Designing and Analysis of ML Experiments

Suman Kumar

Department of computer Science

Troy University

References “Introduction to Machine Learning, 3rd edition Ethen Alpaydin, *Machine learning with Scikit-Learn & tensor flow by Aurelien Geron & various other sources*

Contents

- Machine Learning Project: Big Picture
- Data: Feature Scaling
- Algorithms: Sampling, Validation and Evaluation
- Performance: Measuring Errors
- Data: Data Issues
- Algorithms: Bias and Variance

Contents

- **Machine Learning Project: Big Picture**
- Data: Feature Scaling
- Algorithms: Sampling, Validation and Evaluation
- Performance: Measuring Errors
- Data: Data Issues
- Algorithms: Bias and Variance

Steps of Machine Learning Project

- Look at the big picture
- Get the data
- Discover and visualize the data (to gain insight)
- Prepare the data for machine learning algorithms
- Select a model and train it
  - Explore many models and short-list best ones
- Fine-Tune your model
  - Combine them into great solution
- Present your solution

Effectiveness of Data and Models

- *"We may want to reconsider the trade-off between spending time and money on algorithm development versus spending on corpus development"*[1]

- No Free Lunch Theorem: *If you make absolutely no assumption about the data, there is no reason to prefer one model over any other*[2]

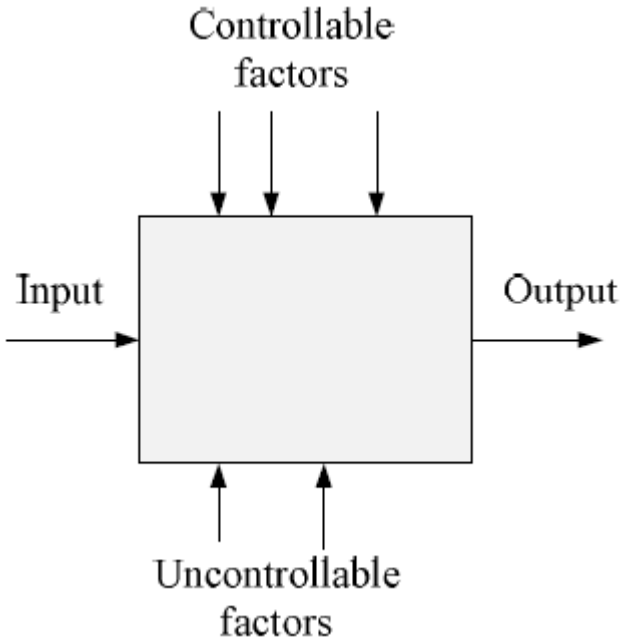
[1]F. Pereira, P. Norvig and A. Halevy, "The Unreasonable Effectiveness of Data," in IEEE Intelligent Systems, vol. 24, no. , pp. 8-12, 2009

[2]Wolpert, David , "The Lack of A Priori Distinctions between Learning Algorithms", Neural Computation, 1996

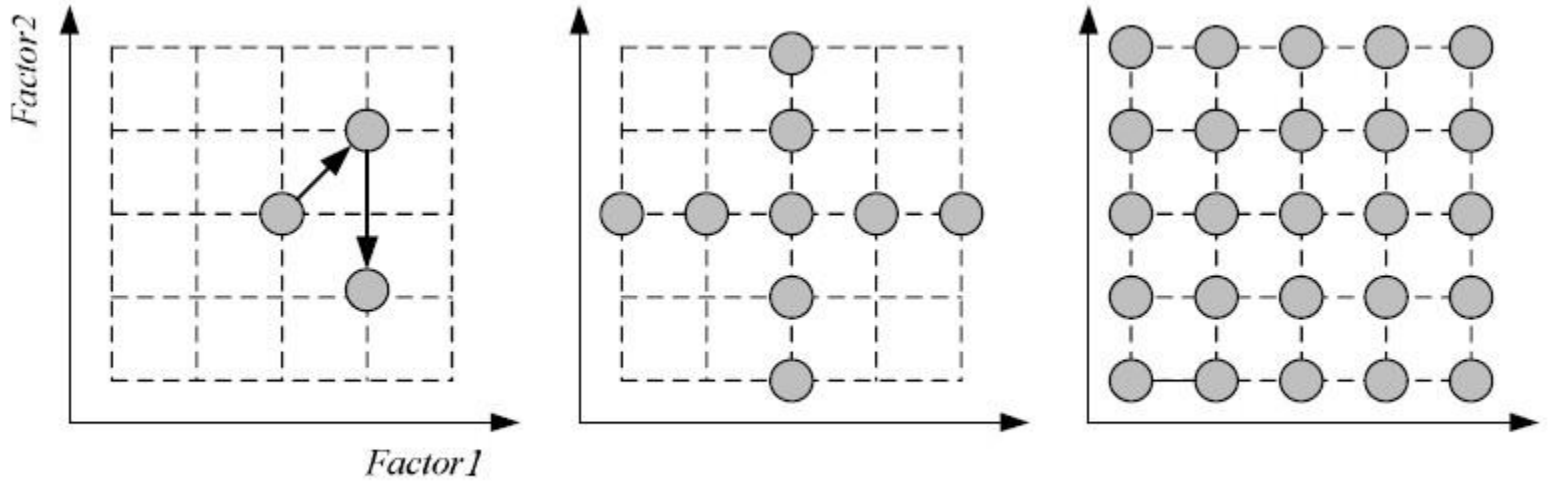
Algorithm Preference

- Criteria (Application-dependent):
  - Misclassification error, or risk (loss functions)
  - Training time/space complexity
  - Testing time/space complexity
  - Interpretability
  - Easy programmability
- Cost-sensitive learning

Factors and Response



Strategies of Experimentation



(a) Best guess (b) One factor at a time (c) Factorial design

Response surface design for approximating and maximizing the response function in terms of the controllable factors

- **Programming Notes:**

```
from pyDOE2 import ccdesign, fullfact
design = fullfact(levels)
ccd = ccdesign(2, center=(4, 4))
df_ccd = pd.DataFrame(ccd, columns=["X1", "X2"])
df_ccd['Y'] = 3 * df_ccd['X1']**2 + 2 * df_ccd['X2'] +
np.random.normal(0, 1, len(df_ccd))
model_ccd = ols("Y ~ X1 + X2 + I(X1**2) + I(X2**2) + X1:X2",
data=df_ccd).fit()
```

Guidelines for ML experiments

A. Aim of the study
B. Selection of the response variable
C. Choice of factors and levels
D. Choice of experimental design
E. Performing the experiment
F. Statistical Analysis of the Data
G. Conclusions and Recommendations

Contents

- Machine Learning Project: Big Picture
- **Data: Feature Scaling**
- Algorithms: Sampling, Validation and Evaluation
- Performance: Measuring Errors
- Data: Data Issues
- Algorithms: Bias and Variance

Feature Scaling

- ML algorithms may not perform well for different features scales.
- Methods
  - Min Max (bring values the range of 0 and 1): $X_{\text{scaled}} = \dfrac{X - X_{\min}}{X_{\max} - X_{\min}}$
  - Z-Score Normalization: $X_{\text{scaled}} = \dfrac{X - \mu}{\sigma}$
  - Max Abs Scaling: $X_{\text{scaled}} = \dfrac{X}{|X_{\max}|}$
  - Robust Scaling: $X_{\text{scaled}} = \dfrac{X - \text{median}(X)}{\text{IQR}}$
    - Interquartile Range (IQR) = 40 - 20 = 20
  - Log Transformation: $X_{\text{scaled}} = \log(X)$
  - Power Transformation (Yeo-Johnson)
    - Aims to stabilize variance and make the data more Gaussian-like.
  - Quantile Transformation
    - Transforms the features to follow a uniform or normal distribution using quantiles

Feature Scaling: Example Calculation

	Input	Min-Max Scaling (0-1)	Z-Score Normalization	Max Abs Scaling	Robust Scaling	Power Transformation (Y	Quantile Transformation	Log Transformation
1	0	0.0	-1.4638501094227998	0.0	-1.0	-1.567836588379048	0.0	0.0
2	1	0.2	-0.8783100656536799	0.2	-0.6	-0.836193515175872	0.2	0.6931471805599453
3	2	0.4	-0.29277002188455997	0.4	-0.2	-0.2100530624428889	0.4	1.0986122886681096
4	3	0.6000000000000001	0.29277002188455997	0.6	0.2	0.3561112817932915	0.6	1.3862943611198906
5	4	0.8	0.8783100656536799	0.8	0.6	0.881485650863418	0.8	1.6094379124341003
6	5	1.0	1.4638501094227998	1.0	1.0	1.3764862333410988	1.0	1.791759469228055

Algorithms Where Feature Scaling Matters

Algorithm	Why It Matters
K-Nearest Neighbors (KNN)	Distance-based algorithm (e.g., Euclidean). Larger scale features dominate distance calculations.
Support Vector Machines (SVM)	Uses dot products and distance metrics; unscaled features distort the margin.
K-Means Clustering	Also distance-based.
Principal Component Analysis (PCA)	Based on variance; unscaled data leads to biased principal components toward higher-variance features.
Gradient Descent-based Algorithms (e.g., Linear Regression, Logistic Regression, Neural Networks)	Scaling helps faster and more stable convergence; unscaled features slow or distort learning.
Naive Bayes with Gaussian Assumption	The Gaussian formula includes standard deviation; scaling changes the likelihood calculation.
L1/L2 Regularized Models (e.g., Ridge, Lasso, ElasticNet)	Regularization penalizes large coefficients, and unscaled features can bias the penalty toward some features.
Neural Networks	Scaling helps weights converge evenly across features and prevents one feature from dominating the activations. [6]

Algorithms Where Feature Scaling Does *Not* Matter Much

Algorithm	Why it does not matter
Decision Trees (e.g., CART, Random Forests)	Based on splits and thresholds, not distances or variances.
Gradient Boosting Trees (e.g., XGBoost, LightGBM)	Similar to decision trees; inherently scale-invariant.
Rule-based Models	Don’t use numeric optimization or distances.

Feature Scaling: Programming Notes


```
from sklearn.preprocessing import ( MinMaxScaler, StandardScaler,
MaxAbsScaler, RobustScaler,  PowerTransformer, QuantileTransformer)
import numpy as np

# X = data set
min_max_scaler = MinMaxScaler()
X_min_max = min_max_scaler.fit_transform(X)

standard_scaler = StandardScaler()
X_standardized = standard_scaler.fit_transform(X)

max_abs_scaler = MaxAbsScaler()
X_max_abs = max_abs_scaler.fit_transform(X)

robust_scaler = RobustScaler()
X_robust = robust_scaler.fit_transform(X)

X_log = np.log1p(X)

yeo_johnson_scaler = PowerTransformer(method='yeo-johnson')
X_yeo_johnson = yeo_johnson_scaler.fit_transform(X)

quantile_scaler = QuantileTransformer(output_distribution='uniform',
random_state=0)
X_quantile = quantile_scaler.fit_transform(X)
```

Contents

- Machine Learning Project: Big Picture
- Data: Feature Scaling
- **Algorithms: Sampling, Validation and Evaluation**
- Performance: Measuring Errors
- Data: Data Issues
- Algorithms: Bias and Variance

Testing and validation

- Training set: train the model using training set
- Test set: test your model on test set
  - Selection: complete random vs stratified sampling
- Error rate: generalization error
- Cross validation: split the training set into smaller training set and a validation test

Practice: 80% training and 20% test

Resampling

- Repeatedly drawing samples from a dataset and refitting a model ***to evaluate the model's performance***

- Useful when
  - The dataset is small, Assessment of model stability and generalizability needed, and Overfitting is a concern

- Methods
  - Hold-Out (Train/Test Split) (e.g., 80%–20%)
  - Cross-Validation
    - Partition data into several subsets (folds) and systematically train/test on them
    - Reduces variance and gives a better estimate of model performance
  - Bootstrap Sampling
    - Samples the data with replacement to create multiple training datasets
    - Used to estimate the confidence interval of a metric
  - Leave-One-Out Cross-Validation (LOOCV)
    - A special case of K-Fold where K = number of

K-Fold Cross-Validation

- Steps:
  - Dataset is split into K equal-sized folds.
  - The model is trained K times, each time leaving out one fold for validation and using the rest for training.
- The need for multiple training/validation sets $\{X_i, V_i\}_i$: Training/validation sets of fold *i*
- *K*-fold cross-validation: Divide X into *k*, $X_i, i=1,\ldots,K$

$$\mathcal{V}_1 = \mathcal{X}_1 \quad \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \Lambda \cup \mathcal{X}_K$$
$$\mathcal{V}_2 = \mathcal{X}_2 \quad \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \Lambda \cup \mathcal{X}_K$$
$$\mathrm{M}$$
$$\mathcal{V}_K = \mathcal{X}_K \quad \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \Lambda \cup \mathcal{X}_{K-1}$$

  - T_i share *K*-2 parts

- Output: Average everyfold results to get the final evaluation score

5×2 Cross-Validation (Dietterich, 1998)

- Used to compare two learning algorithms with better control of variance and correlation between train-test splits.
- 5 rounds of 2-fold cross-validation
- Compared to k-fold CV, it exhibits
  - Low bias & low variance due to random splits
  - independent performance estimates.

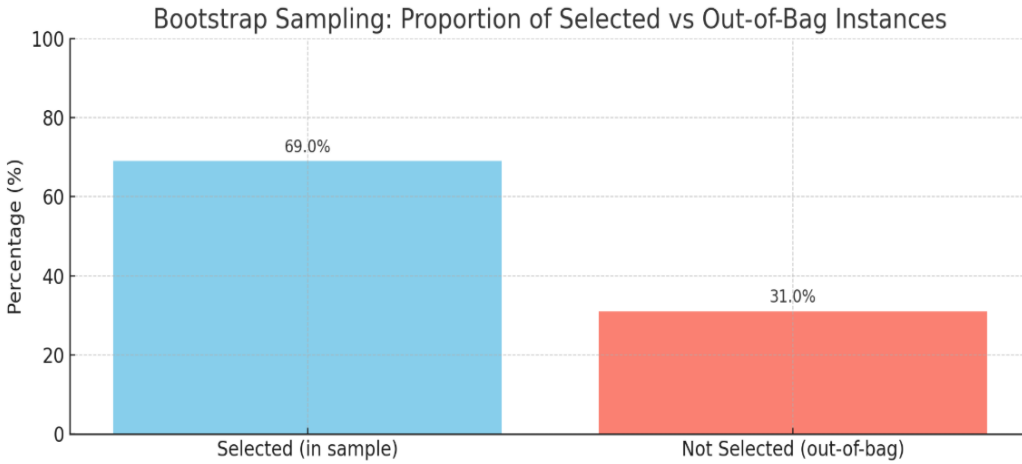
$$\mathcal{T}_1 = \mathcal{X}_1^{(1)} \quad \mathcal{V}_1 = \mathcal{X}_1^{(2)}$$
$$\mathcal{T}_2 = \mathcal{X}_1^{(2)} \quad \mathcal{V}_2 = \mathcal{X}_1^{(1)}$$
$$\mathcal{T}_3 = \mathcal{X}_2^{(1)} \quad \mathcal{V}_3 = \mathcal{X}_2^{(2)}$$
$$\mathcal{T}_4 = \mathcal{X}_2^{(2)} \quad \mathcal{V}_4 = \mathcal{X}_2^{(1)}$$
$$\vdots$$
$$\mathcal{T}_9 = \mathcal{X}_5^{(1)} \quad \mathcal{V}_9 = \mathcal{X}_5^{(2)}$$
$$\mathcal{T}_{10} = \mathcal{X}_5^{(2)} \quad \mathcal{V}_{10} = \mathcal{X}_5^{(1)}$$

Bootstrapping

- Used to generate multiple samples from a single sample.
- Draw instances from a dataset *with replacement*
- Prob that we do not pick an instance after N draws

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

that is, only 36.8% is new!



- When to use it:
  - The dataset is small
  - We want to estimate uncertainty (confidence intervals, model stability)
  - Model evaluation: Especially when cross-validation is computationally expensive.

Sampling & Evaluation : Programming Notes

- 80-20 Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, shuffle=True)
```

- K-Fold cross validation

```
from sklearn.model_selection import KFold, cross_val_score
kf = KFold(n_splits=5, shuffle=True, random_state=1)
scores = cross_val_score(model, X, y, cv=kf)
```

- 5x2 cross validation

```
from sklearn.model_selection import ShuffleSplit
# X = data set
rs = ShuffleSplit(n_splits=5, test_size=0.5, random_state=42)

for train_idx, test_idx in rs.split(X):
    # First run
    model.fit(X[train_idx], y[train_idx])
    scores.append(accuracy_score(y[test_idx], model.predict(X[test_idx])))

    # Swap train and test
    model.fit(X[test_idx], y[test_idx])
    scores.append(accuracy_score(y[train_idx], model.predict(X[train_idx])))
```

- Sample with replacement:

```
sample = np.random.choice(data, size=n_size, replace=True)
```

Contents

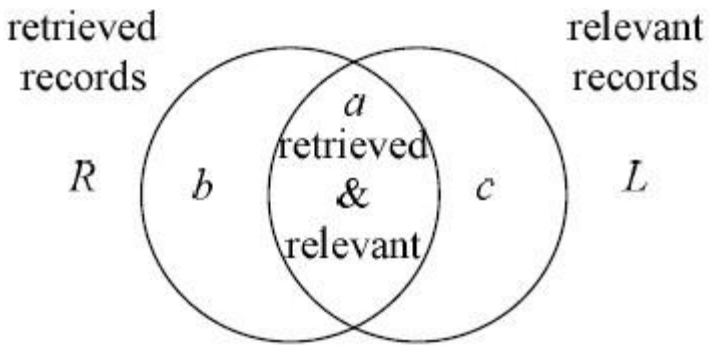
- Machine Learning Project: Big Picture
- Data: Feature Scaling
- Algorithms: Sampling, Validation and Evaluation
- **Performance: Measuring Errors**
- Data: Data Issues
- Algorithms: Bias and Variance

Measuring Error: Classification

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

- Error rate = # of errors / # of instances = (FN+FP) / N
- Recall = # of found positives / # of positives = TP / (TP+FN)
  = sensitivity = hit rate = True positive rate
- Precision = # of found positives / # of found = TP / (TP+FP)
- Specificity = TN / (TN+FP)
- False alarm rate = FP / (FP+TN) = 1 - Specificity

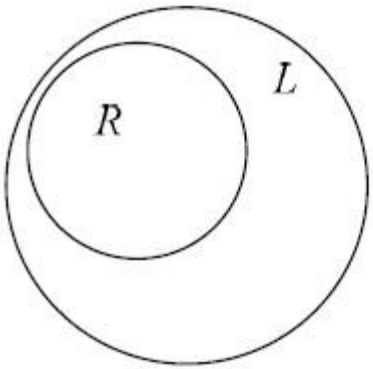
Precision and Recall: Database Example



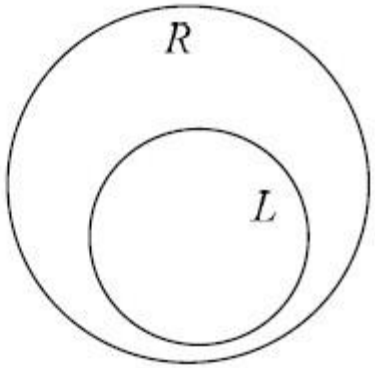
$$\text{Precision: } \frac{a}{a + b}$$

$$\text{Recall: } \frac{a}{a + c}$$

(a) Precision and recall



(b) Precision = 1



(c) Recall = 1

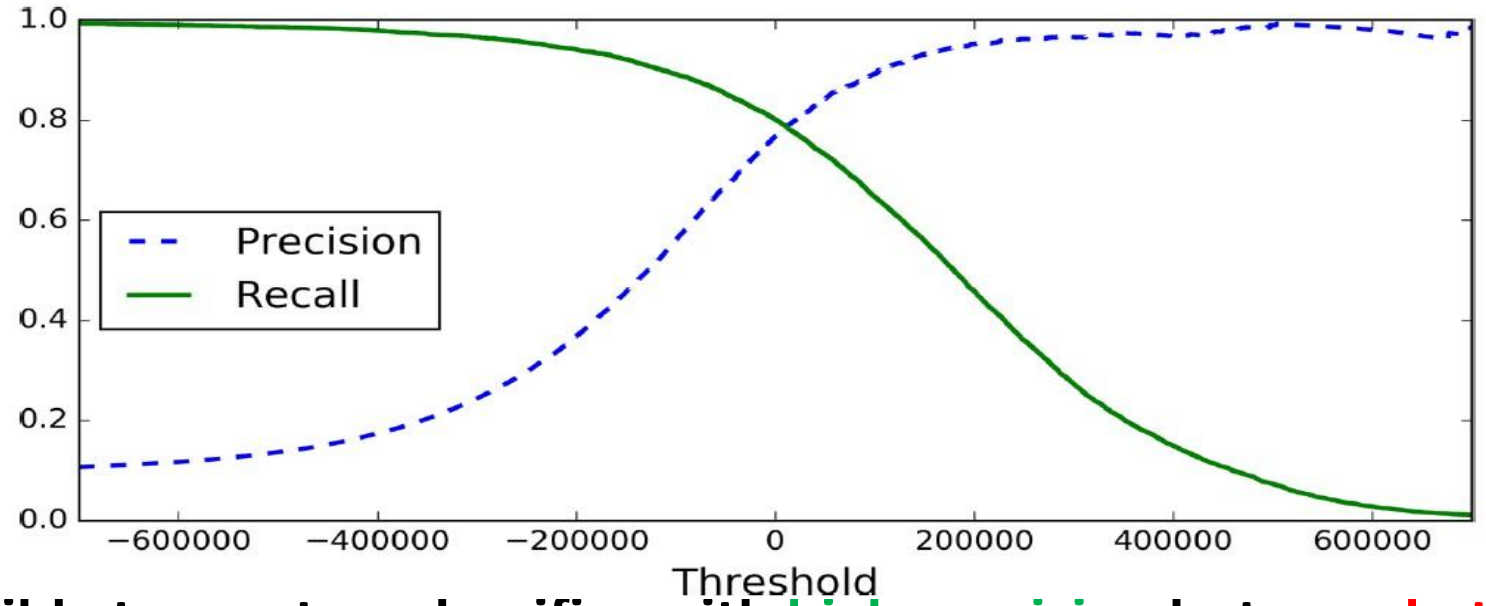
F_1 Score: Combine Precision & Recall

- $$\frac{1}{F_1} = \frac{\frac{1}{Precision} + \frac{1}{Recall}}{2}$$

- Classifier will get higher score only when precision and recall both are high.

Precision, Recall, F_1: Comparision

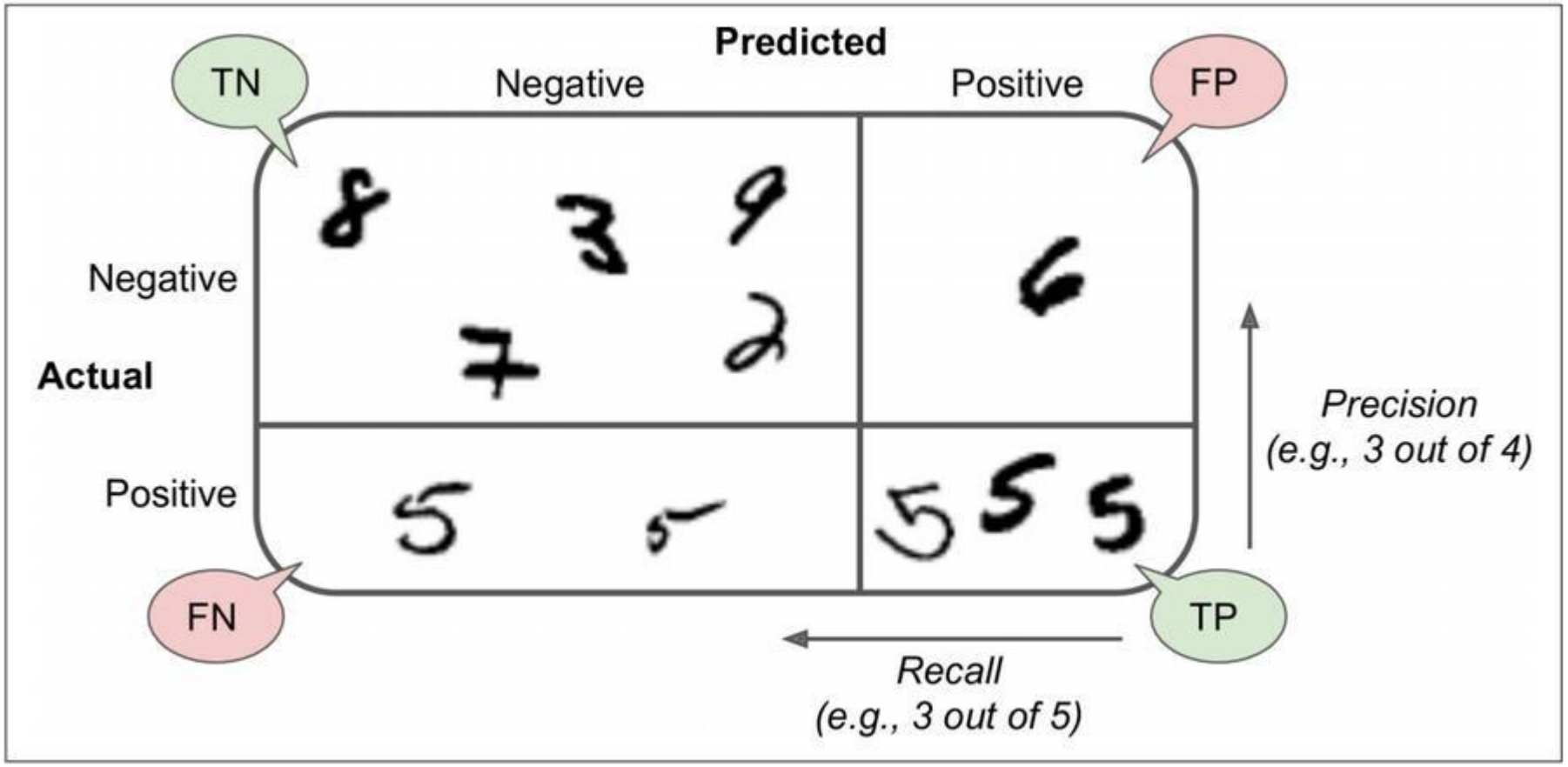
- F_1: favors classifier with similar precision and recall
- Precision/Recall tradeoff
  - Precision and recall impact each other inversely
- Impact of threshold on precision and recall



Feasible to create a classifier with high precision but on what recall ?

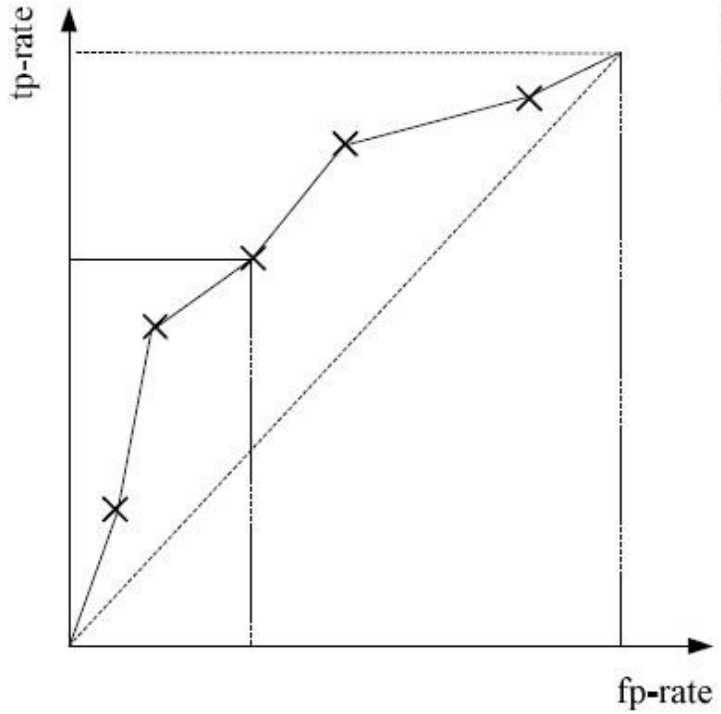
Confusion matrix

- Count the number of times some class A classified as some class B

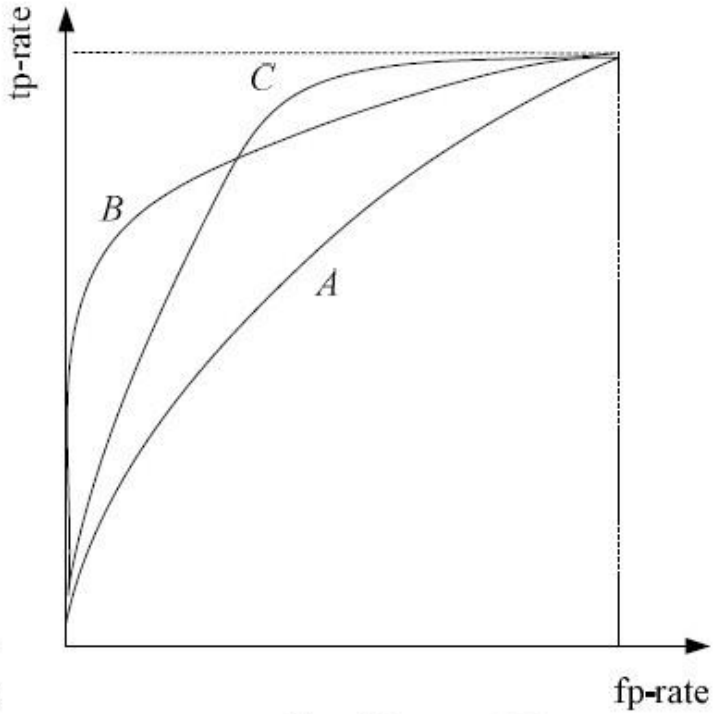


ROC (Receiver Oprating Characteristics) Curve

- Defined as True positive rate vs False positive rate
- Typically used with binary classifier
- Perfect classifier: with AUC (Area under the curve) 1



(a) Example ROC curve



(b) Different ROC curves for different classifiers

Measuring error: Regression

- Root mean square error (RMSE): Std deviation of errors

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

m: number of instances, $x^{(i)}$: vector of all feature values

h: system's prediction function aka hypothesis, **X**: matrix containing all feature values

- Mean Absolute Error (MAE): $\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right|$

Machine Learning: Performance Measures

- Classification: Misclassification error
- Regression : Prediction Error
- Clustering: Spread of the cluster (Will be discussed later in the course)
- Associations: Support/confidence (Will be discussed later in the course)
- Reinforcement Learning: Cost/Reward (Outside of the syllabus)

Measuring Error: Programming Notes

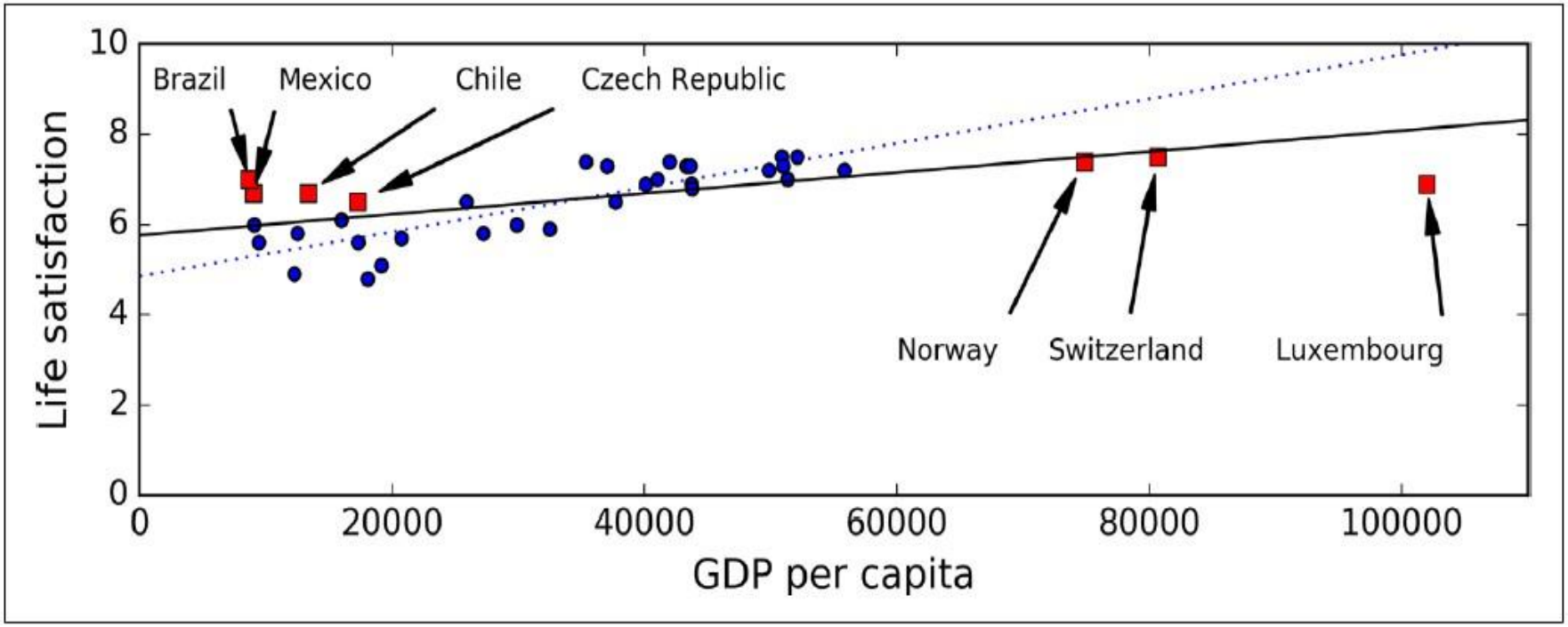

```
from sklearn.metrics import (accuracy_score, precision_score,
    recall_score, confusion_matrix, f1_score, mean_absolute_error,
    confusion_matrix,roc_curve, roc_auc_score, mean_squared_error )

fpr, tpr, thresholds = roc_curve(y_true, y_scores)

auc = roc_auc_score(y_true, y_scores)

f1 = f1_score(y_true, y_pred)

cm = confusion_matrix(y_true, y_pred)

accuracy = accuracy_score(y_true, y_pred)
error_rate = 1 - accuracy

precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

specificity = tn / (tn + fp)
false_alarm_rate = fp / (fp + tn)

rmse = np.sqrt(mean_squared_error(y_true, y_pred))
mae = mean_absolute_error(y_true, y_pred)
```

Contents

- Machine Learning Project: Big Picture
- Data: Feature Scaling
- Algorithms: Sampling, Validation and Evaluation
- Performance: Measuring Errors
- **Data: Data Issues**
- Algorithms: Bias and Variance

Issues with data: Insufficient Qty

- Must have enough data to be representative of new cases.
- Minimize Sampling Bias



Issues with data: Poor Quality

- Errors
- Outliers
- Noise
- Missing features
  - Fix: ignore or fill-up values.

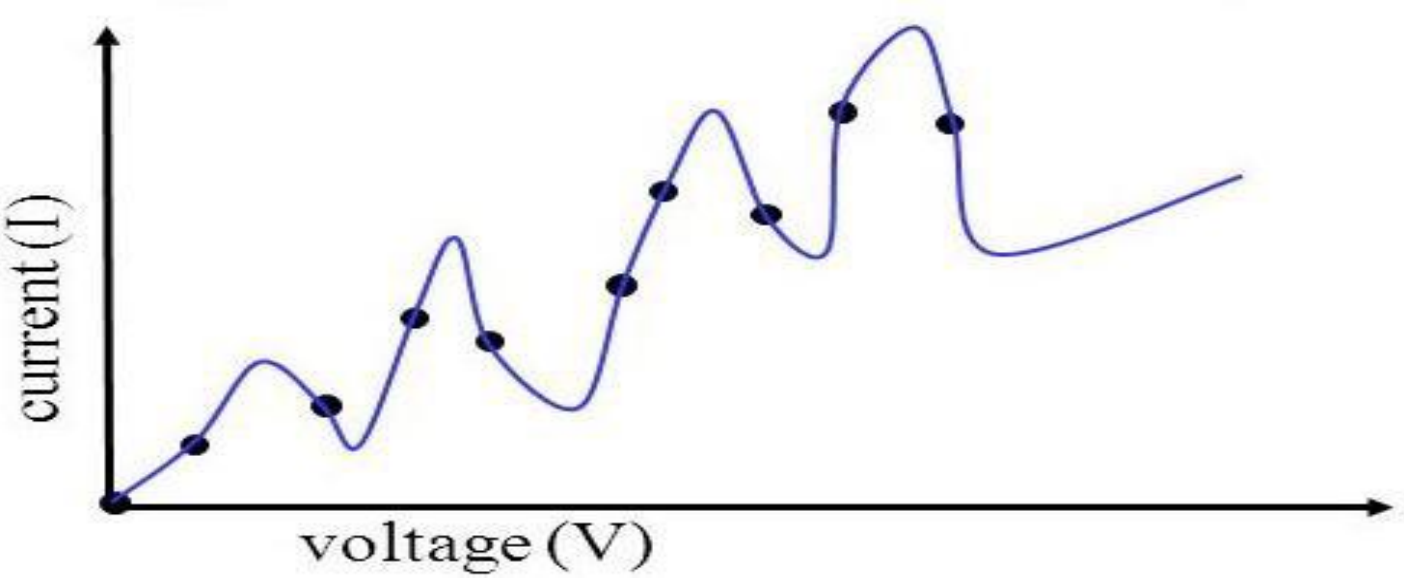
Practice: spend time cleaning up your data

Issues with data: Irrelevant features

- Must fix good set of features
- Feature Engineering
  - Selection: select the most useful features
  - Extraction: Combine existing features to create more useful ones.
  - Creation of new features by gathering new data

Over-Fitting

- Over-fitting: Over generalization
- When: When model learns the details and noise to the point of negatively impacting the performance on new model
  - When model is too complex

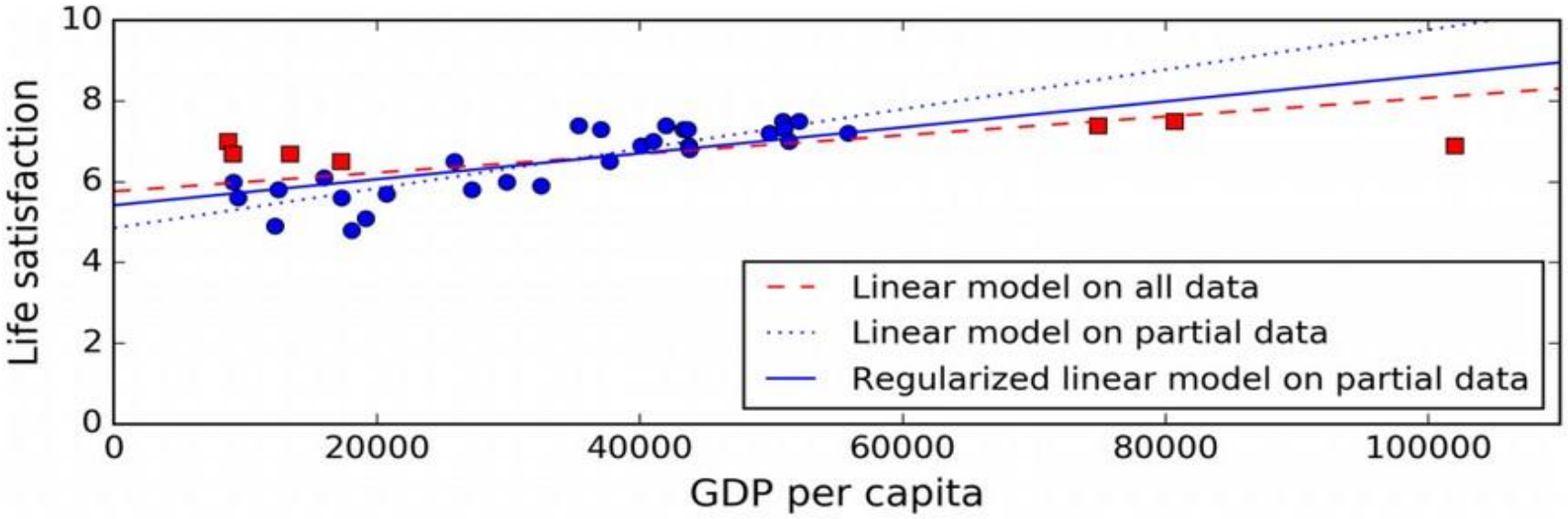


Over -Fitting of Ohm's law

Over-Fitting: Fix

- Regularization: Select models with fewer parameters
  - Hyperparameter: parameters that control the amount of regularization
- Reduce noise: fix data errors
- Gather more training data

Example: Life_satisfaction = $\theta_0 + \theta_1$ x GDP_per_capita



Underfitting

- Too simple model

- Fix
  - Selection more powerful model with more parameters
  - Feeding better features
  - Reduce constraints on the model

Contents

- Machine Learning Project: Big Picture
- Data: Feature Scaling
- Algorithms: Sampling, Validation and Evaluation
- Performance: Measuring Errors
- Data: Data Issues
- **Algorithms: Bias and Variance**

Prediction error of ML algorithms

There is a function $y = f(x) + \epsilon$

$E(\epsilon) = 0$; *and variance* σ^2

Goal is to find a function $\hat{f}(x)$ *that approximates f(x)*

Expected error on pont *x*:

$$\mathrm{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \left(\mathrm{Bias}\left[\hat{f}(x)\right]\right)^2 + \mathrm{Var}\left[\hat{f}(x)\right] + \sigma^2$$

- Bias error : $\mathrm{E}\left[\hat{f}(x)\right] - f(x)$
- Variance error: $\mathrm{E}[\hat{f}(x)^2] - \mathrm{E}[\hat{f}(x)]^2$
- Irreducible error: σ^2

Bias

- Assumptions that lead to generalization

Two types

- Language bias: assumption about model (Linear, non linear etc.)
- Search bias: the order in which models are examined
- Low Bias: Decision Trees, k-Nearest Neighbors and Support Vector Machines.
- High bias: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Variance

- The estimate of the target function will change given different training data

- Low variance: small changes to the est. of the target function
  - Linear Regression, Linear Discriminant Analysis and Logistic Regression.

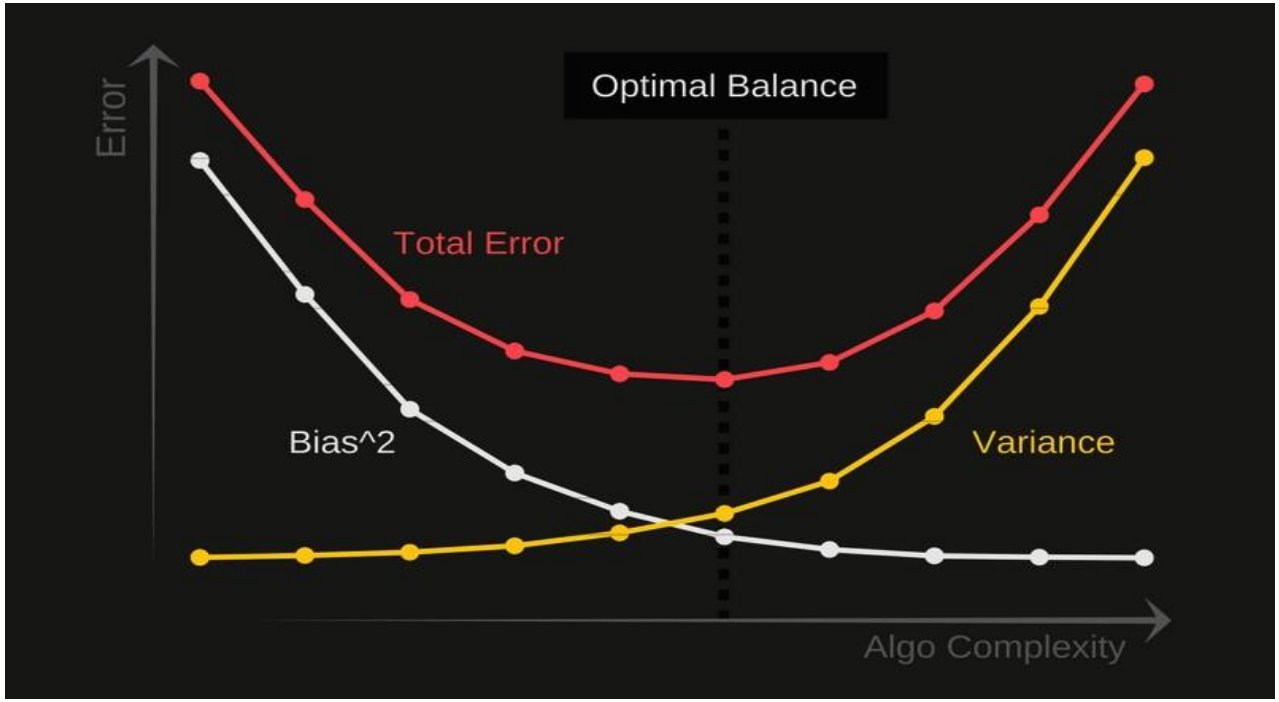
- High variance: large changes to the est. of the target function
  - nonparametric ML algorithms, e.g. Decision Trees, k-Nearest Neighbors, SVM

Bias and variance

Picture source: https://elitedatascience.com

Bias vs Variance

- ML work flow Goal: Low bias and low variance



- Typical Observation
  - High bias/Low variance: Parametric / linear ML algorithms
  - Low bias/high variance: Non-parametric / non-linear machine learning algorithms

Learning from Data: Steps

- Selection performance measure
- Divide into test set and training set
- Visualize
- Look for correlations
- Prepare the data
  - Data cleaning
- Select and train a model
- Evaluate model
- Fine Tune model