outrageously
**AMBITIOUS**

# AIPI520 Week 2: Modeling Process

Duke
PRATT SCHOOL of ENGINEERING

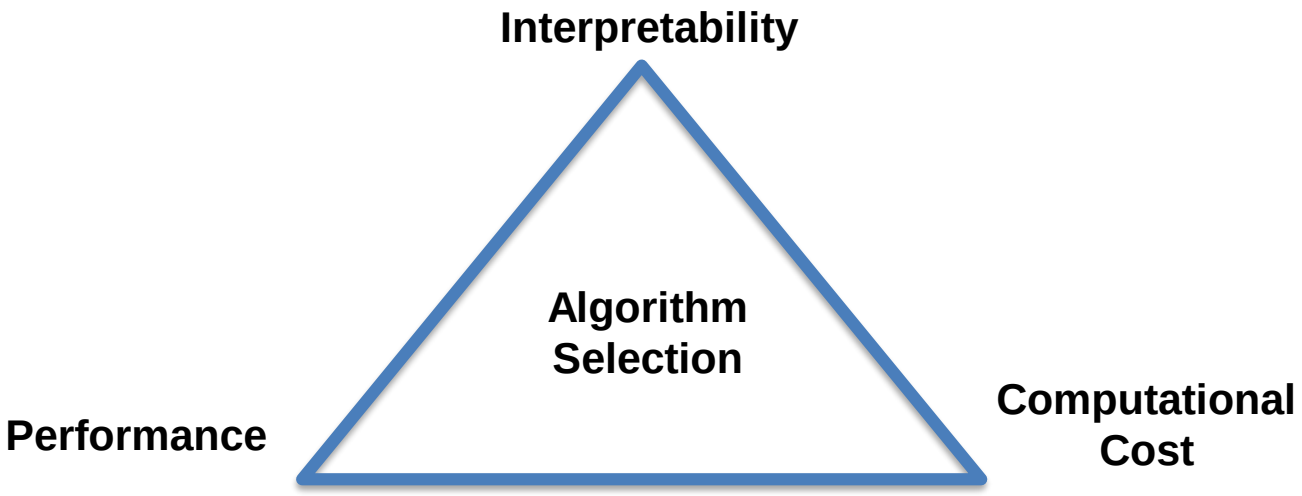
outrageously
**AMBITIOUS**

# Modeling Process

Duke
PRATT SCHOOL *of*
ENGINEERING

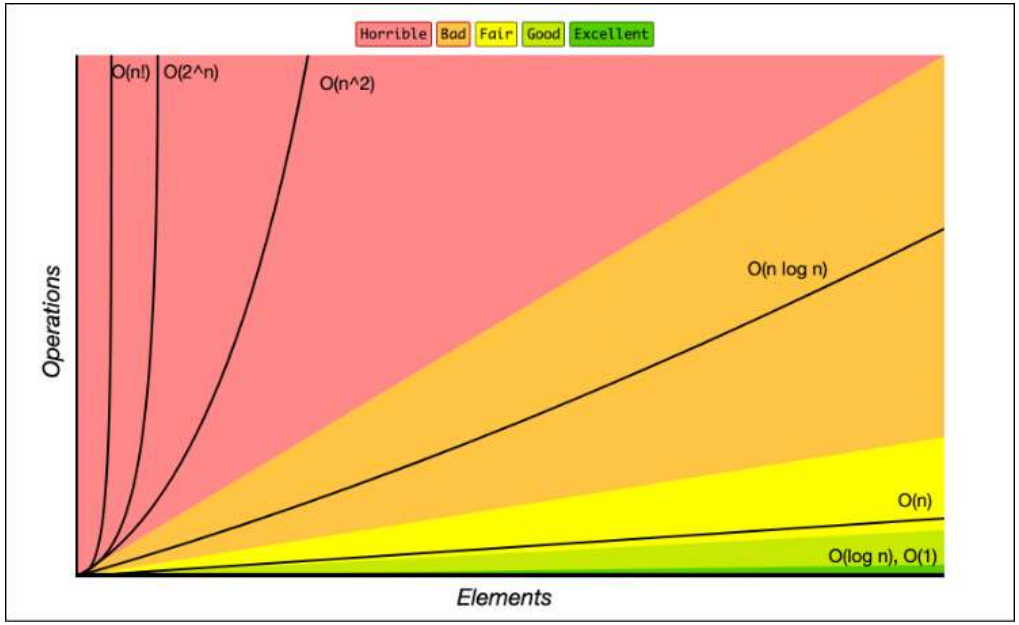
# Algorithm selection

- Depending on our task, we have a variety of algorithms to choose from
- No one algorithm is always better ("no free lunch theorem")
- Our selection will depend on three criteria:



# Computational cost

- Time + memory
- Training
- Inference



# Performance vs Interpretability

"Suppose you have cancer and you have to choose between a black-box AI surgeon that cannot explain how it works but has a 90% cure rate and a human surgeon with an 80% cure rate. **Do you want the AI surgeon to be illegal**?"

- Geoffrey Hinton

# Occam’s Razor

**All else being equal, choose the simpler solution.**

https://en.wikipedia.org/wiki/Occam's_razor

Duke
PRATT SCHOOL of ENGINEERING

**Brandon Rohrer**
@_brohrer_

ML strategy tip

When you have a problem, build two solutions - a deep Bayesian transformer running on multicloud Kubernetes and a SQL query built on a stack of egregiously oversimplifying assumptions. Put one on your resume, the other in production. Everyone goes home happy.

6:45 AM · Aug 12, 2021 · Twitter for iPhone

**781** Retweets **133** Quote Tweets **4,316** Likes

**Daniel Tompkins**
@TompkinsDaniel

Replying to @_brohrer_

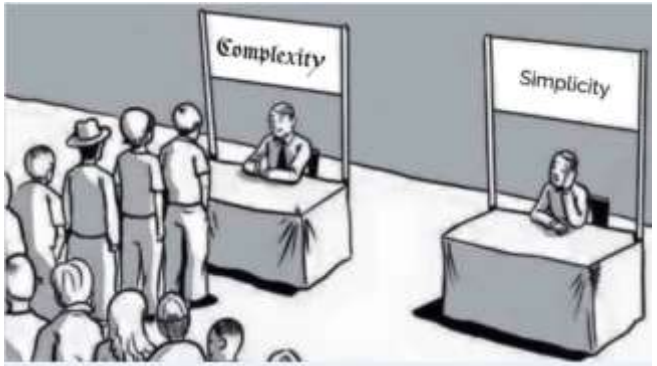
Sometimes the things that are good for making a product aren't things that are attractive as paper topics. I'd like to see more papers and conferences give space for "industry relevance."

5 bil param; SOTA —> publish it
5 mil param; 98% SOTA —> ship it

11:07 AM · Aug 28, 2022 · Twitter for iPhone

# Why are simpler systems better

- Easier to build, scale and maintain
- Lower operational costs
- Generalize better and are more robust to data drift
- Do not necessarily perform worse than complex models
  - Tree-based models > deep neural networks on 45 mid-sized tabular datasets
  - Simple averaging ≥ complex optimizers on multi-task learning problems
  - Simple methods > complex methods in forecasting accuracy across 32 papers
  - Dot product > neural collaborative filtering in item recommendation and retrieval




*Recommended article: https://eugeneyan.com/writing/simplicity/*

# Algorithm selection best practice

1. Start with developing a simple but reasonable heuristic, and measure the performance using your heuristic to make predictions
2. Next, apply a simple ML algorithm (e.g. linear regression, or KNN for classification) and evaluate performance to get a baseline
3. Now, experiment with different algorithms, keeping in mind the right balance between your three goals:
   - Performance
   - Interpretability
   - Computational efficiency

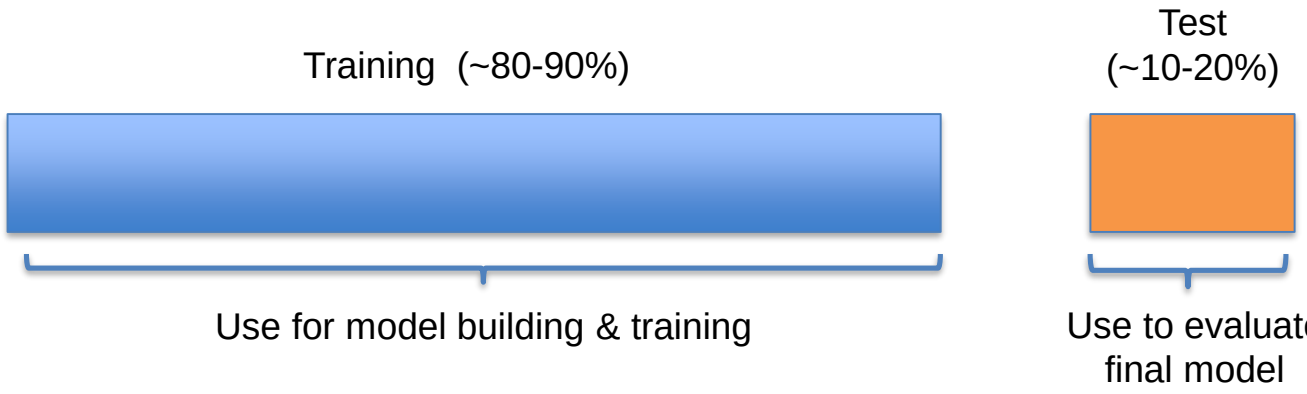
outrageously
**AMBITIOUS**

# Model Evaluation

Duke
PRATT SCHOOL of ENGINEERING

# Training & Test Sets

- Goal of predictive modeling is to create a model that makes accurate predictions on new unseen data
- We cannot estimate performance on data we do not have, so instead we split our data into two sets
  - **Training set** - build and train the model
  - **Test set** – Evaluate model performance performance




Duke
PRATT SCHOOL of ENGINEERING

# Sampling

- Why do we sample?
  - We don't have access to all possible data
  - It is infeasible to process all the data we have access to

- Sampling occurs at multiple points
  1. Sampling from the entire population to create training data
  2. Sampling from the training data to create a test set
  3. Sampling from the training data to do quick experiments

- Two main types of sampling: **nonprobability sampling** and **random sampling**

# Nonprobability sampling

- Nonprobability sampling is when the selection of data is not based on any probability criteria
- Types of nonprobability sampling:
  - **Convenience sampling** – samples selected based on their availability
    - E.g. Google Image Search for pictures of flowers
  - **Snowball sampling** – Future samples selected based on existing samples
    - E.g. scraping Twitter accounts based on users you follow
  - **Judgment sampling** – Domain experts decide what samples to use
  - **Quota sampling** – Select samples based on quotas for sub-groups
    - E.g. a survey where you collect 100 responses from users of each age group

# Nonprobability sampling

- Samples are not representative of real-world data and thus are highly likely to cause bias in your model
- However, this is a very common approach in practice
- Examples:
  - Large language models are trained on easily collected data – Wikipedia, Reddit, Common Crawl etc
  - Sentiment analysis models often trained on IMDB movie reviews or Amazon reviews – biased towards people who leave reviews online
  - Self-driving cars – initially trained only on Phoenix, AZ and Bay Area. Waymo later expanded to include Kirkland, WA for rainy weather

# Random sampling

- Random sampling involves randomly choosing samples from population
- Several types:
  - **Simple random sampling** – all samples have equal probability of being selected
    - Downside – certain rare categories may not appear in the sampled data
  - **Stratified sampling** – divide population into groups you care about and sample a % randomly from each group
    - One challenge is that some observations may fall into multiple groups
  - **Weighted sampling** – each sample is given a weight which determines the probability of being selected
    - Allows you to apply domain expertise in selection e.g. weight recent data higher
    - Helps when your available data has a different distribution than the true data

# Sampling for test set

- Take a random sample of indices to use for test set, use remainder for training
- Use Scikit-learn's train_test_split method

```
X_train,X_test,y_train,y_test = train_test_split(X, y, random_state=0,test_size=0.15)
```

- If we want to do stratified sampling we use the 'stratify' parameter

```
X_train,X_test,y_train,y_test = train_test_split(X, y, random_state=0,test_size=pct,stratify=y)
```

Duke
PRATT SCHOOL of ENGINEERING

# Data Leakage

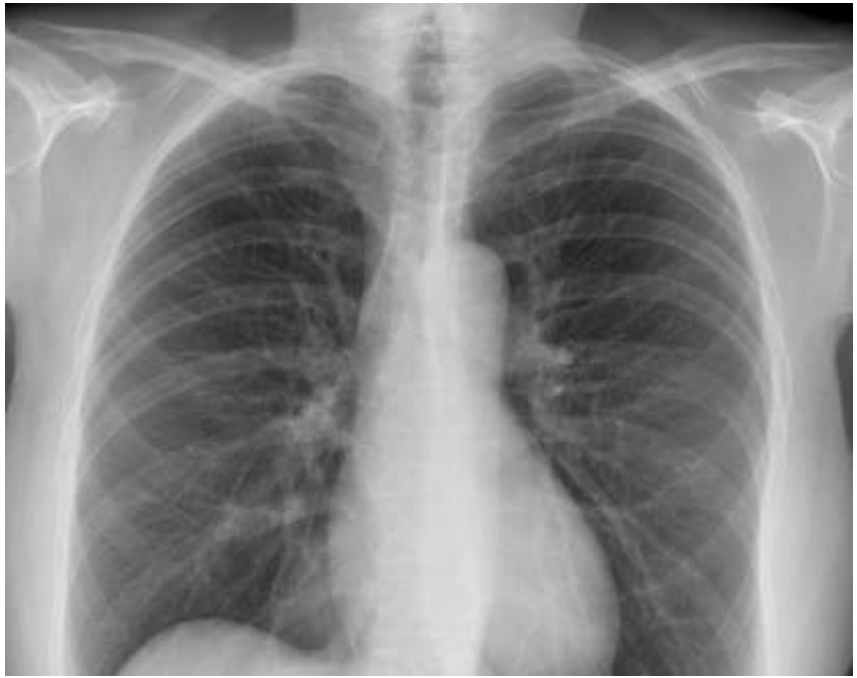
- **"Data leakage"** occurs when some of our test set data "leaks" into model building and influences the development of the model
- For example, if we use all of our data to select our features, or compare algorithms
- This **invalidates the estimated performance** of the model and causes it to be overoptimistic
- In severe cases, it can cause our model to be worthless

# Data Leakage


- Data from the future is accidentally used within training data

# Data Leakage

- A variable within the training data is an indicator of the target



# Common causes of data leakage

**Not good but not terrible**

- Selecting features before splitting
- Scaling before splitting
- Filling missing values before splitting
- Grouped observations are separated into different splits
  - E.g. two CT scans from the same patient taken a week apart

**Very severe**

- Splitting time series data randomly instead of by time
- Failure to remove duplicate observations
  - Particularly dangerous when merging multiple datasets together
  - Common issue in several studies using ML to detect COVID-19
- Different data generation processes for different classes

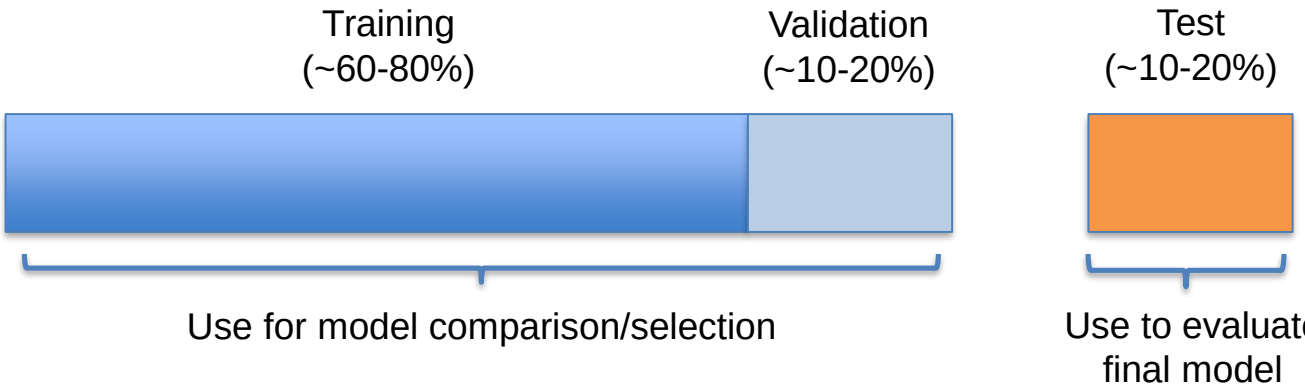
outrageously
**AMBITIOUS**

# Comparing Models

Duke
PRATT SCHOOL of ENGINEERING

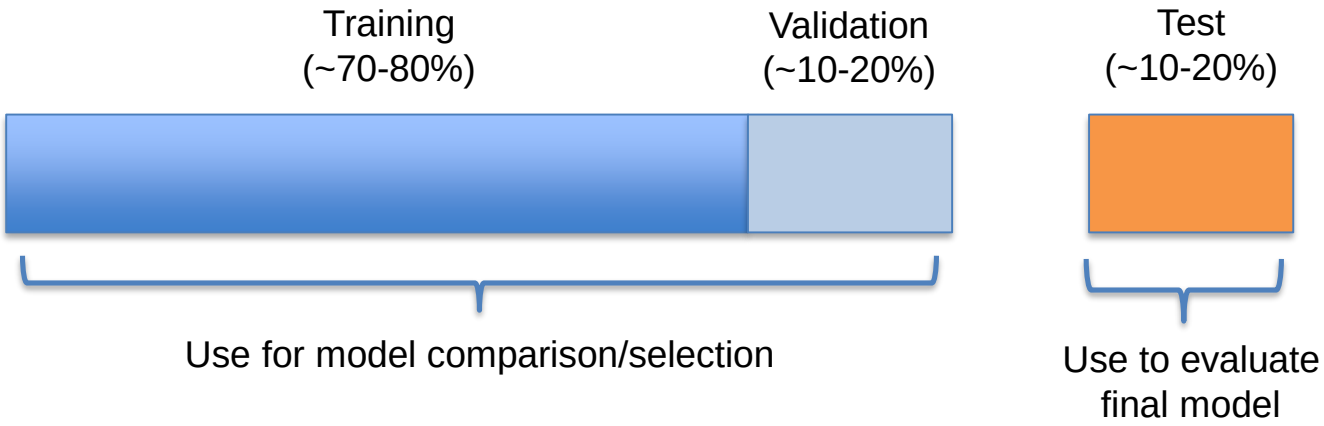
# Comparing models- validation sets

- Often we want to compare models to select the optimal model
- If we use the test set to compare model performance, it is not longer an unbiased indicator of performance
- Instead, we split the data 3 ways into training, validation and test sets
- We use the validation set for model selection, and report performance on the test set



# Question

What might be some potential problems with using a validation set to do your model selection?



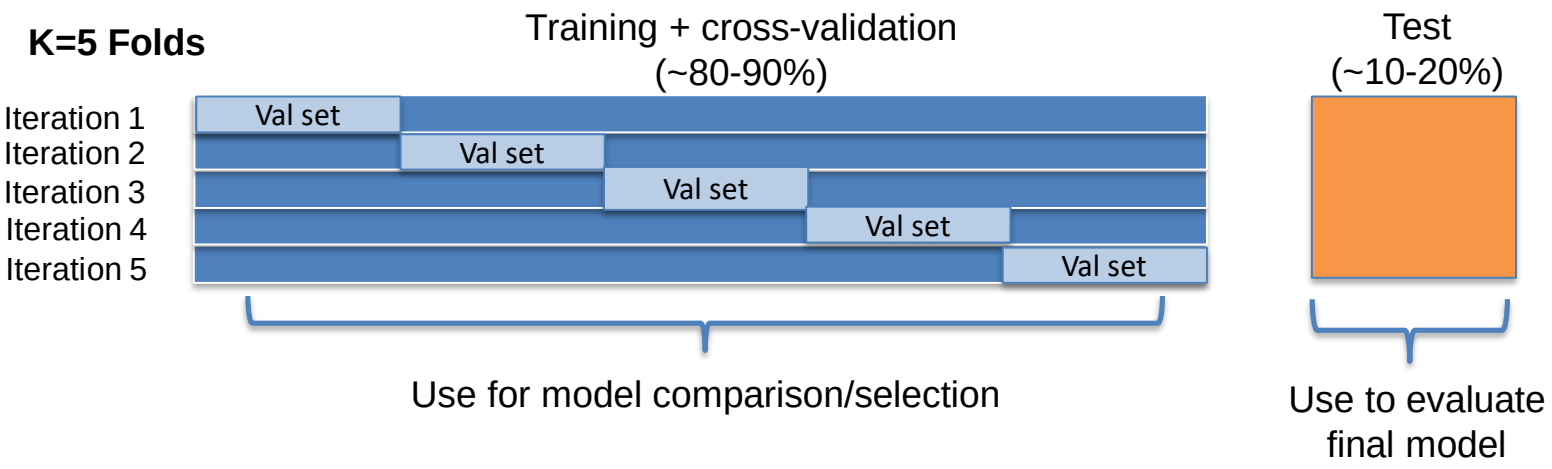
outrageously
**AMBITIOUS**

# Cross-Validation

Duke
PRATT SCHOOL of ENGINEERING

# K-Folds Cross Validation

Rather than using a fixed validation set, we train and run the model(s) multiple times, each time using a different subset ("fold") as the validation set




K Validation fold

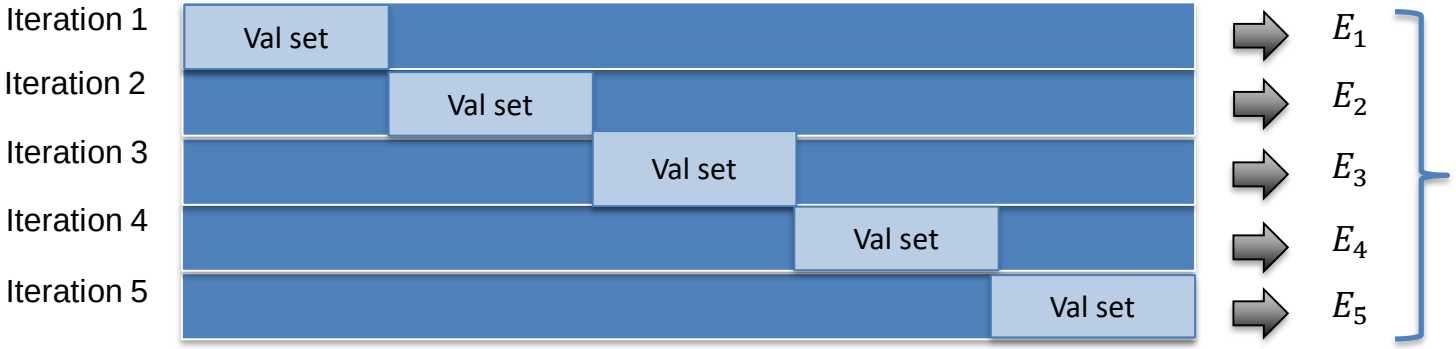
Training folds

Duke
PRATT SCHOOL of ENGINEERING

# K-Folds Cross Validation

We calculate the error on the validation fold for each iteration, and then average them together to get the average error

**K=5 Folds**




$$E = \frac{1}{5}\sum_{i=1}^{5} E_i$$

K Validation fold

Training folds

# Other types of cross-validation

- **ShuffleSplit cross-validation**
  - For each fold, we randomly select a % of observations to use for validation and use the rest for training
  - E.g. we can set K=4 and validation % = 10%. Each fold, we randomly select 10% of observations for validation
  - This can cut down on compute time, but the tradeoff is we may use observations multiple times, or not use some observations, introducing bias

- **Leave-one-out cross-validation**
  - Set K = number of points
  - Each observation becomes a validation set one time, and all other observations are used for training
  - Average your performance over all K folds

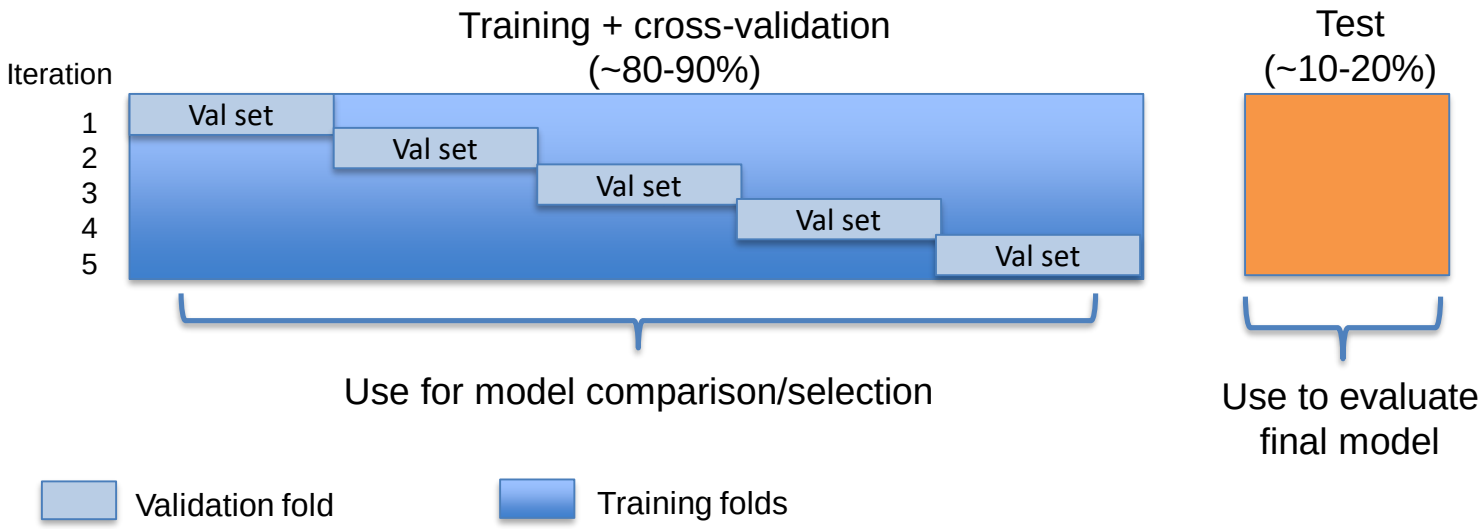
# Why use cross-validation?

- Maximizes the data available for training the model – important for small datasets
- Provides a better evaluation of how well the model can generalize to new data – validation performance is not accidentally biased by choice of datapoints to use for validation

Cross-validation is the typical approach used in industry for model selection and tuning, except in cases of very large datasets

# CV and Model Evaluation

Remember, when you complete your cross-validation and do your model selection, you still need to evaluate your model on the test set for a fair approximation of performance



# CV using Scikit-learn

- Divide data into "k" folds

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=10)
```

- For each model you want to compare:

```
for model in [model_a,model_b]:
```

  - For each of "k" iterations:

```
for (train_idx,val_idx) in kf.split(X=X_train,y=y_train):
```

    - Get the training folds and the validation fold for the iteration

```
X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]
```

    - Fit the model on the training data

```
model.fit(X_fold_train,y_fold_train)
```

    - Get predictions for the validation fold, calculate performance and store it

```
preds = model.predict(X_fold_val)
acc_val = sum(preds==y_fold_val)/len(y_fold_val)
```

  - Calculate mean validation performance across all iterations

```
mean_acc = np.mean(acc_folds)
```

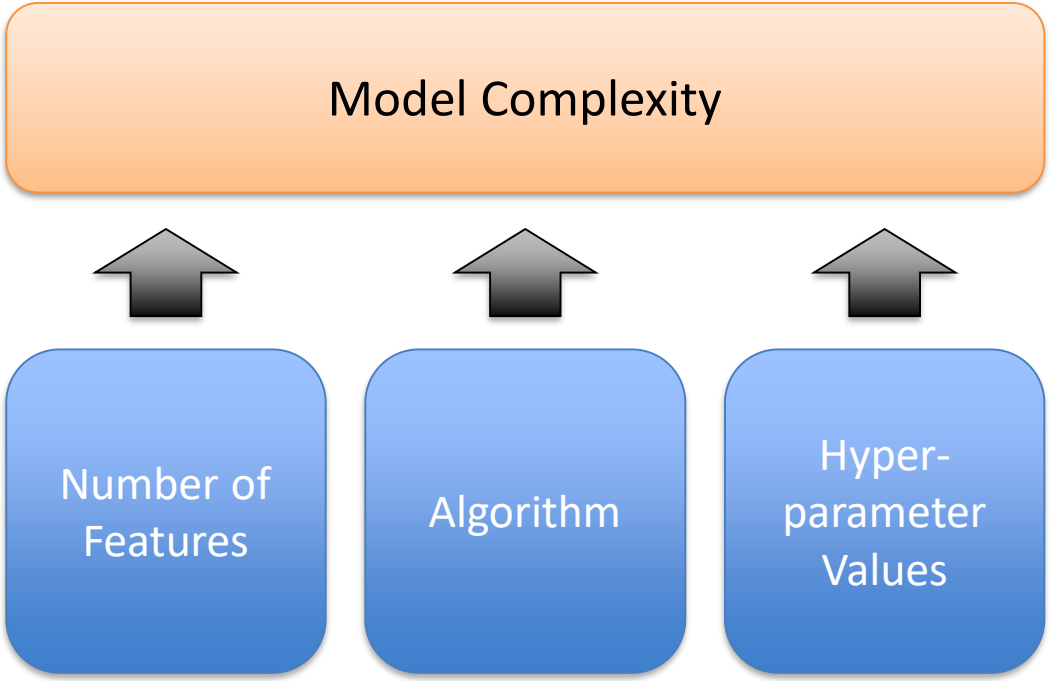
- Compare CV average performance and select model

outrageously
AMBITIOUS

# Model Selection: The Bias-Variance Tradeoff

Duke
PRATT SCHOOL of ENGINEERING

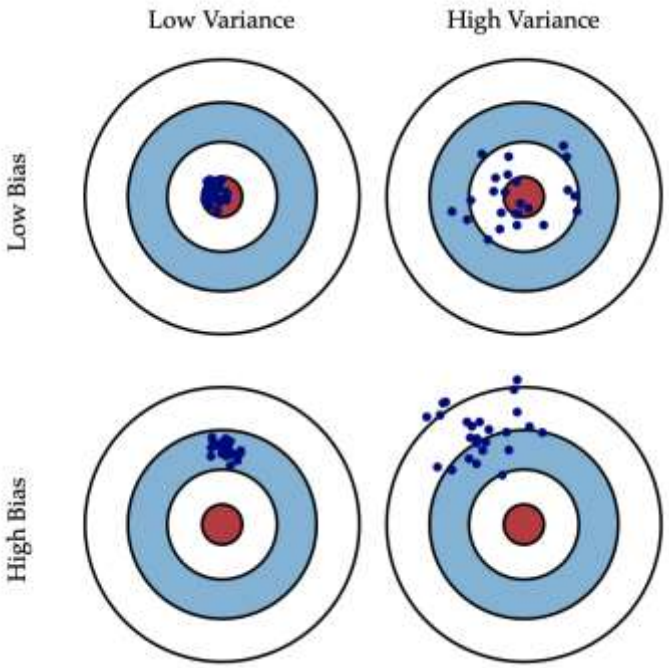
# Model Complexity



# Bias and Variance

- **Bias** is error from incorrect assumptions within the model
  - Often introduced by modeling a real-life problem using a simpler model that is unable to fully capture the underlying patterns in data

- **Variance** refers to the sensitivity of the model to fluctuations in the training data
  - Usually due to model fitting itself to fine patterns which may just be noise




*See: http://scott.fortmann-roe.com/docs/BiasVariance.html*

# Bias – Variance Tradeoff

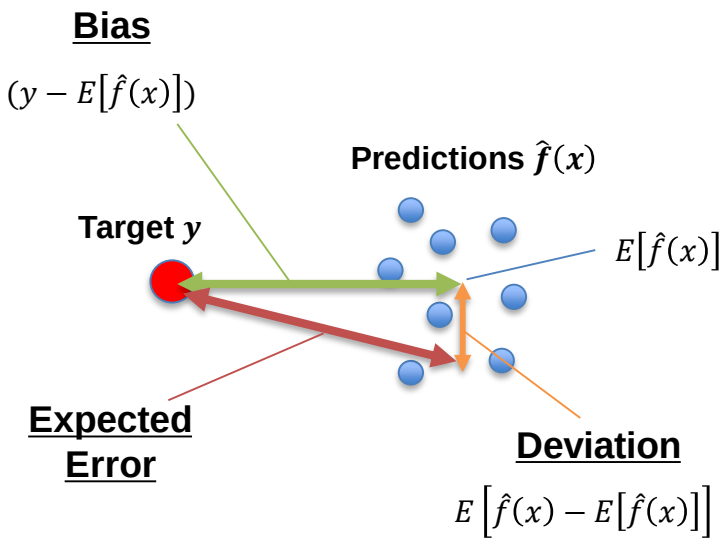
For a model y= $\hat{f}(x)$, at any point (*x*,*y*):

$$\textit{Expected Squared Error} = E\left[\left(y - \hat{f}(x)\right)^2\right]$$

$$= (y - E[\hat{f}(x)])^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma_e^2$$

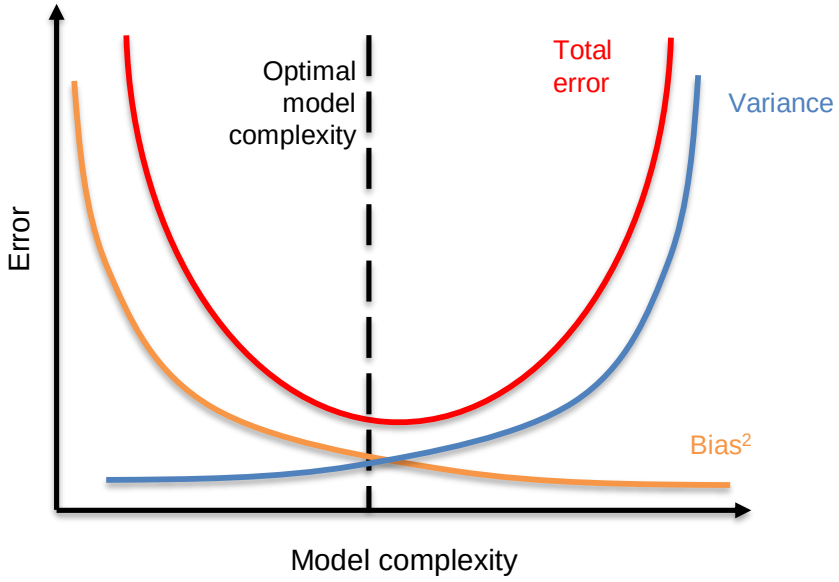
**Bias²** **Variance**

$$= Bias^2 + Var + \underbrace{\sigma_e^2}_{\text{Irreducible error}}$$

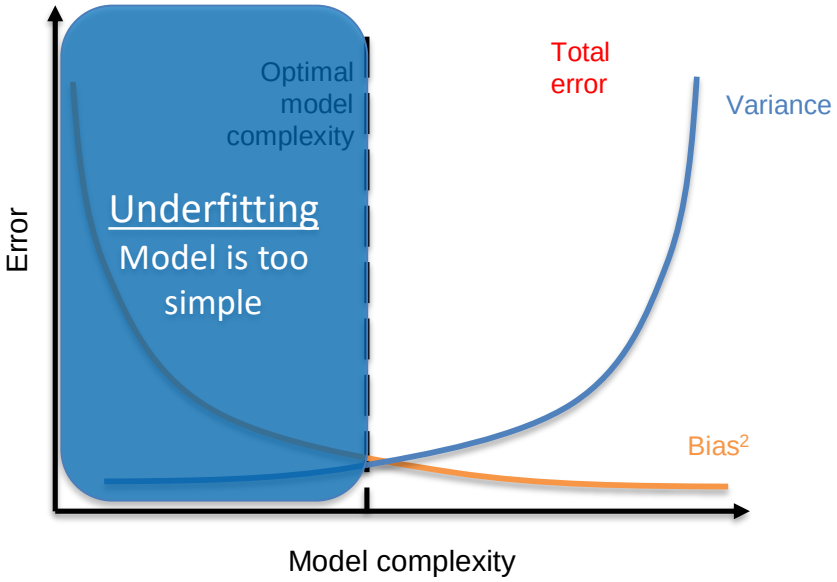


# Bias – Variance Tradeoff

- $\text{Total Error} = \text{Bias}^2 + \text{Var} + \sigma_e^2$
- Bias and variance move in opposite directions
- Simpler models often have **higher** bias and **lower** variance
- Complex models typically have **lower** bias but **higher** variance



# Underfitting vs. Overfitting



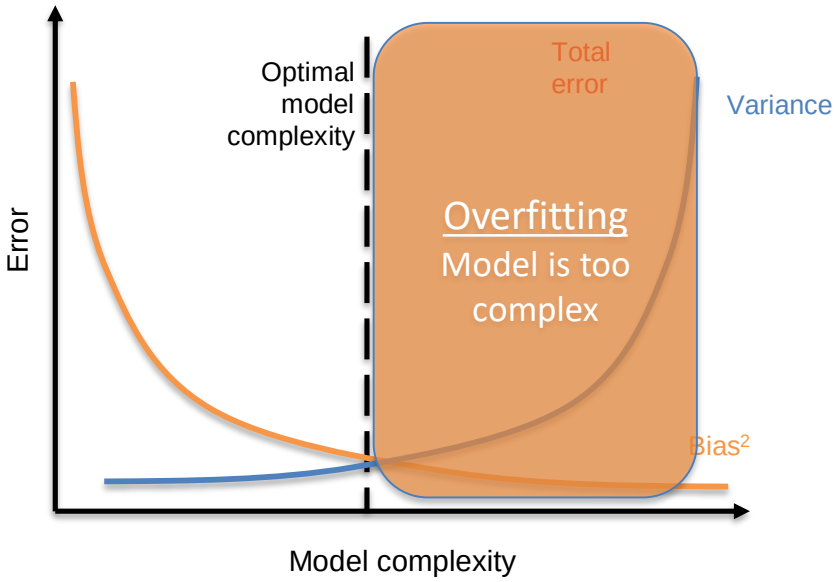




*Image source: http://scott.fortmann-roe.com/docs/BiasVariance.html*

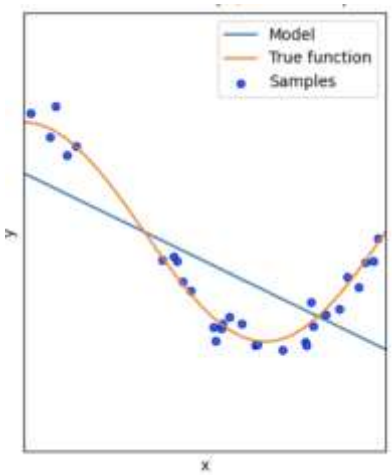
Duke
PRATT SCHOOL of ENGINEERING

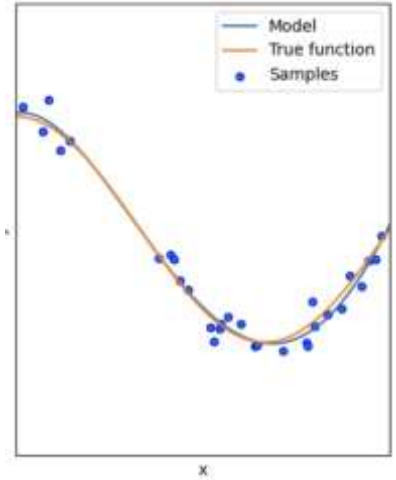
# Underfitting vs. Overfitting

**Underfitting**
Model is too simple

**Good Fit**
Model fits well, with some error

**Overfitting**
Model is too complex



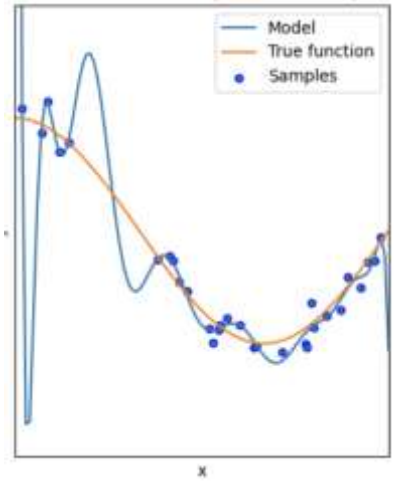




*Image source: Scikit Learn documentation*

# How do we recognize overfitting?

- As we increase complexity, training score continues to improve
- However, validation score improves to a point but then drops as model overfits the training data
- Eventually, training performance & validation performance diverge indicating model is overfit

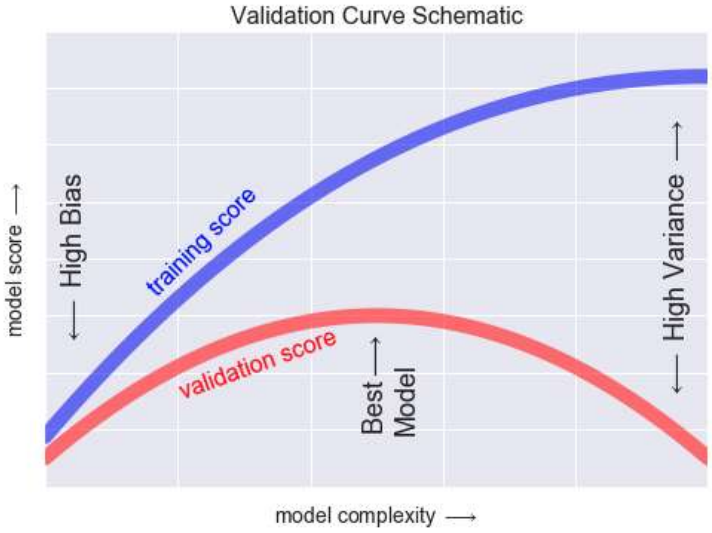



*Image source: Python Data Science Handbook*

# Finding the optimal model

- Our goal is to find the optimal model complexity to balance bias and variance, thereby minimizing error
- What influences complexity?
  - **Choice of algorithm**
  - **Number of features**
  - **Hyperparameter settings**
- Optimal complexity varies with amount of data – the more data, the more complex model you can use
- We find this using the validation approaches we discussed last class

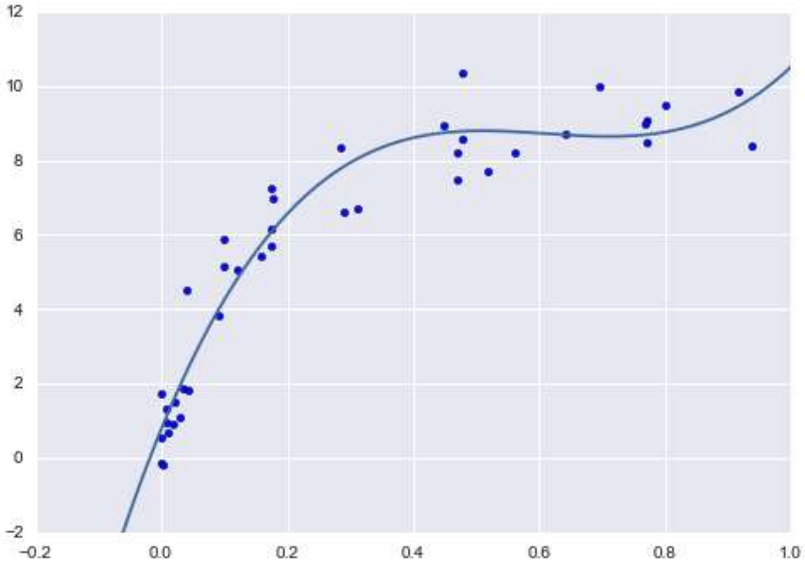



*Image source: Python Data Science Handbook*

Duke
PRATT SCHOOL of ENGINEERING

# DEMO: VALIDATION CURVES

outrageously
**AMBITIOUS**

# Metrics

Duke
PRATT SCHOOL of
ENGINEERING

# Outcomes vs. Outputs

**Outcome**

- Refers to the desired business impact on your organization or for your customer
- Stated in terms of the expected impact (which is often $)
- Does NOT contain model performance metrics or other technical metrics

**Output**

- Refers to the desired output from the model
- Measured in terms of a model performance metric
- Typically not communicated to the customer
- Set this AFTER setting the desired outcome

# Outcomes vs. Outputs

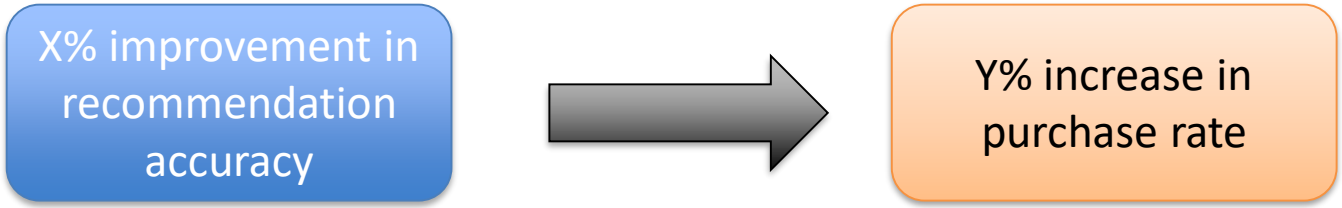
| | A tool to predict turbulence for airlines | A power demand forecasting tool for a utility |
|---|---|---|
| **Output** | • AUROC or precision/recall of turbulence prediction (binary or 1-5 scale) | • MSE (most likely) or MAE/MAPE |
| **Outcome** | • Low # of safety incidents per year, or lower $ of safety-related claims | • Lower cost per MWh of power produced<br>• Lower emissions rate per MWh |

# Thought Exercise

Suppose we work for an ecommerce company and have been asked to determine whether we should move from batch prediction to online prediction.

How would we go about determining whether we should do this?

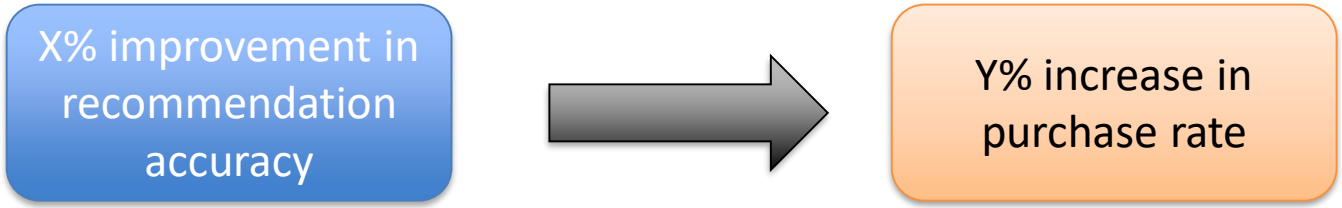
# Hypothesis




*Purchases = f(Accuracy)*

- How do we determine what % improvement in recommendation performance we can get?
- How do we determine what % increase in purchase rate we can expect?

# Hypothesis




*Purchases = f(Accuracy)*

- How do we determine what % improvement in recommendation performance we can get? A/B testing
- How do we determine what % increase in purchase rate we can expect? Historical data

# Baselines

- Metrics are meaningless without a baseline for comparison
- There are a few important baselines to determine:
  1. **Zero rule baseline** (Most common class / mean value)
  2. **Heuristic baseline**
  3. **Human baseline**
  4. **Existing solution baseline**

outrageously
**AMBITIOUS**

# Regression Error Metrics

Duke
PRATT SCHOOL of ENGINEERING

# MSE, MAE and MAPE

**Mean Squared Error**

$$MSE = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$$

- Most popular regression error metric
- Heavily influenced by outliers - penalizes large errors much more than small ones
- Influenced by scale of data
- Sometimes used as RMSE

**Mean Absolute Error**

$$MAE = \frac{1}{n}\sum_i |y_i - \hat{y}_i|$$

- Also influenced by scale
- More robust to outliers
- Can be easier to interpret in context of the problem
- Abs value is less desirable for calculations (squared terms are easier to minimize via differentiation)

**Mean Absolute % Error**

$$MAPE = \frac{1}{n}\sum_i \frac{|y_i - \hat{y}_i|}{y_i}$$

- Converts error to a percentage
- Popular among non-technical managers because it is easily understood
- Skewed by high % errors for low values of y

# Example: MAE vs. MSE/RMSE

**CASE 1: Evenly distributed errors**

| ID | Error | \|Error\| | Error^2 |
|---|---|---|---|
| 1 | 2 | 2 | 4 |
| 2 | 2 | 2 | 4 |
| 3 | 2 | 2 | 4 |
| 4 | 2 | 2 | 4 |
| 5 | 2 | 2 | 4 |
| 6 | 2 | 2 | 4 |
| 7 | 2 | 2 | 4 |
| 8 | 2 | 2 | 4 |
| 9 | 2 | 2 | 4 |
| 10 | 2 | 2 | 4 |

| MAE | RMSE |
|---|---|
| 2.000 | 2.000 |

**CASE 2: Small variance in errors**

| ID | Error | \|Error\| | Error^2 |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 |
| 5 | 1 | 1 | 1 |
| 6 | 3 | 3 | 9 |
| 7 | 3 | 3 | 9 |
| 8 | 3 | 3 | 9 |
| 9 | 3 | 3 | 9 |
| 10 | 3 | 3 | 9 |

| MAE | RMSE |
|---|---|
| 2.000 | 2.236 |

**CASE 3: Large error outlier**

| ID | Error | \|Error\| | Error^2 |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 |
| 10 | 20 | 20 | 400 |

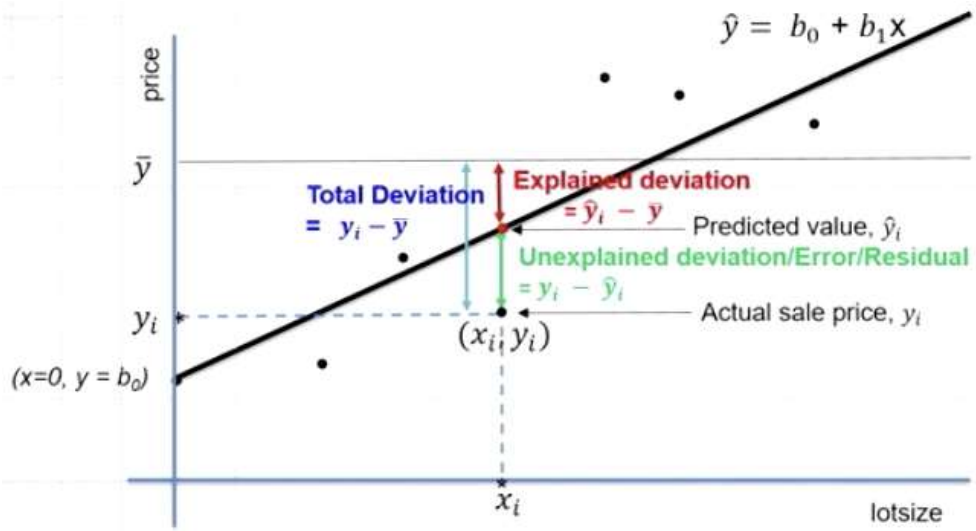
| MAE | RMSE |
|---|---|
| 2.000 | 6.325 |

- MSE/RMSE penalizes severe errors much more than MAE
- This can be desirable if being off by a lot one time is much worse than being off by a little every time

Image source: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

# Coefficient of determination ($R^2$)

- Deviation of points from mean can be broken down into two components:
  - ”Explained deviation” – portion of deviation predicted by your model
  - “Unexplained deviation” – difference between prediction and actual point



# Coefficient of determination ($R^2$)

| | | | | |
|---|---|---|---|---|
| $\sum_i (y_i - \bar{y})^2$ | = | $\sum_i (y_i - \hat{y}_i)^2$ | + | $\sum_i (\bar{y} - \hat{y}_i)^2$ |
| SST | = | SSE | + | SSR |
| Total Sum of Squares | = | Sum of Squared Errors | + | Sum of Squares Regression |
| Total deviation from mean | | Unexplained deviation (error) | | Deviation explained by model |

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

$R^2$ ranges between 0 and 1

R-squared is typically used to communicate how well your feature variables (X) explain the variability in your target variable (y)

Duke
PRATT SCHOOL of ENGINEERING

# Adjusted $R^2$

- R-squared is often viewed as a measure of **goodness of fit** rather than prediction ability -> calculated "in-sample" on the training data itself
- As we add more features, the R-squared almost always goes up because the model finds additional patterns of correlation (which might be noise)
- However, this does not mean the model can generate better predictions on new data -> remember our discussion on overfitting
- To help with this problem, we often report "adjusted R-squared", which penalizes for more features (added complexity)

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(n - 1)}{(n - p - 1)}$$

"Is R-squared useless?": https://data.library.virginia.edu/is-r-squared-useless/

# How to calculate these metrics?

- Calculate them manually or use sklearn.metrics
- Remember: calculate these on the **validation and/or test sets** (depending on whether you are comparing models or evaluating a model)

| Metric | sklearn function |
|---|---|
| Mean squared error (MSE) | mean_squared_error(y_true,y_pred) |
| Mean absolute error (MAE) | mean_absolute_error(y_true,y_pred) |
| Mean absolute % error (MAPE) | [NEW in 0.24] mean_absolute_percentage_error(y_true,y_pred) |
| Coef. of determination ($R^2$) | r2_score(y_true,y_pred) |

# EXERCISE: REGRESSION METRICS

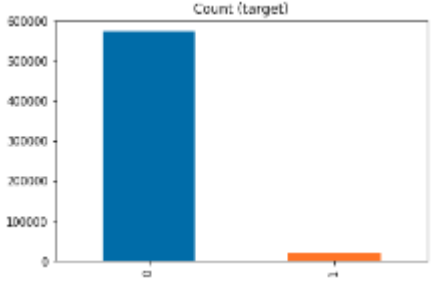
outrageously
**AMBITIOUS**

# Classification Error Metrics

Duke
PRATT SCHOOL of ENGINEERING

# Accuracy

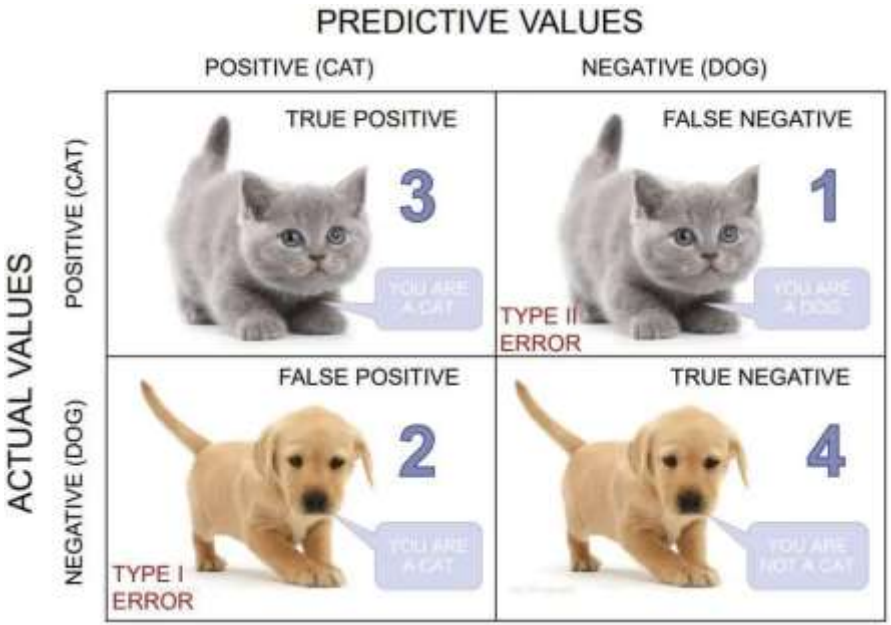
- Accuracy is the most popular and easiest to understand error metric
- However, accuracy can be deceiving when there is a class imbalance
- Consider this situation:
  - I am building a model to predict whether patients will have a certain rare disease
  - I gather a dataset with thousands of patients and several features, along with a label of whether they were diagnosed with the rare disease (“1”) or not (“0”)
  - Using this dataset, I create a classifier with 99.4% accuracy!
- What’s the problem?
  - My dataset had very high <u>class imbalance</u>
  - My model just predicted “0” for every patient
  - And it was right 99.4% of the time!




Duke
PRATT SCHOOL of ENGINEERING

# Confusion Matrix – Binary Classification

| True class, $y$ \ Predicted class, $\hat{y}$ | 1 (Positive) | 0 (Negative) |
|---|---|---|
| 1 (Positive) | True positives | False negatives |
| 0 (Negative) | False positives | True negatives |



# Confusion Matrix – Binary Classification

**Predicted class, $\hat{y}$**

| **True class, $y$** | 1 (Positive) | 0 (Negative) |
|---|---|---|
| 1 (Positive) | True positives | False negatives |
| 0 (Negative) | False positives | True negatives |

**True Positive Rate (TPR)**
**Recall**
**Sensitivity**

How many of all positives did the model correctly classify as positives?

$$\frac{\text{True positives}}{\text{All positives}} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

# Confusion Matrix – Binary Classification

| True class, $y$ \ Predicted class, $\hat{y}$ | 1 (Positive) | 0 (Negative) |
|---|---|---|
| 1 (Positive) | True positives | False negatives |
| 0 (Negative) | False positives | True negatives |

## False Positive Rate (FPR)

How many of all negatives did the model incorrectly classify as positives?

$$\frac{\text{False positives}}{\text{All negatives}} = \frac{\text{False positives}}{\text{False positives} + \text{True negatives}}$$

# Confusion Matrix – Binary Classification

| True class, $y$ \ Predicted class, $\hat{y}$ | 1 (Positive) | 0 (Negative) |
|---|---|---|
| 1 (Positive) | True positives | False negatives |
| 0 (Negative) | False positives | True negatives |

## Precision

How many of the predicted positives are actually positives?

$$\frac{\text{True positives}}{\text{All predicted positives}} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

# Confusion matrix cheat sheet

| | | True condition | | | | |
|---|---|---|---|---|---|---|
| | Total population | Condition positive | Condition negative | Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$ | Accuracy (ACC) $= \frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$ | |
| Predicted condition | Predicted condition positive | **True positive** | **False positive**, Type I error | Positive predictive value (PPV), Precision $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$ | False discovery rate (FDR) $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$ | |
| | Predicted condition negative | **False negative**, Type II error | **True negative** | False omission rate (FOR) $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$ | Negative predictive value (NPV) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$ | |
| | | True positive rate (TPR), Recall, Sensitivity, probability of detection, Power $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$ | False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$ | Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$ | Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR−}}$ | $F_1$ score $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$ |
| | | False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$ | Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$ | Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$ | | |

*Source: Wikipedia*

# Question

- Our model predicts every observation as “positive”
  - Is the **recall / true positive rate** high or low?
  - What about the **false positive rate**?
  - How about the **precision**?

# ROC Curves

- A Receiver Operating Characteristic (ROC) curve plots the True Positive Rate (TPR) and False Positive Rate (FPR) for different threshold values
- What is a **threshold**?
  - Most classification models return the probability of the positive class
  - The data scientist sets a threshold for the positive class – if the probability returned by the model is above threshold, the assigned label is positive
  - Typically the threshold is set to 0.5 by default

Classifier decision rule:

$$\hat{y} = \begin{cases} 1, & x > thresh \\ 0, & x \leq thresh \end{cases}$$

What happens to TPR if we increase the threshold to 0.9?
What about FPR?
What if we go the other way to 0.1?

# ROC Curves

**To build a ROC curve:**

- Run the model and get the output probabilities
- For each value in range(0,1):
  - Set value as threshold value
  - Calculate predictions by comparing model output probabilities to threshold
  - Compare the predictions to targets and calculate the TPR and FPR values
- Plot the values for all thresholds on a graph of TPR vs FPR

| | Model Output | Actual Target | Thresh = 0.3 | Thresh = 0.5 | Thresh = 0.7 |
|---|---|---|---|---|---|
| 1 | 0.85 | 1 | 1 | 1 | 1 |
| 2 | 0.04 | 0 | 0 | 0 | 0 |
| 3 | 0.62 | 1 | 1 | 1 | 0 |
| 4 | 0.37 | 0 | 1 | 0 | 0 |
| 5 | 0.55 | 0 | 1 | 1 | 0 |
| TPR | | | 2/2 | 2/2 | 1/2 |
| FPR | | | 2/3 | 1/3 | 0/3 |

# ROC Curves

**To build a ROC curve:**

- Run the model and get the output probabilities
- For each value in range(0,1):
  - Set value as threshold value
  - Generate predictions by comparing model output probabilities to threshold
  - Compare the predictions to targets and calculate the TPR and FPR values
- Plot the values for all thresholds on a graph of TPR vs FPR

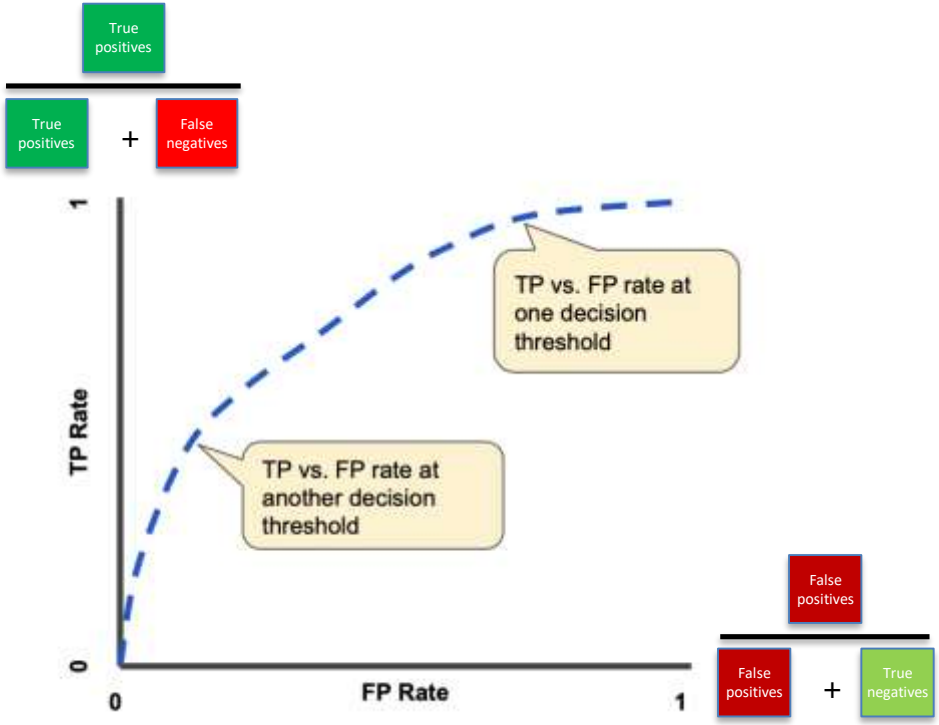
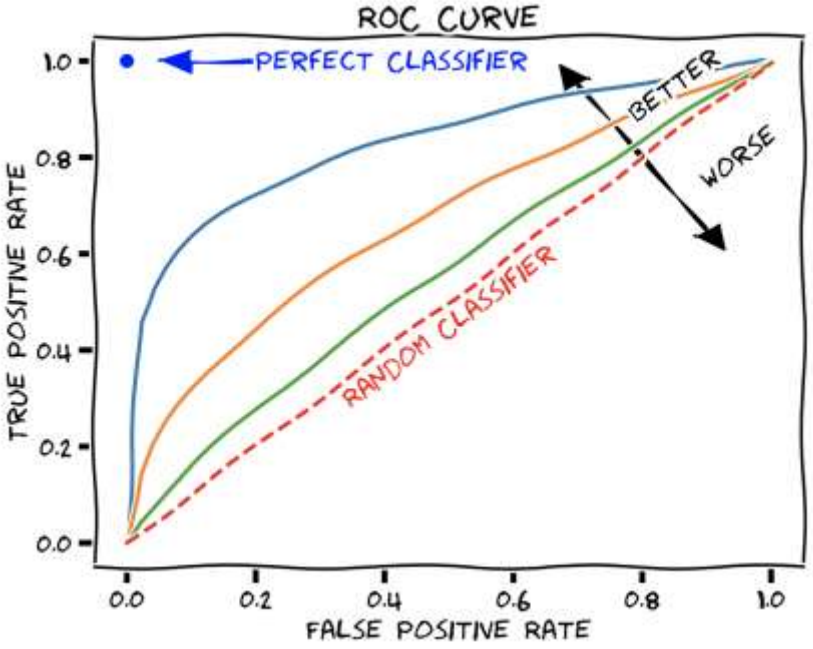



*Image source: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc*

Duke
PRATT SCHOOL of ENGINEERING

# Area Under ROC (AUROC)

- A common error metric for classification models is AUROC
- This represents the area under the ROC curve
- A perfect classifier will have have AUROC=1
- A random classifier will have AUROC=0.5
  - Why? See: https://datascience.stackexchange.com/questions/31872/auc-roc-of-a-random-classifier
- Therefore, a model should have an AUROC between 0.5 and 1 (higher is better)

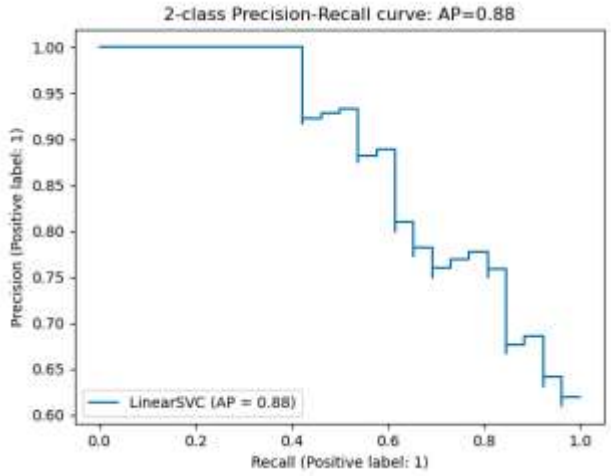



*https://en.wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:Roc-draft-xkcd-style.svg*

# Precision-Recall Curve

- Another evaluation technique is the precision-recall (PR) curve
- This measures the tradeoff between recall and precision as the model threshold is varied (just like we did for the ROC)
- PR curves are especially useful if we have high class imbalance (e.g. a lot of 0's and only a few 1's)
  - Unlike ROC curves, they do not factor in True Negatives

$$\frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$



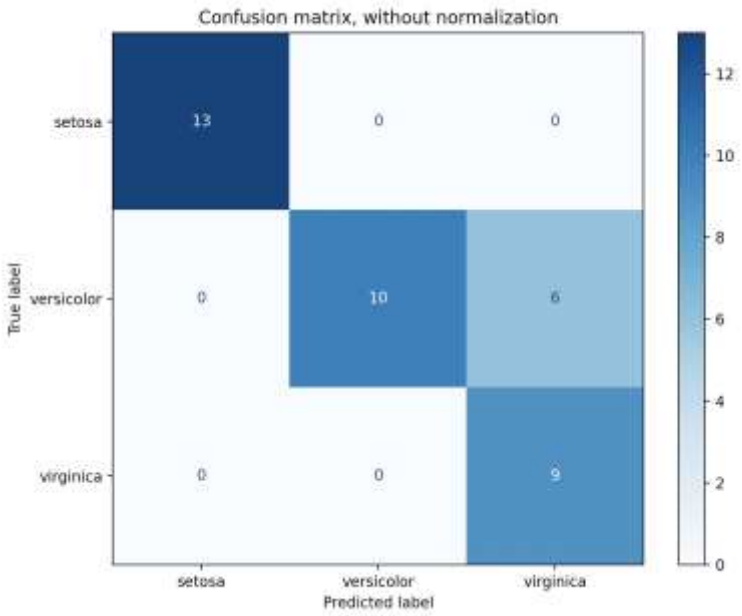

$$\frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

*Image source: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#sphx-glr-auto-examples-model-selection-plot-precision-recall-py*

Duke
PRATT SCHOOL of ENGINEERING

# Multiclass Confusion Matrix

- The multiclass confusion matrix shows us where the classifier model is struggling to differentiate between classes
- A perfect classifier would have values only on the diagonals
- To plot the confusion matrix we use scikit learn's plot_confusion_matrix() method




Duke
PRATT SCHOOL of ENGINEERING

# Multiclass Confusion Matrix

- We can calculate the same metrics as we did before, but now we do it <u>for each class</u>
- Scikit learn will do this for us using the classification_report() method
- We can also report overall "macro-averaged" precision and recall using a simple average across all classes

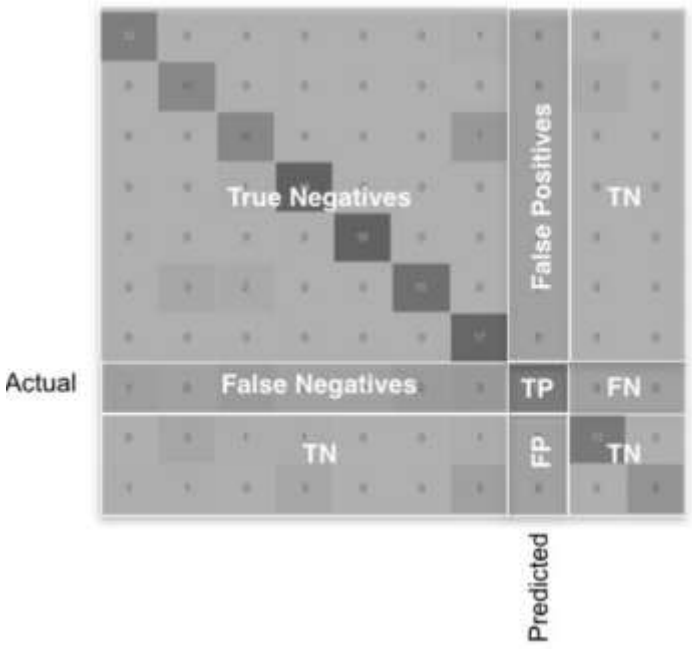



*Image source: https://towardsdatascience.com/multi-class-classification-extracting-performance-metrics-from-the-confusion-matrix-b379b427a872*

# $F_1$ Score

- Precision or recall alone do not give us the whole picture of how well our classifier is doing, particularly for data with high class imbalance
  - We can use the $F_1$ score, which takes both precision and recall into account

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

- This is common in academic papers but needs to be used carefully, because it gives equal weight to precision and recall
  - E.g. classifying a healthy person as sick can have very different consequences as classifying a sick person as healthy

# DEMO: CLASSIFICATION METRICS

# Wrap-Up: Metric Selection

- Selecting proper metrics is a key activity of Step 1 of the CRISP-DM process: Business Understanding
- What we have discussed today is "output metrics", but it is even more important to consider "outcome metrics"
- Your choice of metric should reflect the nature of your problem and the consequences of being wrong
  - E.g. for a regression problem, is it much worse to be very wrong a few times, or a little wrong a lot of times?
  - Or, for a classification problem, are false positives or false negatives worse?