model-validation

July 19, 2019

1 Model Validation: How to Know How Much Your Model Knows

- Matthew Emery - Senior Data Scientist @ Imbellus Inc.
- www.matthewemery.ca
- Find the code at https://github.com/lstmemery/lunch-and-learn-validation

1.1 What Are We Covering?

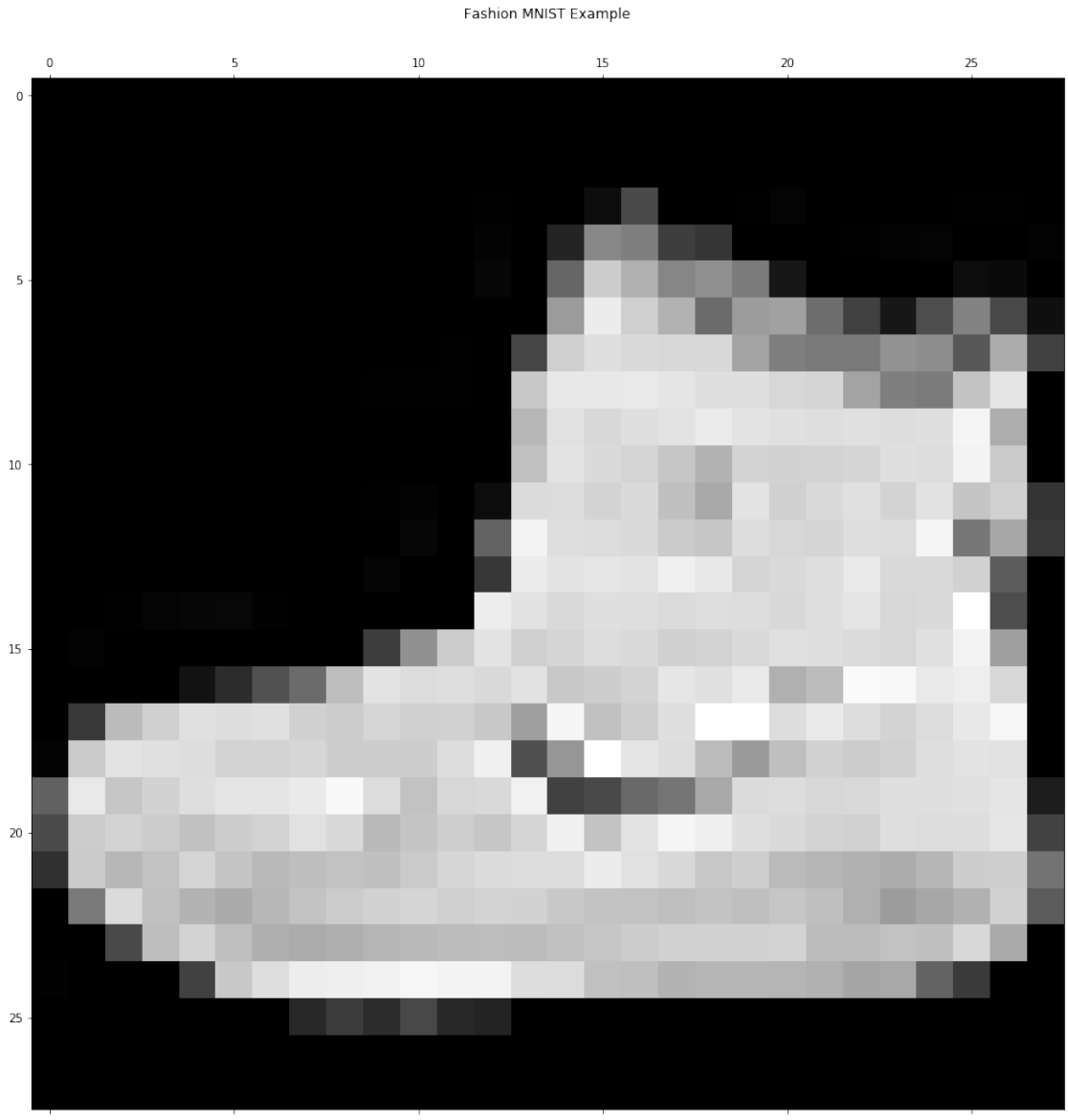
1. A quick explainer on Fashion MNIST and Decision Trees
2. Overfitting Explanation
3. The Golden Rule of Machine Learning
4. Optimization Bias and Cross-Validation

```
In [1]: from sklearn.tree import DecisionTreeClassifier, export_graphviz
        import gzip
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split, cross_val_score, RepeatedStratifi
        from math import log, sqrt
        from pathlib import Path
        import numpy as np
        import bokeh.plotting as bk
        import graphviz

        fmnist_class_names = ["T-shirt", "Trouser", "Pullover", "Dress", "Coat",
                              "Sandal", "Shirt", "Sneaker", "Bag", "Ankle Boot"]

        %matplotlib notebook
```

```
In [2]: with gzip.open(Path("data", "train-labels-idx1-ubyte.gz")) as label_path:
            labels = np.frombuffer(label_path.read(), dtype=np.uint8,
                                   offset=8)

        with gzip.open(Path("data", "train-images-idx3-ubyte.gz")) as image_path:
            features = np.frombuffer(image_path.read(),
                                     dtype=np.uint8,
                                     offset=16).reshape(len(labels), 784)
```

```
In [5]: plt.gray()
        plt.matshow(features[0].reshape((28,28)))
        plt.title("Fashion MNIST Example")
        plt.rcParams['figure.figsize'] = [50, 50]
        plt.show()
```

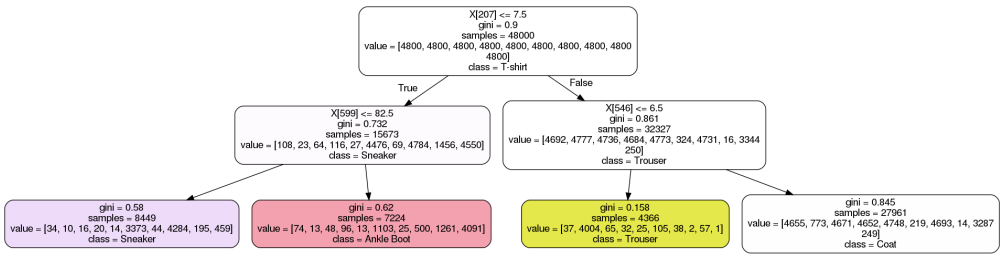
```
<Figure size 3600x3600 with 0 Axes>
```




```
In [6]: features[0]
```

```
Out[6]: array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                 0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   1,   0,   0,  13,  73,   0,   0,   1,
  4,   0,   0,   0,   0,   1,   1,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   3,   0,  36, 136, 127,  62,
 54,   0,   0,   0,   1,   3,   4,   0,   0,   3,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0, 102, 204,
176, 134, 144, 123,  23,   0,   0,   0,   0,  12,  10,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,  72,
 15,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
  0,  69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,
 88, 172,  66,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
  1,   1,   0, 200, 232, 232, 233, 229, 223, 223, 215, 213, 164,
127, 123, 196, 229,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0, 183, 225, 216, 223, 228, 235, 227, 224,
222, 224, 221, 223, 245, 173,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0, 193, 228, 218, 213, 198, 180,
212, 210, 211, 213, 223, 220, 243, 202,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   1,   3,   0,  12, 219, 220, 212, 218,
192, 169, 227, 208, 218, 224, 212, 226, 197, 209,  52,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99, 244, 222,
220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119, 167,  56,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
 92,   0,   0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,
  0, 237, 226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215,
218, 255,  77,   0,   0,   3,   0,   0,   0,   0,   0,   0,   0,
 62, 145, 204, 228, 207, 213, 221, 218, 208, 211, 218, 224, 223,
219, 215, 224, 244, 159,   0,   0,   0,   0,   0,  18,  44,  82,
107, 189, 228, 220, 222, 217, 226, 200, 205, 211, 230, 224, 234,
176, 188, 250, 248, 233, 238, 215,   0,   0,  57, 187, 208, 224,
221, 224, 208, 204, 214, 208, 209, 200, 159, 245, 193, 206, 223,
255, 255, 221, 234, 221, 211, 220, 232, 246,   0,   3, 202, 228,
224, 221, 211, 211, 214, 205, 205, 205, 220, 240,  80, 150, 255,
229, 221, 188, 154, 191, 210, 204, 209, 222, 228, 225,   0,  98,
233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217, 241,
 65,  73, 106, 117, 168, 219, 221, 215, 217, 223, 223, 224, 229,
 29,  75, 204, 212, 204, 193, 205, 211, 225, 216, 185, 197, 206,
198, 213, 240, 195, 227, 245, 239, 223, 218, 212, 209, 222, 220,
221, 230,  67,  48, 203, 183, 194, 213, 197, 185, 190, 194, 192,
202, 214, 219, 221, 220, 236, 225, 216, 199, 206, 186, 181, 177,
172, 181, 205, 206, 115,   0, 122, 219, 193, 179, 171, 183, 196,
204, 210, 213, 207, 211, 210, 200, 196, 194, 191, 195, 191, 198,
192, 176, 156, 167, 177, 210,  92,   0,   0,  74, 189, 212, 191,
```




```
175, 172, 175, 181, 185, 188, 189, 188, 193, 198, 204, 209, 210,
210, 211, 188, 188, 194, 192, 216, 170,   0,   2,   0,   0,   0,
 66, 200, 222, 237, 239, 242, 246, 243, 244, 221, 220, 193, 191,
179, 182, 182, 181, 176, 166, 168,  99,  58,   0,   0,   0,   0,
  0,   0,   0,   0,   0,  40,  61,  44,  72,  41,  35,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  0,   0,   0,   0],  dtype=uint8)
```

- When I first wrote this tutorial, I used MNIST
- Then I found out about Fashion MNIST
- Benefits:

    1. Harder than MNIST
    2. Less used than MNIST
    3. Better represents modern computer vision tasks

- How many of you have heard of MNIST?

1.2 A Quick Explainer on Decision Trees

- A decision tree is trained by looking at each pixel value and seeing what breakpoint would split classes the best
- Pick the best pixel to split on and continue
- The size of the tree is a hyperparameter
- I decided to choose a decision tree to illustrate this for a couple of reasons

1. Decision Trees are easy to understand
2. No need to do any preprocessing
3. It shows that even simple models can do simple computer vision tasks

1.3 Train Test Split

```
In [4]: (train_features, validation_features,
         train_labels, validation_labels) = train_test_split(
            features,
```

```
            labels,
            random_state=0,
            shuffle=True,
            test_size=0.20, # This is fine enough
            stratify=labels
        )
```

- This is the core way we evaluate models
- If we train on one dataset, how well does it do on the holdout?
- Always set a seed so you can compare
- Test size is traditionally 80/20 - this mostly folklore though
- Do not shuffle if you are dealing with time series
- Stratification is essential when you are dealing with unbalanced labels

In [5]:
```
models, train_scores = [], []

for depth in range(1, 20):
    model = DecisionTreeClassifier(max_depth=depth, random_state=0)
    model.fit(train_features, train_labels)

    score = model.score(train_features, train_labels)
    print("Depth:", depth, "Training Accuracy:", round(score, 2))
    models.append(model); train_scores.append(score)
```

```
Depth: 1 Training Accuracy: 0.2
Depth: 2 Training Accuracy: 0.36
Depth: 3 Training Accuracy: 0.5
Depth: 4 Training Accuracy: 0.65
Depth: 5 Training Accuracy: 0.71
Depth: 6 Training Accuracy: 0.74
Depth: 7 Training Accuracy: 0.78
Depth: 8 Training Accuracy: 0.81
Depth: 9 Training Accuracy: 0.83
Depth: 10 Training Accuracy: 0.85
Depth: 11 Training Accuracy: 0.87
Depth: 12 Training Accuracy: 0.89
Depth: 13 Training Accuracy: 0.91
Depth: 14 Training Accuracy: 0.93
Depth: 15 Training Accuracy: 0.94
Depth: 16 Training Accuracy: 0.96
Depth: 17 Training Accuracy: 0.97
Depth: 18 Training Accuracy: 0.98
Depth: 19 Training Accuracy: 0.98
```

In [6]:
```
# This is how the first Decision Tree Illustration Was Made
graph_data = export_graphviz(models[1],
                             filled=True,
                             rounded=True,
```

```
                                    class_names=fmnist_class_names)
        graph = graphviz.Source(graph_data, format="png")
        graph.render("decision_tree", "img")
```

```
Out[6]: 'img/decision_tree.png'
```

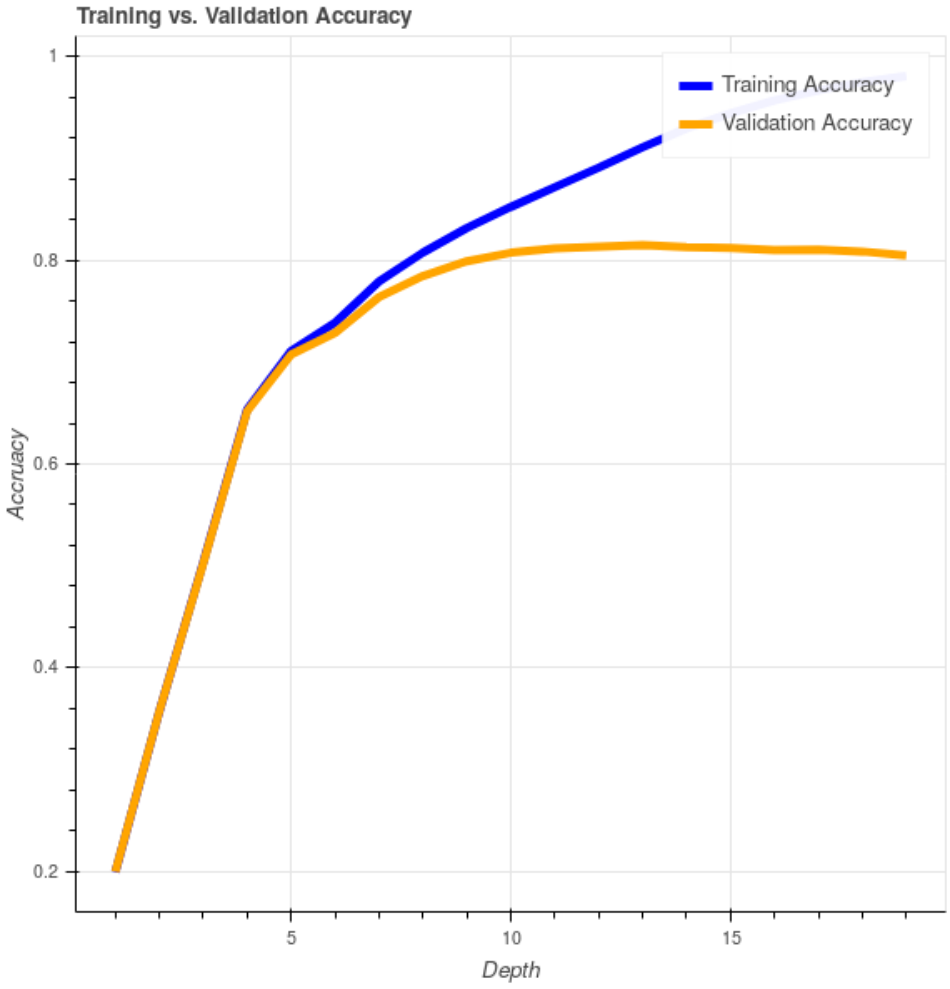
```
In [7]: validation_scores = []

        for depth, model in enumerate(models):
            score = model.score(validation_features, validation_labels)
            print("Depth:", depth, "Validation Accuracy:", round(score, 2))
            validation_scores.append(score)
```

```
Depth: 0 Validation Accuracy: 0.2
Depth: 1 Validation Accuracy: 0.36
Depth: 2 Validation Accuracy: 0.5
Depth: 3 Validation Accuracy: 0.65
Depth: 4 Validation Accuracy: 0.71
Depth: 5 Validation Accuracy: 0.73
Depth: 6 Validation Accuracy: 0.76
Depth: 7 Validation Accuracy: 0.78
Depth: 8 Validation Accuracy: 0.8
Depth: 9 Validation Accuracy: 0.81
Depth: 10 Validation Accuracy: 0.81
Depth: 11 Validation Accuracy: 0.81
Depth: 12 Validation Accuracy: 0.82
Depth: 13 Validation Accuracy: 0.81
Depth: 14 Validation Accuracy: 0.81
Depth: 15 Validation Accuracy: 0.81
Depth: 16 Validation Accuracy: 0.81
Depth: 17 Validation Accuracy: 0.81
Depth: 18 Validation Accuracy: 0.8
```

```
In [8]: plot_1 = bk.figure(title="Training vs. Validation Accuracy",
            )
        plot_1.xaxis.axis_label = "Depth"
        plot_1.yaxis.axis_label = "Accruacy"
        plot_1.line(x = range(1, 20), y = train_scores,
                    line_width = 5, color = "blue", legend = "Training Accuracy")
        plot_1.line(x = range(1, 20), y = validation_scores,
                    line_width = 5, color = "orange", legend = "Validation Accuracy")
        bk.show(plot_1)
```

- If you want to show Bokeh plots inline you need

```
jupyter labextension install jupyterlab_bokeh
```


Training vs. Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Depth
1
0.8
0.6
0.4
0.2
5
10
15

1.4 Overfitting

- The gap between training and validation accuracy is **overfitting**
- **Interpretation**: Our model has memorized part of the data set instead of learning the underlying rules
- If the validation accuracy was higher than our training that's **underfitting**

1.5 Think About Studying for an Exam

- Training your model is like the model reviewing its notes
- Validating your model is when you take the midterm
- Deploying your model is the final

1.6 Golden Rule of Machine Learning:

2 The test cannot influence training in any way

- If you know the answers on the exam ahead of time, you won't know if you actually learned the material

2.1 Common Mistakes

- Time Series: Incorporating information from the future in your model (i.e., quarterly results before end-of-quarter)
- Imputing based on the combined train-test dataset
- Taking a peek on the test data halfway through training your model

2.2 Should I Just Fit a Million Models Until I Find Something?

3 No!

3.1 An Illustration

- Sign up for my service, and I'll email you a prediction of whether or not the S&P 500 goes up or down that morning
- Every work day for two weeks I'm right
- **What do you need to ask me before you should trust my model?**

3.2 How Many Other People Did I Send Emails To?

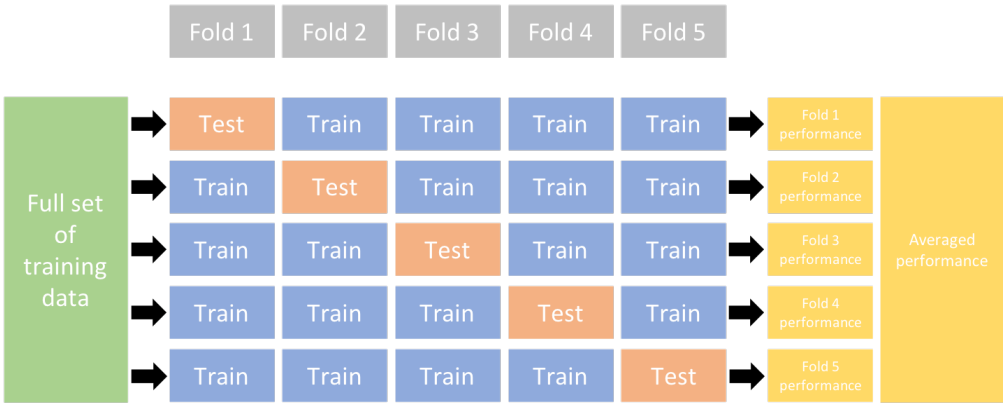
- Two business weeks is 10 business days

$$2^{10} = 1024$$

- If I sent 1024 people different emails (Up, Down, Up, etc.) I'm guaranteed to be right once

3.3 The Same Thing Happens with Machine Learning Models

- We call this **optimization bias**
- Sometimes you find something that fits your validation data set through dumb luck



https://bradleyboehmke.github.io/hands-on-machine-learning-with-r/regression-performance.html

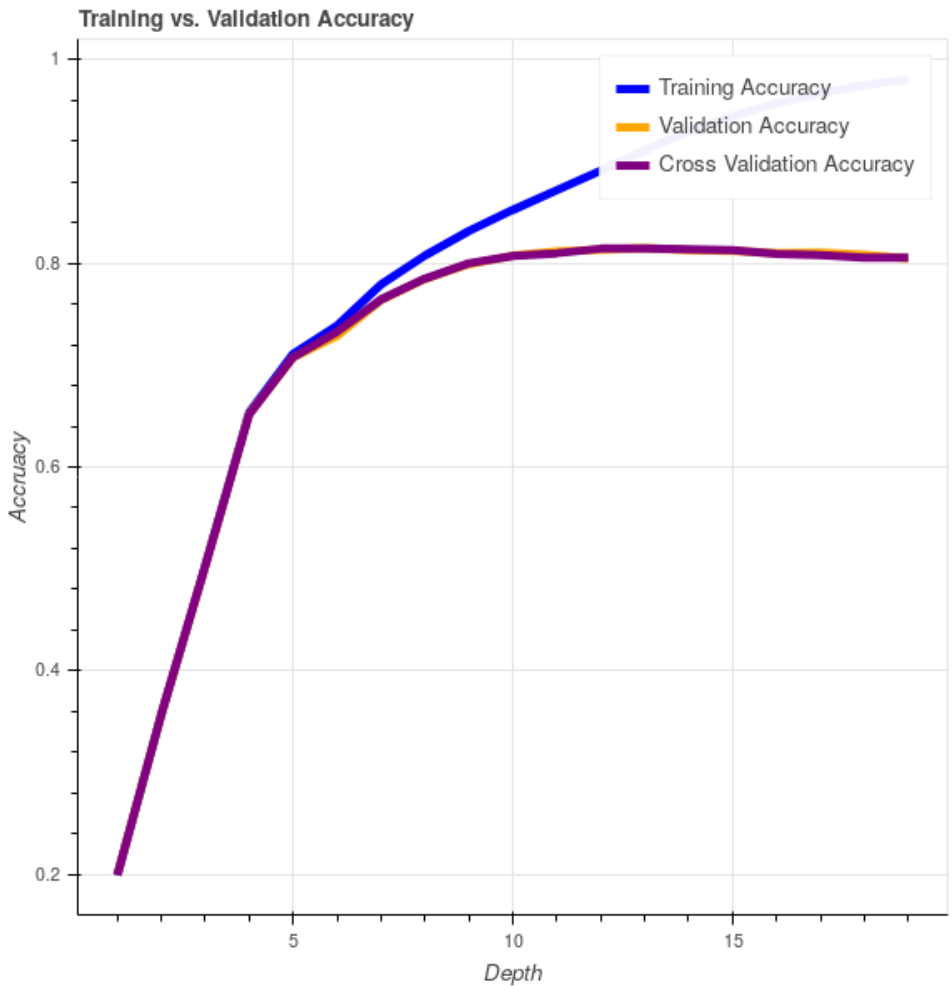
3.4 How Do We Decrease the Effect of Optimization Bias?

- **Cross-validation** (or repeated cross-validation)
- Leave a **test set** that you evaluate very rarely (once a week or less)
- Set a limit to the number of models you will evaluate

3.5 Cross-Validation

```
In [9]: cv_scores = []
        for depth in range(1, 20):
            model = DecisionTreeClassifier(max_depth=depth, random_state=0)

            cv_score = cross_val_score(
                model, features, labels, cv = 5, n_jobs = -1
            )

            print("Depth:", depth, "Mean:", round(np.mean(cv_score), 2),
                  "SD:", round(np.std(cv_score), 3))

            cv_scores.append(cv_score)
```

```
Depth: 1 Mean: 0.2 SD: 0.0
Depth: 2 Mean: 0.36 SD: 0.002
Depth: 3 Mean: 0.5 SD: 0.003
Depth: 4 Mean: 0.65 SD: 0.002
Depth: 5 Mean: 0.71 SD: 0.003
Depth: 6 Mean: 0.73 SD: 0.003
Depth: 7 Mean: 0.76 SD: 0.003
Depth: 8 Mean: 0.79 SD: 0.002
Depth: 9 Mean: 0.8 SD: 0.002
Depth: 10 Mean: 0.81 SD: 0.003
Depth: 11 Mean: 0.81 SD: 0.004
Depth: 12 Mean: 0.81 SD: 0.003
```




```
Depth: 13 Mean: 0.81 SD: 0.003
Depth: 14 Mean: 0.81 SD: 0.002
Depth: 15 Mean: 0.81 SD: 0.003
Depth: 16 Mean: 0.81 SD: 0.004
Depth: 17 Mean: 0.81 SD: 0.004
Depth: 18 Mean: 0.81 SD: 0.004
Depth: 19 Mean: 0.81 SD: 0.005
```

```
In [10]: plot_2 = plot_1

         plot_2.line(x = range(1, 20), y = np.mean(cv_scores, axis=-1),
                     line_width = 5, color = "purple", legend = "Cross Validation Accuracy")
         bk.show(plot_2)
```

- Notice that this doesn't change much
- This is because we already have sufficient data to get an accurate result

- Try this on a smaller dataset
- This is good news, cross-validation takes longer than validation

3.6 Summary

- Always create a validation set (cross-validation if you have a small amount of data)
- Never let information from your validation set leak into to your training
- Don't train models for no reason

3.7 Bibliography

[1] "Data splitting | Machine Learning." [Online]. Available: https://www.includehelp.com/ml-ai/data-splitting.aspx. [Accessed: 17-Mar-2019].

[2]"1.10. Decision Trees — scikit-learn 0.20.3 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#tree. [Accessed: 17-Mar-2019].

[3]M. Schmidt, "DSCI 573: Model Selection and Feature Selection 1."

[4]T. Sarkar, "How to analyze 'Learning': Short tour of computational learning theory," Towards Data Science, 26-Oct-2018. [Online]. Available: https://towardsdatascience.com/how-to-analyze-learning-short-tour-of-computational-learning-theory-9d93b15fc3e5. [Accessed: 03-Mar-2019].

[5]A MNIST-like fashion product database. Benchmark :point_right: : zalandoresearch/fashion-mnist. Zalando Research, 2019.

[6]I. Guyon and T. B. Laboratories, "A scaling law for the validation-set training-set size ratio," p. 11.