one might be: Has problems where head is sideways

- Fix the most frequently occuring problem first

- Mislabeled data can be its own type of problem. If this occurs frequently enough, fix it (but don’t forget the lucky guesses).
  It is more important to fix validation and test set labels though.

- Fix only what appears to be a systematic error

How to Approach a Machine Learning Project

We recommend this approach for industrial projects (not basic research)

1. Data quality assessment is always worth the time
2. Visualize your data
3. Specify and implement a water-proof quality assessment workflow
4. Implement a quick-and-dirty baseline classifier or regressor
5. Run error analysis and diagnostics
   a) If you have a notion of irreducible error, reduce whichever of the two error measures is bigger first, re-evaluate and repeat until you get close to irreducible error
   b) If you do not have this notion, reduce bias, then reduce variance, repeat until things do not improve anymore
   c) If you cannot improve but think there is more potential, perform detailed sample analysis (previous slide)
6. Learn and improve !

How to get Fired as a Data Scientist – Part 2

In [128]:
```
data.head()
```

Out[128]:

	Price	Year	Cylinders	Gears
0	44800	1996	8	5
1	22800	1999	8	5
2	183710	2008	8	6
3	19900	2006	4	6
4	18999	2003	5	5

In [129]:
```
model = KMeans(n_clusters=3).fit(data)

centers = pd.DataFrame(model.cluster_centers_, columns=data.columns)
centers = centers.round(1)
centers['Cluster Size'] = np.bincount(model.labels_)

centers
```

Out[129]:

	Price	Year	Cylinders	Gears	Cluster Size
0	17376.6	2007.9	4.3	5.6	40836
1	55158.1	2011.2	5.1	6.5	13126
2	169479.1	2010.4	8.5	6.7	1246

In [128]:
```
data.head()
```

Out[128]:

	Price	Year	Cylinders	Gears
0	44800	1996	8	5
1	22800	1999	8	5
2	183710	2008	8	6
3	19900	2006	4	6
4	18999	2003	5	5

Do not cluster with unnormalized data

In [129]:
```
model = KMeans(n_clusters=3).fit(data)

centers = pd.DataFrame(model.cluster_centers_, columns=data.columns)
centers = centers.round(1)
centers['Cluster Size'] = np.bincount(model.labels_)

centers
```

Out[129]:

	Price	Year	Cylinders	Gears	Cluster Size
0	17376.6	2007.9	4.3	5.6	40836
1	55158.1	2011.2	5.1	6.5	13126
2	169479.1	2010.4	8.5	6.7	1246

In [45]:
```
data.head()
```

Out[45]:

	Price	Year	Cylinders	Gears	Pink
0	44800	1996	8	5	False
1	22800	1999	8	5	False
2	183710	2008	8	6	False
3	19900	2006	4	6	False
4	18999	2003	5	5	False

In [46]:
```
data['Pink'].value_counts()
```

Out[46]:
```
False    55190
True        18
Name: Pink, dtype: int64
```

In [47]:
```
train, test = train_test_split(data, test_size=0.2)
X_train, X_test = train.drop('Pink', axis=1), test.drop('Pink', axis=1)

scaler = StandardScaler().fit(X_train)
X_train[X_train.columns] = scaler.transform(X_train)
X_test[X_test.columns] = scaler.transform(X_test)
```

In [48]:
```
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform')
knn.fit(X_train, train.Pink)

print('Accuracy %.5f: ready for production !' % accuracy_score(test.Pink, knn.predict(X_test)))
```

```
Accuracy 0.99946: ready for production !
```

In [45]:
```
data.head()
```

Out[45]:

	Price	Year	Cylinders	Gears	Pink
0	44800	1996	8	5	False
1	22800	1999	8	5	False
2	183710	2008	8	6	False
3	19900	2006	4	6	False
4	18999	2003	5	5	False

Do not use accuracy in the presence of skewed data

In [46]:
```
data['Pink'].value_counts()
```

Out[46]:
```
False    55190
True        18
Name: Pink, dtype: int64
```

In [47]:
```
train, test = train_test_split(data, test_size=0.2)
X_train, X_test = train.drop('Pink', axis=1), test.drop('Pink', axis=1)

scaler = StandardScaler().fit(X_train)
X_train[X_train.columns] = scaler.transform(X_train)
X_test[X_test.columns] = scaler.transform(X_test)
```

In [48]:
```
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform')
knn.fit(X_train, train.Pink)

print('Accuracy %.5f: ready for production !' % accuracy_score(test.Pink, knn.predict(X_test)))
```

```
Accuracy 0.99946: ready for production !
```

In [404]:
```
data.head()
```

Out[404]:

	Year	Cylinders	Gears	Price
0	1996	8	5	44800
1	1999	8	5	22800
2	2008	8	6	183710
3	2006	4	6	19900
4	2003	5	5	18999

In [406]:
```
scaler = StandardScaler().fit(data.drop('Price', axis=1))

train, test = train_test_split(data, test_size=0.2)
X_train, X_test = train.drop('Price', axis=1), test.drop('Price', axis=1)

X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)

model = KNeighborsRegressor(n_neighbors=1)
model.fit(X_train, train.Price)
print('R2 is %.2f' % r2_score(test.Price, model.predict(X_test)))
```

```
R2 is 0.59
```

In [404]:
```
data.head()
```

Out[404]:

	Year	Cylinders	Gears	Price
0	1996	8	5	44800
1	1999	8	5	22800
2	2008	8	6	183710
3	2006	4	6	19900
4	2003	5	5	18999

Test set used to determine normalization parameters

In [406]:
```
scaler = StandardScaler().fit(data.drop('Price', axis=1))

train, test = train_test_split(data, test_size=0.2)
X_train, X_test = train.drop('Price', axis=1), test.drop('Price', axis=1)

X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)

model = KNeighborsRegressor(n_neighbors=1)
model.fit(X_train, train.Price)
print('R2 is %.2f' % r2_score(test.Price, model.predict(X_test)))
```

```
R2 is 0.59
```

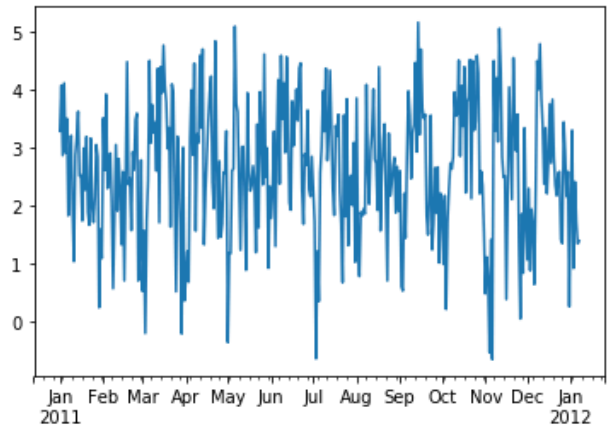
In [433]:
```
data.head()
```

Out[433]:

	state	cycle	target
2011-01-01	5	-1.14	3.294526
2011-01-02	5	0.05	4.084435
2011-01-03	8	0.50	2.872311
2011-01-04	7	1.09	4.122563
2011-01-05	9	0.05	2.917532

In [435]:
```
ts.target.plot()
```

Out[435]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a16b43d30>
```



In [445]:
```
train, test = train_test_split(ts)
X_train, X_test = train.drop('target', axis=1), test.drop('target', axis=1)
model = LinearRegression().fit(X_train, train.target)
print('R2 = %.2f is terribly bad - so is the conceptual error in this code' %
      r2_score(test.target, model.predict(X_test)))
```

```
R2 = -0.05 is terribly bad - so is the conceptual error in this code
```

In [433]:
```
data.head()
```

Out[433]:

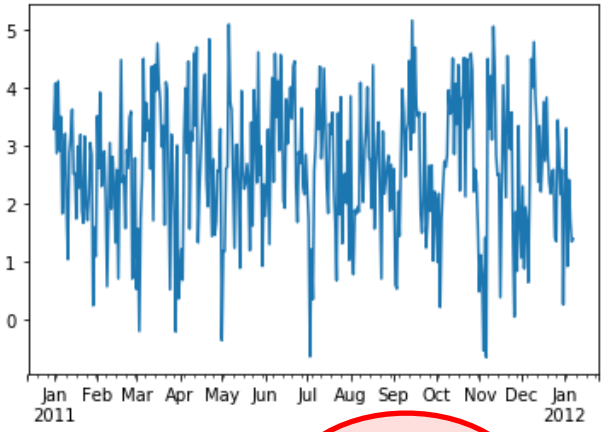
	state	cycle	target
2011-01-01	5	-1.14	3.294526
2011-01-02	5	0.05	4.084435
2011-01-03	8	0.50	2.872311
2011-01-04	7	1.09	4.122563
2011-01-05	9	0.05	2.917532

Randomized splitting

in case of time series

In [435]:
```
ts.target.plot()
```

Out[435]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a16b43d30>
```



In [445]:
```
train, test = train_test_split(ts)
X_train, X_test = train.drop('target', axis=1), test.drop('target', axis=1)
model = LinearRegression().fit(X_train, train.target)
print('R2 = %.2f is terribly bad - so is the conceptual error in this code' %
      r2_score(test.target, model.predict(X_test)))
```

```
R2 = -0.05 is terribly bad - so is the conceptual error in this code
```

```
df.head()
```

	dna_hash	visit_number	blood_pressure	cholesterol	medication	outcome
0	14251	1	120	200	1	0
1	14251	2	122	205	1	0
2	23416	1	130	250	2	1
3	23416	2	128	240	2	1
4	32525	1	110	180	3	1

```
X_train, X_test, y_train, y_test = train_test_split(df.drop("outcome", axis=1),
                                                    df.outcome)

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
accuracy_score(y_test, clf.predict(X_test))
```

```
df.head()
```

	dna_hash	visit_number	blood_pressure	cholesterol	medication	outcome
0	14251	1	120	200	1	0
1	14251	2	122	205	1	0
2	23416	1	130	250	2	1
3	23416	2	128	240	2	1
4	32525	1	110	180	3	1

```
X_train, X_test, y_train, y_test = train_test_split(df.drop("outcome", axis=1),
                                                    df.outcome)

clf = RandomForestClassifier()
clf.fit(X_train, y_train)
accuracy_score(y_test, clf.predict(X_test))
```

In the presence of grouped data, use GroupKfold or GroupShuffleSplit

In [23]:
```
data.head()
```

Out[23]:

	Year	Cylinders	Gears	Price
0	1996	8	5	44800
1	1999	8	5	22800
2	2008	8	6	183710
3	2006	4	6	19900
4	2003	5	5	18999

In [32]:
```
model = [SVR(), LinearRegression(), DecisionTreeRegressor()]
names = ["Support Vector Machine","Linear Regression", "Decision Tree"]

for index in range(0,3) :

    train, test = train_test_split(data, test_size=0.2)
    X_train, X_test = train.drop('Price', axis=1), test.drop('Price', axis=1)

    scaler = StandardScaler().fit(X_train)
    X_train, X_test = scaler.transform(X_train), scaler.transform(X_test)

    model[index].fit(X_train, train.Price)
    print('R2 of '+names[index]+' is %.2f' % r2_score(test.Price, model[index].predict(X_test)))
```

```
R2 of Support Vector Machine is -0.14
R2 of Linear Regression is 0.58
R2 of Decision Tree is -0.02
```

Conclusion: Linear regression performs best on the test set

In [23]:
```
data.head()
```

Out[23]:

	Year	Cylinders	Gears	Price
0	1996	8	5	44800
1	1999	8	5	22800
2	2008	8	6	183710
3	2006	4	6	19900
4	2003	5	5	18999

Split inside loops makes evaluation incomparable

In [32]:
```
model = [SVR(), LinearRegression(), DecisionTreeRegressor()]
names = ["Support Vector Machine","Linear Regression", "Decision Tree"]

for index in range(0,3) :

    train, test = train_test_split(data, test_size=0.2)
    X_train, X_test = train.drop('Price', axis=1), test.drop('Price', axis=1)

    scaler = StandardScaler().fit(X_train)
    X_train, X_test = scaler.transform(X_train), scaler.transform(X_test)

    model[index].fit(X_train, train.Price)
    print('R2 of '+names[index]+' is %.2f' % r2_score(test.Price, model[index].predict(X_test)))
```

```
R2 of Support Vector Machine is -0.14
R2 of Linear Regression is 0.58
R2 of Decision Tree is -0.02
```

Conclusion: Linear regression performs best on the test set

```
# create dataset (no bug here)
(x_train, y_train), (x_val, y_val) = keras.datasets.mnist.load_data()
x_train = x_train.reshape(60000, 784).astype("float32") / 255
x_test = x_test.reshape(10000, 784).astype("float32") / 255
y_train = y_train.astype("float32")
y_test = y_test.astype("float32")
```

```
# create model (no bugs in model architecture)
inputs = keras.Input(shape=(784,), name="digits")
x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
x = layers.Dense(64, activation="relu", name="dense_2")(x)
outputs = layers.Dense(10, activation="softmax", name="predictions")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

# compile the model (no bugs here)
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy",
              metrics=["sparse_categorical_accuracy"])
```

```
# run the cross validation (no bug in any of the parameters)
kfold = KFold(n_splits=10, shuffle=True)
scores = []
for train_idx, test_idx in kfold.split(x_train, y_train):
    model.fit(x_train[train_idx], y_train[train_idx], batch_size=64, epochs=1)
    scores.append(model.evaluate(x_train[test_idx], y_train[test_idx], batch_size=128))
```

```python
# create dataset (no bug here)
(x_train, y_train), (x_val, y_val) = keras.datasets.mnist.load_data()
x_train = x_train.reshape(60000, 784).astype("float32") / 255
x_test = x_test.reshape(10000, 784).astype("float32") / 255
y_train = y_train.astype("float32")
y_test = y_test.astype("float32")
```

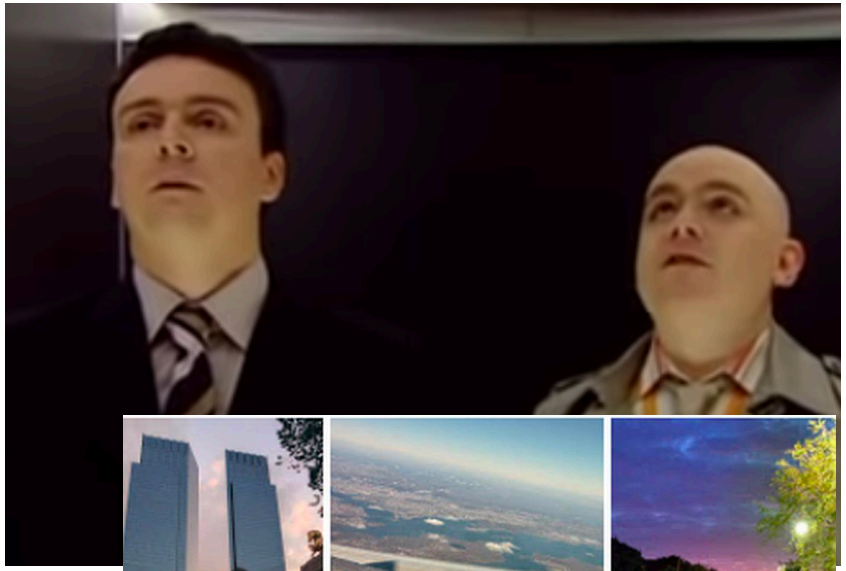
```python
# create model (no bugs in model architecture)
inputs = keras.Input(shape=(784,), name="digits")
x = layers.Dense(64, activation="relu", name="dense_1")(inputs)
x = layers.Dense(64, activation="relu", name="dense_2")(x)
outputs = layers.Dense(10, activation="softmax", name="predictions")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

# compile the model (no bugs here)
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy",
              metrics=["sparse_categorical_accuracy"])
```

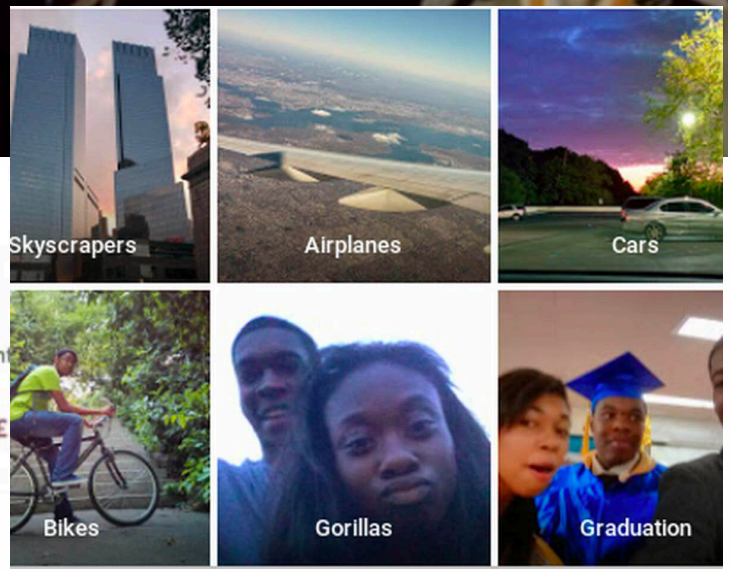
Model not properly reset, each subsequent fold continues training

```python
# run the cross validation (no bug in any of the parameters)
kfold = KFold(n_splits=10, shuffle=True)
scores = []
for train_idx, test_idx in kfold.split(x_train, y_train):
    model.fit(x_train[train_idx], y_train[train_idx], batch_size=64, epochs=1)
    scores.append(model.evaluate(x_train[test_idx], y_train[test_idx], batch_size=128))
```

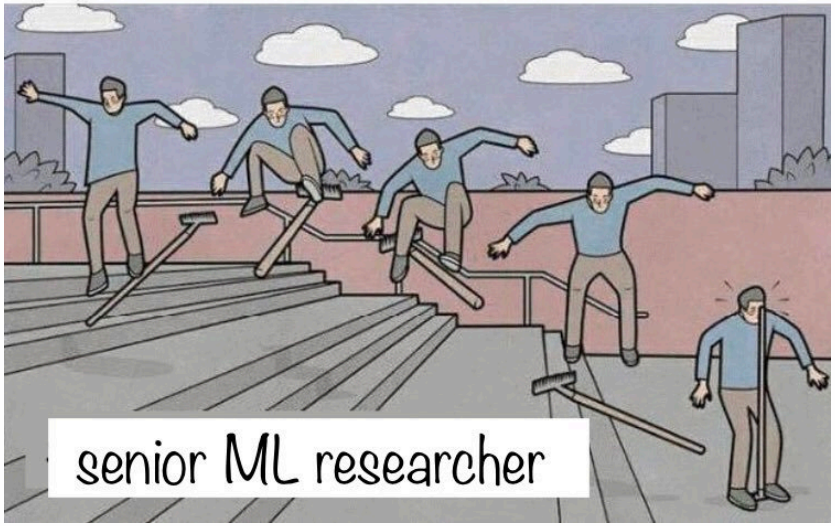
What can go wrong after deployment?

AI Fails will happen!





Good Luck!



Questions ?