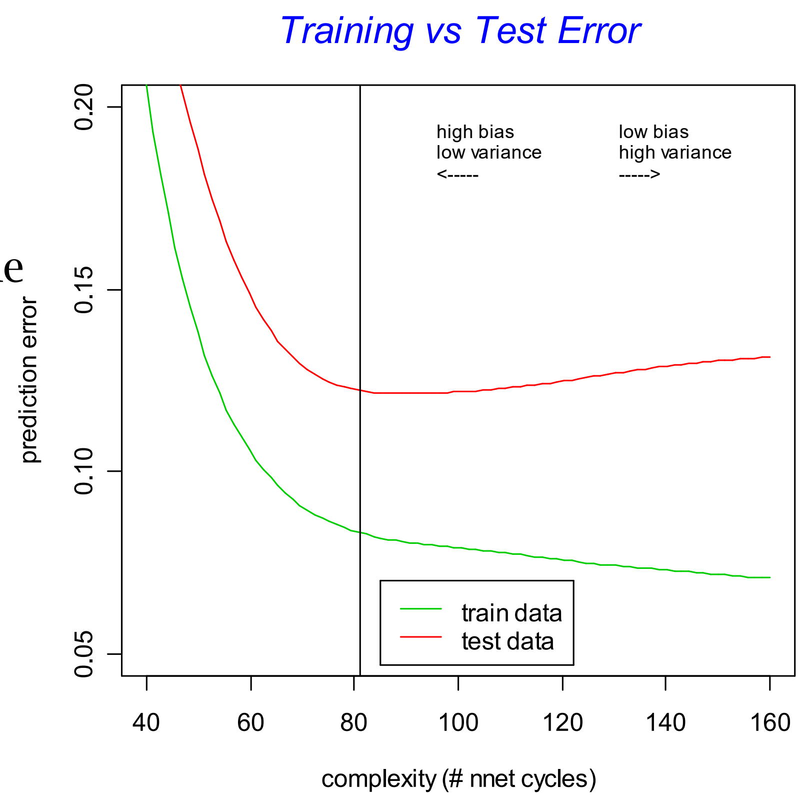
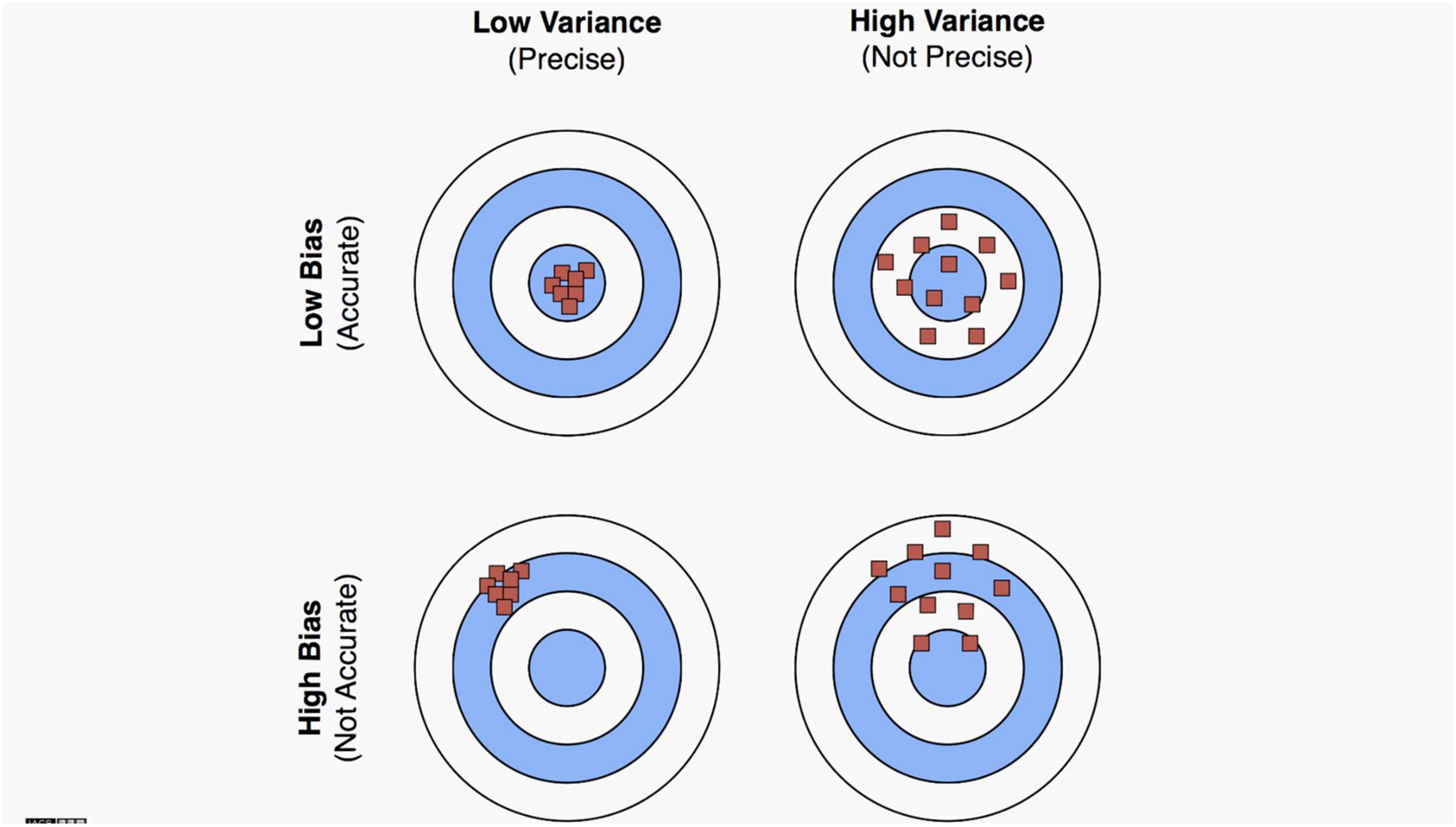
Objectives

- Regularization methods and Lasso
- Metrics for machine learning performance
    - Metrics for regression
    - Metrics for classification
- A short introduction to XGBoost
- How to install a new Python library and more Python examples

The Bias-Variance Tradeoff

- Complex model:
  - Low "bias":
    - the model fit is good on the *training data*.
    - i.e., the model value is close to the data's expected value.
  - High "Variance":
    - Model more likely to make a wrong prediction.
- Bias alone is not the name of the game.

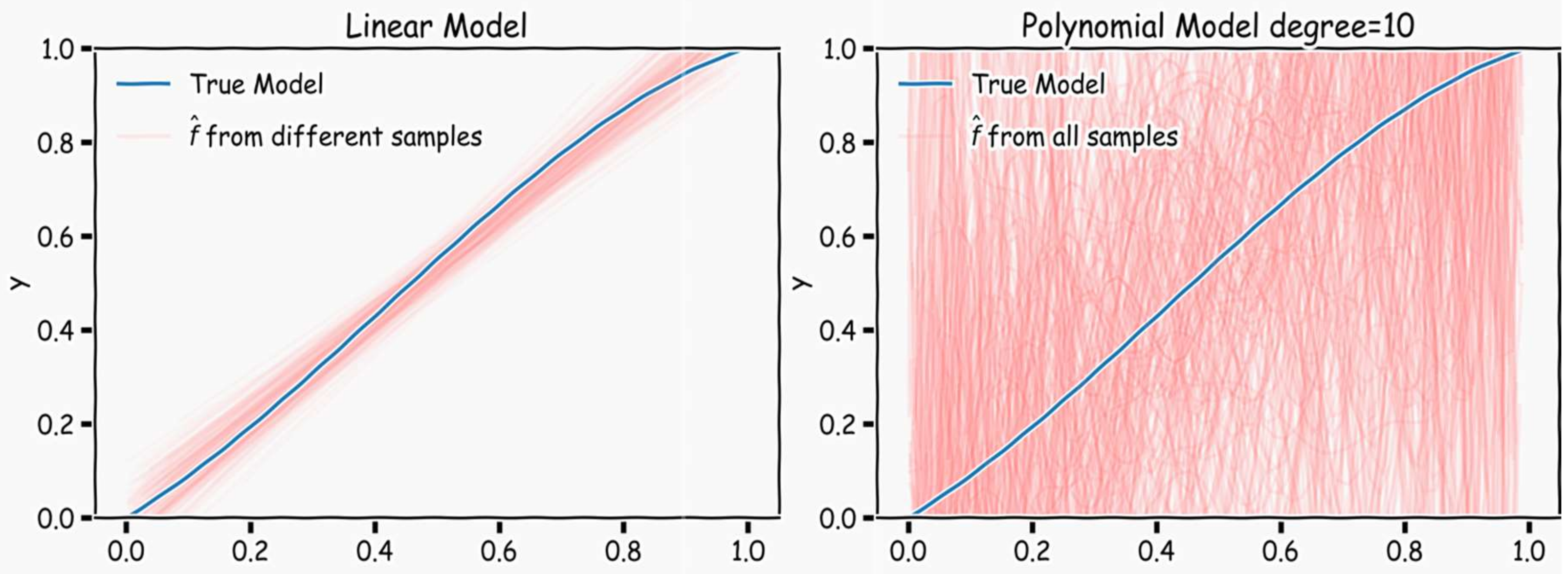



Low Variance
(Precise)
High Variance
(Not Precise)
Low Bias
(Accurate)
High Bias
(Not Accurate)

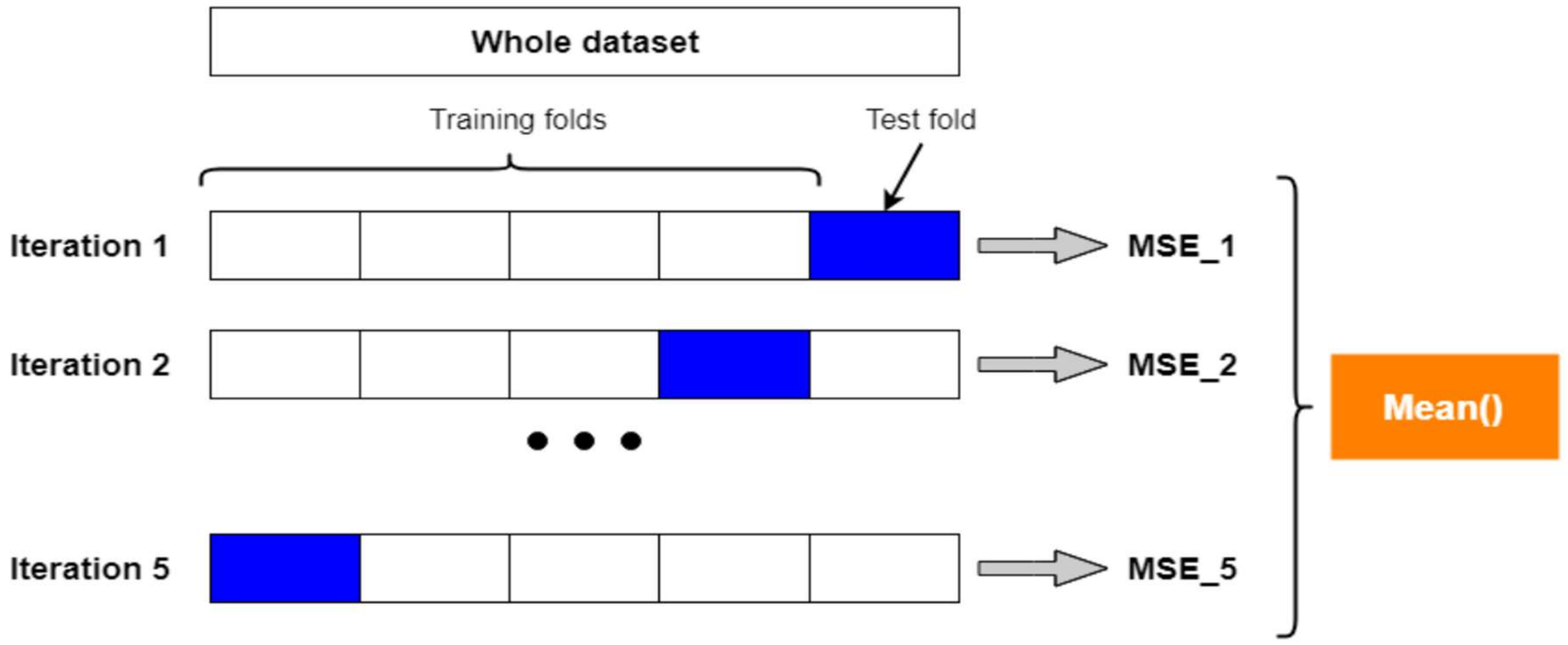
Bias vs Variance

Left: 2000 best fit straight lines, each fitted on a different 20 point training set.

Right: Best-fit models using degree 10 polynomial

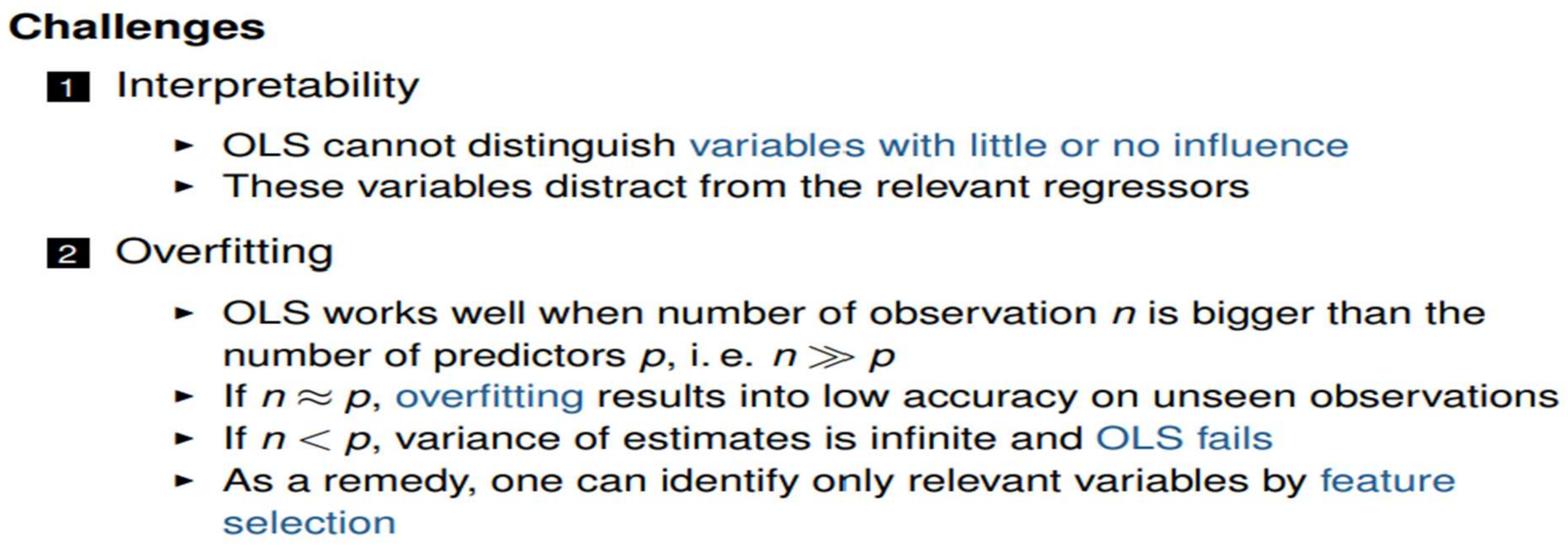


5-fold cross-validation for evaluating a model's performance



Motivation for Regularization

- Linear models are frequently favorable due to their interpretability and
- often good predictive performance
- Yet, Ordinary Least Squares (OLS) estimation faces challenges



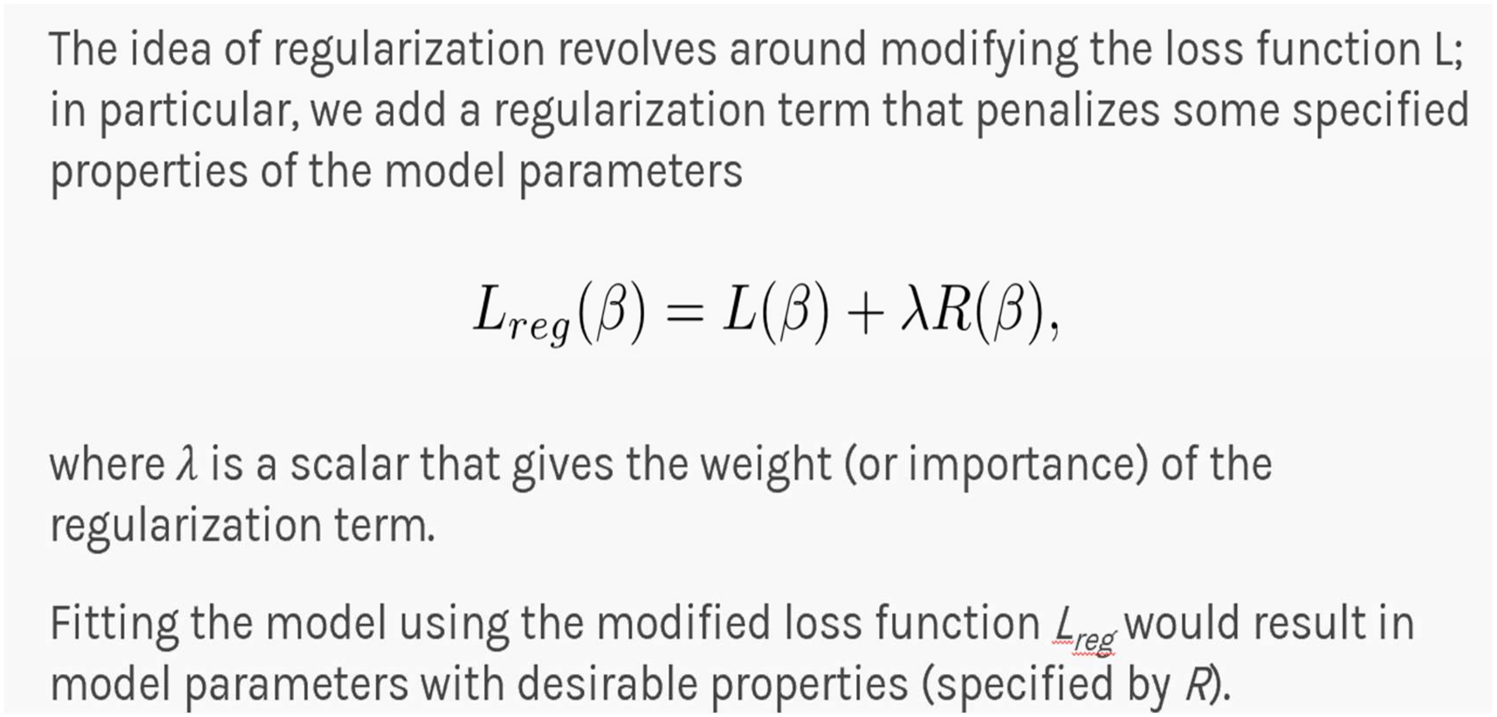
Challenges

1. Interpretability
   - OLS cannot distinguish variables with little or no influence
   - These variables distract from the relevant regressors
2. Overfitting
   - OLS works well when number of observation n is bigger than the number of predictors p, i. e. $n \gg p$
   - If $n \approx p$, overfitting results into low accuracy on unseen observations
   - If $n < p$, variance of estimates is infinite and OLS fails
   - As a remedy, one can identify only relevant variables by feature selection

Fitting techniques as alternatives to OLS

- Subset selection
  - Pick only a subset of all p variables which is assumed to be relevant
  - Estimate model with least squares using these reduced set of variables
- Dimension reduction
  - Project p predictors into a d-dimensional subspace with $d < p$
  - These d features are used to fit a linear model by least squares
- Shrinkage methods, also named regularization
  - Fit model with all p variables
  - However, some coefficients are shrunken towards zero
  - Has the effect of reducing variance

Regularization: an overview



The idea of regularization revolves around modifying the loss function L; in particular, we add a regularization term that penalizes some specified properties of the model parameters

$$L_{reg}(\beta) = L(\beta) + \lambda R(\beta),$$

where λ is a scalar that gives the weight (or importance) of the regularization term.

Fitting the model using the modified loss function L_{reg} would result in model parameters with desirable properties (specified by R).

Regularization Methods

- Fit linear models with least squares but impose constraints on the coefficients
- Instead, alternative formulations add a penalty in the OLS formula
- Best known are ridge regression and LASSO (least absolute shrinkage operator)
  - Ridge regression can shrink parameters close to zero
  - LASSO models can shrink some parameters exactly to zero

  $\rightarrow$ Performs implicit variable selection

Least Absolute Shrinkage and Selection Operator (LASSO)

- Ridge regression always includes p variables, but LASSO performs variables selection
- LASSO only changes the shrinkage penalty

$$\boldsymbol{\beta}_{\text{LASSO}} = \min_{\boldsymbol{\beta}} \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2}_{\text{RSS}} + \underbrace{\lambda \sum_{j=1}^{p}|\beta|_j}_{\text{shrinkage penalty}}$$

- Here, the LASSO uses the L_1-norm $\|\boldsymbol{\beta}\|_1 = \sum_j |\beta_j|$
- This penalty allows coefficients to shrink towards exactly zero
- LASSO usually results into sparse models, that are easier to interpret

Comparison

Both ridge regression and LASSO can be rewritten as

$$\boldsymbol{\beta}_{\text{ridge}} = \min_{\boldsymbol{\beta}} \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2}_{\text{RSS}} \quad \text{s.t.} \quad \sum_{j=1}^{p}\beta_j^2 \le \theta$$

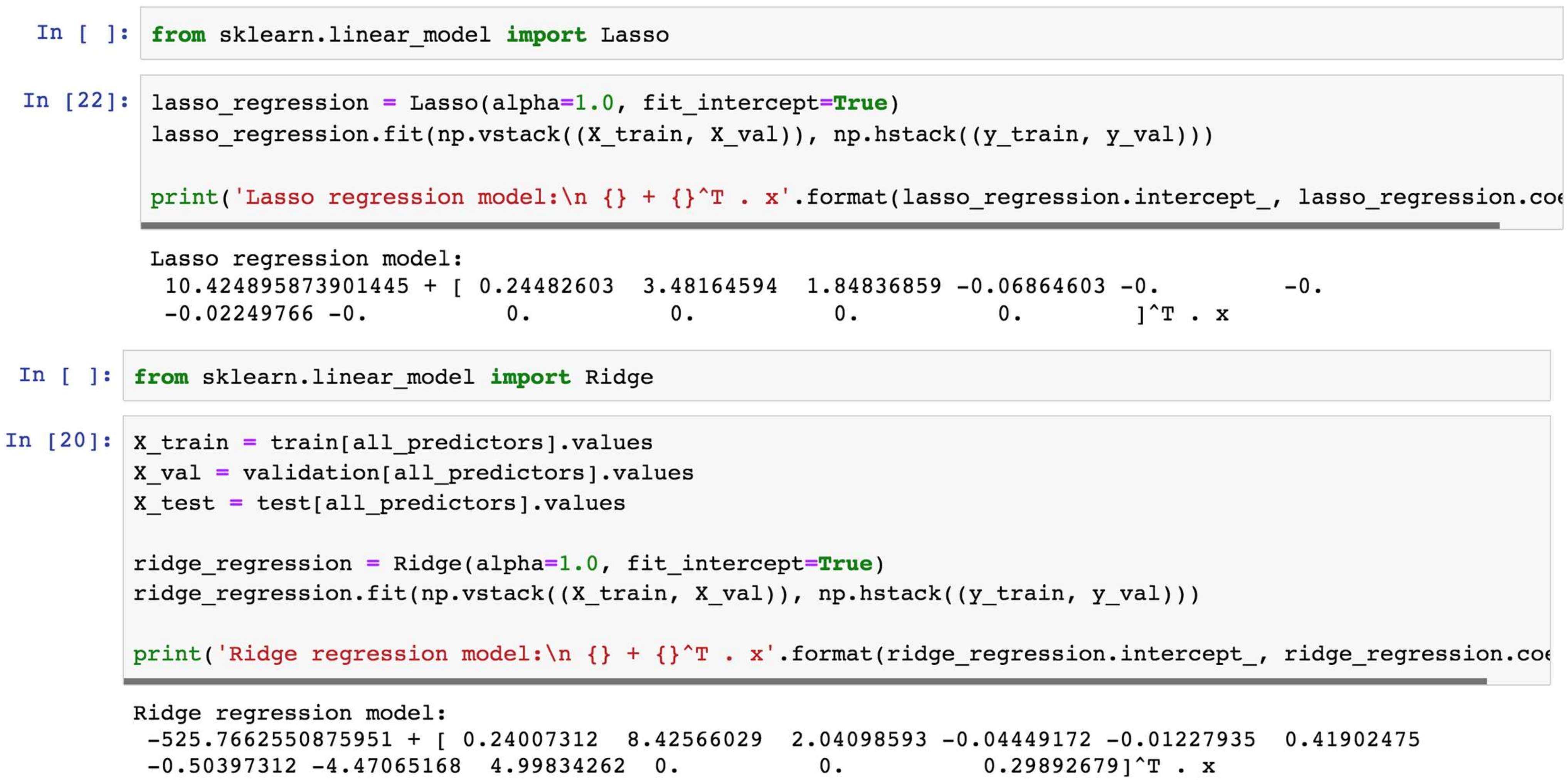
$$\boldsymbol{\beta}_{\text{LASSO}} = \min_{\boldsymbol{\beta}} \underbrace{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2}_{\text{RSS}} \quad \text{s.t.} \quad \sum_{j=1}^{p}|\beta_j| \le \theta$$

Choosing λ

In both ridge and LASSO regression, we see that the larger our choice of the **regularization parameter** λ, the more heavily we penalize large values in β,

- If λ is close to zero, we recover the MSE, i.e. ridge and LASSO regression is just ordinary regression.
- If λ is sufficiently large, the MSE term in the regularized loss function will be insignificant and the regularization term will force β_{ridge} and β_{LASSO} to be close to zero.

To avoid ad-hoc choices, we should select λ using cross-validation.



```
In [ ]: from sklearn.linear_model import Lasso
```

```
In [22]: lasso_regression = Lasso(alpha=1.0, fit_intercept=True)
         lasso_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

         print('Lasso regression model:\n {} + {}^T . x'.format(lasso_regression.intercept_, lasso_regression.coe
```

```
Lasso regression model:
 10.424895873901445 + [ 0.24482603  3.48164594  1.84836859 -0.06864603 -0.          -0.
 -0.02249766 -0.          0.          0.          0.          0.        ]^T . x
```

```
In [ ]: from sklearn.linear_model import Ridge
```

```
In [20]: X_train = train[all_predictors].values
         X_val = validation[all_predictors].values
         X_test = test[all_predictors].values

         ridge_regression = Ridge(alpha=1.0, fit_intercept=True)
         ridge_regression.fit(np.vstack((X_train, X_val)), np.hstack((y_train, y_val)))

         print('Ridge regression model:\n {} + {}^T . x'.format(ridge_regression.intercept_, ridge_regression.coe
```

```
Ridge regression model:
 -525.7662550875951 + [ 0.24007312  8.42566029  2.04098593 -0.04449172 -0.01227935  0.41902475
 -0.50397312 -4.47065168  4.99834262  0.          0.          0.29892679]^T . x
```

Summary

- Regularization methods
  - Regularization methods bring advantages beyond OLS
  - Cross validation chooses tuning parameter λ
  - LASSO performs variable selection
  - Cross validation finds the best approach for a given dataset

Performance Metrics in Machine Learning

Regression metrics

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Error (MAE)

Mean Squared Error (MSE)

Mean squared error is perhaps the most popular metric used for regression problems. It essentially finds the average of the squared difference between the target value and the value predicted by the regression model.

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2$$

Where:

- y_j: ground-truth value
- y_hat: predicted value from the regression model
- N: number of datums

Root Mean Squared Error (RMSE)

Root Mean Squared Error corresponds to the square root of the average of the squared difference between the target value and the value predicted by the regression model. Basically, sqrt(MSE). Mathematically it can be represented as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \breve{y}_j)^2}$$

Mean Absolute Error (MAE)

Mean Absolute Error is the average of the difference between the ground truth and the predicted values. Mathematically, its represented as :

$$MAE = \frac{1}{N} \sum_{j=1}^{N} |y_j - \check{y}_j|$$

Where:

- y_j: ground-truth value
- y_hat: predicted value from the regression model
- N: number of datums

Classification metrics

- Accuracy
- Confusion Matrix (not a metric but fundamental to others)
- Precision and Recall
- F1-score
- AU-ROC

Examples of Classification Task

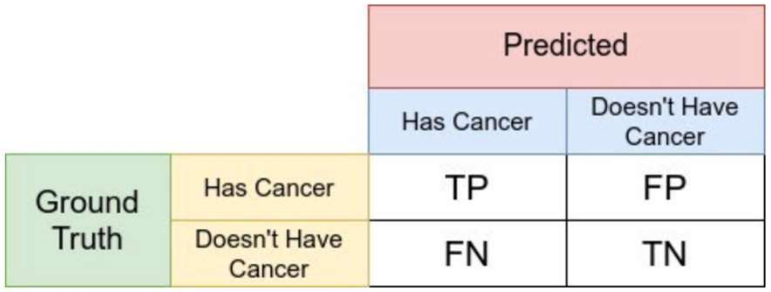
- Predicting tumor cells as benign or malignant
- Classifying credit card transactions as legitimate or fraudulent
- Categorizing news stories as finance, weather, entertainment, sports, etc

Accuracy

- Accuracy: percentage of correct classifications

$$\text{Accuracy} = \frac{\text{Total test instances classified correctly}}{\text{Total number of test instances}}$$

Confusion Matrix



		Predicted	
		Has Cancer	Doesn't Have Cancer
Ground Truth	Has Cancer	TP	FP
	Doesn't Have Cancer	FN	TN

Confusion Matrix for H^0

- **True Positive(TP)** signifies how many positive class samples your model predicted correctly.
- **True Negative(TN)** signifies how many negative class samples your model predicted correctly.
- **False Positive(FP)** signifies how many negative class samples your model predicted incorrectly. This factor represents **Type-I error** in statistical nomenclature. This error positioning in the confusion matrix depends on the choice of the null hypothesis.
- **False Negative(FN)** signifies how many positive class samples your model predicted incorrectly. This factor represents **Type-II** error in statistical nomenclature.

Precision

Precision is the ratio of true positives and total positives predicted:

$$P = \frac{TP}{TP+FP} = \frac{\text{Cancer patients correctly identified}}{\text{Cancer patients correctly identified+incorrectly labelled cancer patients as non-cancerous}}$$

0<P<1

Recall/Sensitivity/Hit-Rate

A **Recall** is essentially the ratio of true positives to all the positives in ground truth.

$$R = \frac{TP}{TP+FN} = \frac{\text{Cancer patients correctly identified}}{\text{Cancer patients correctly identified+incorrectly labelled non-cancer patients as cancerous}}$$

$0<R<1$

F1-score

The F1-score metric uses a combination of precision and recall. In fact, the F1 score **is the harmonic mean of the two.** The formula of the two essentially is:

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

A low F1 score tells you (almost) nothing—it only tells you about performance at a threshold. Low recall means we didn't try to do well on very much of the entire test set. Low precision means that, among the cases we identified as positive cases, we didn't get many of them right.

- ***Macro F1*** calculates the F1 separated by class but not using weights for the aggregation:

$$F1_{class1} + F1_{class2} + \cdots + F1_{classN}$$

  which resuls in a bigger penalisation when your model does not perform well with the minority classes(which is exactly what you want when there is imbalance)

- ***Weighted F1 score*** calculates the F1 score for each class independently but when it adds them together uses a weight that depends on the number of true labels of each class:

$$F1_{class1} * W_1 + F1_{class2} * W_2 + \cdots + F1_{classN} * W_N$$

  therefore favouring the majority class (which is want you usually dont want)

AUROC (Area under Receiver operating characteristics curve)

Better known as AUC-ROC score/curves. It makes use of true positive rates(TPR) and false positive rates(FPR).

$$TPR = \frac{TP}{TP+FN} \qquad FPR = \frac{FP}{FP+TN}$$

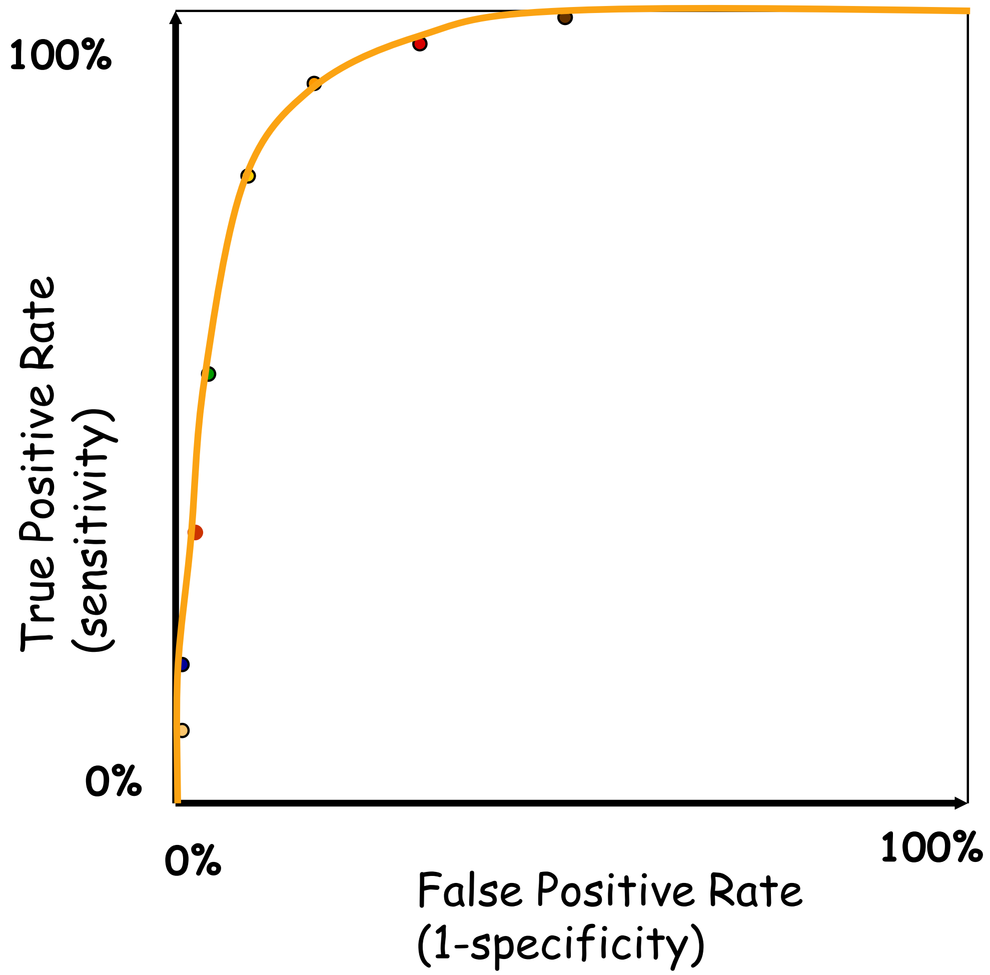
- Intuitively **TPR/recall** corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points. In other words, the higher the TPR, the fewer positive data points we will miss.
- Intuitively **FPR/fallout** corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points. In other words, the higher the FPR, the more negative data points we will misclassify.

ROC Analysis:
Example: Evaluating Medical Tests

- Sensitivity =The probability of having a positive test result among those with a positive diagnosis for the disease
  - SE = True Positives / (True Positives + False Negatives)

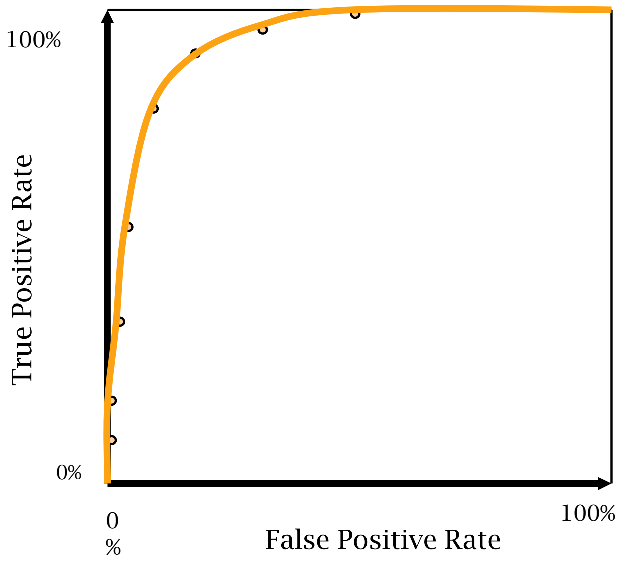
- Specificity = The probability of having a negative test result among those with a negative diagnosis for the disease
  - SP = True Negatives / (True Negatives + False Positives)

ROC curve

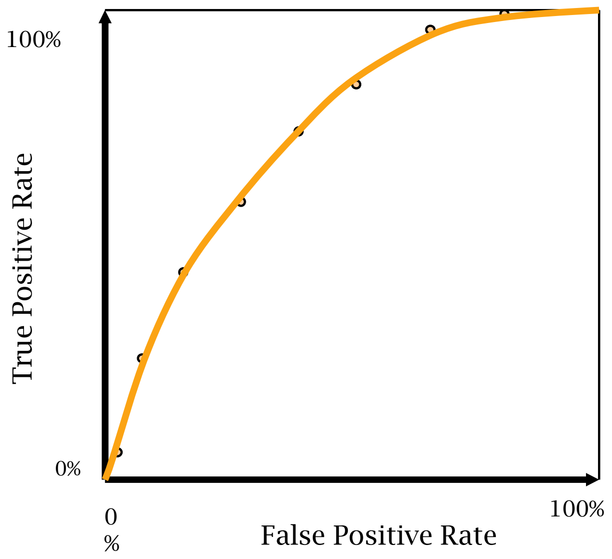

100%
0%
True Positive Rate
(sensitivity)
0%
100%
False Positive Rate
(1-specificity)

ROC curve comparison

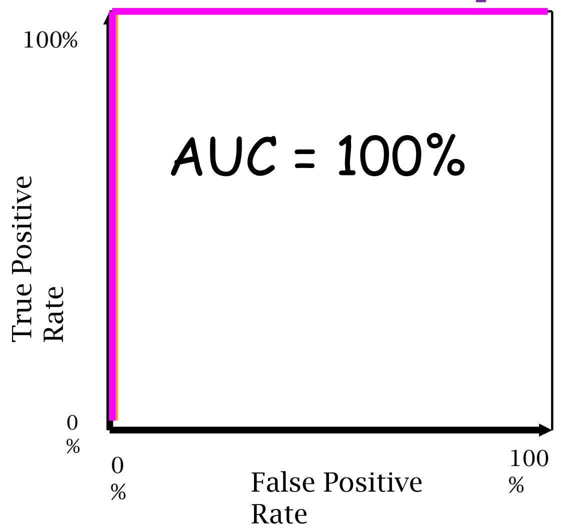
A good classifier

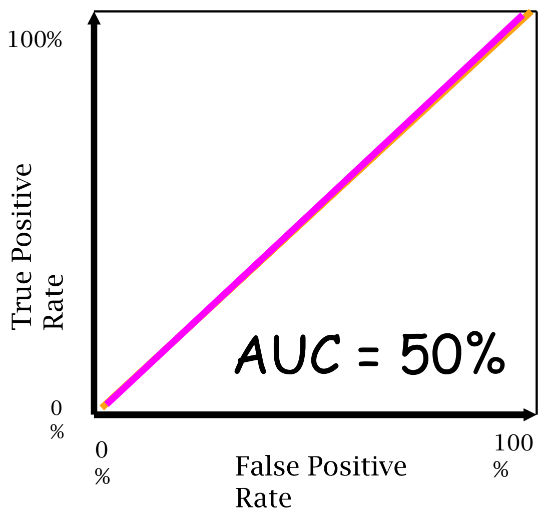


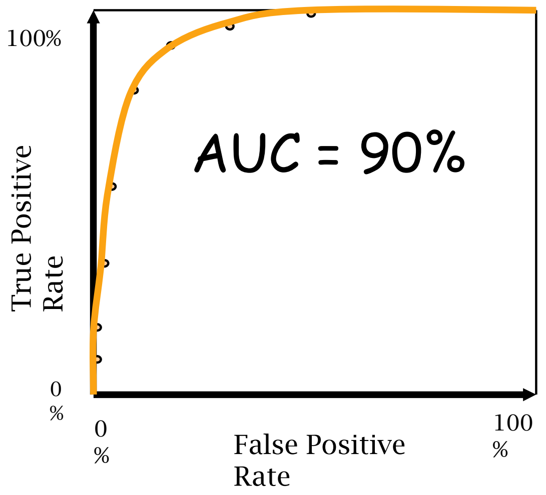
A poor classifier

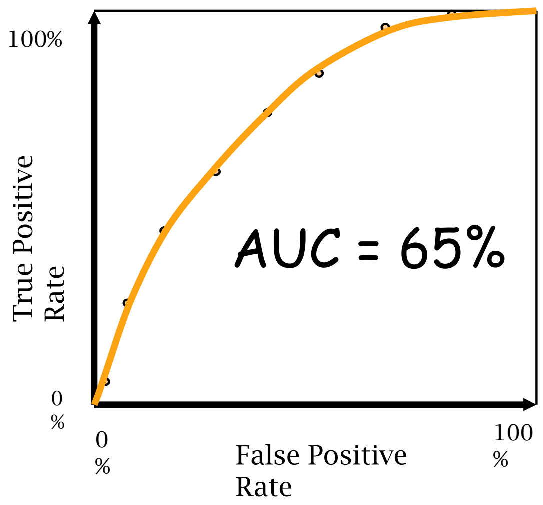


AUC for ROC curves



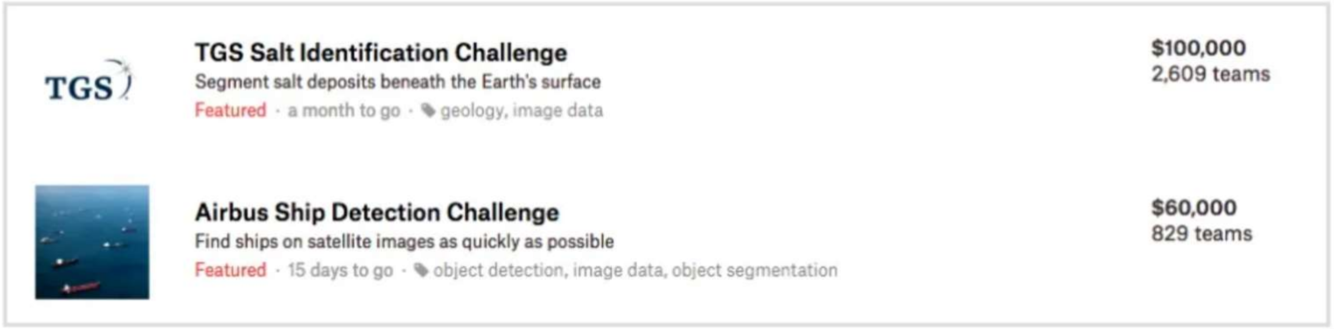






Kaggle.com

- A site where numerous machine learning competitions have been held.
  - When the organizer (usually companies like Amazon, Netflix) provides the data (usually real-life dataset), the team who best predicts the correct answer with the provided data wins.
- The winners will get a prize or get a chance to join the company depends on the competition.

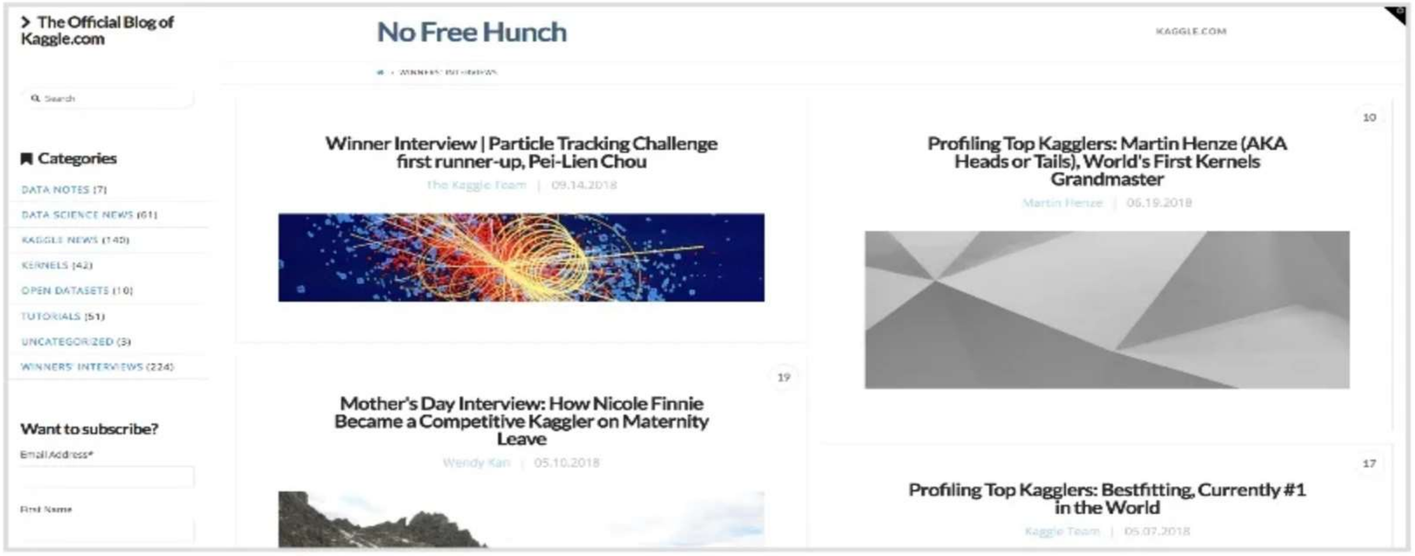


TGS Salt Identification Challenge
Segment salt deposits beneath the Earth's surface
Featured · a month to go · geology, image data
$100,000
2,609 teams

Airbus Ship Detection Challenge
Find ships on satellite images as quickly as possible
Featured · 15 days to go · object detection, image data, object segmentation
$60,000
829 teams

One important rule in Kaggle

- The winners have to disclose how they won.
  - http://blog.kaggle.com/category/winners-interviews/



> The Official Blog of Kaggle.com

No Free Hunch

KAGGLE.COM

WINNERS' INTERVIEWS

Search

Categories

DATA NOTES (7)
DATA SCIENCE NEWS (61)
KAGGLE NEWS (140)
KERNELS (42)
OPEN DATASETS (10)
TUTORIALS (51)
UNCATEGORIZED (3)
WINNERS' INTERVIEWS (224)

Want to subscribe?

Email Address*

First Name

Winner Interview | Particle Tracking Challenge first runner-up, Pei-Lien Chou
The Kaggle Team | 09.14.2018

19
Mother's Day Interview: How Nicole Finnie Became a Competitive Kaggler on Maternity Leave
Wendy Kan | 05.10.2018

10
Profiling Top Kagglers: Martin Henze (AKA Heads or Tails), World's First Kernels Grandmaster
Martin Henze | 06.19.2018

17
Profiling Top Kagglers: Bestfitting, Currently #1 in the World
Kaggle Team | 05.07.2018

- **One of the Popular Tools of Winners is XGBoost.**

What makes XGBoost so popular?

- Speed and performance: comparatively faster than other ensemble classifiers.
- Core algorithm is parallelizable
- Consistently outperforms other algorithm methods: It has shown better performance on a variety of machine learning benchmark datasets.
- Wide variety of tuning parameters : XGBoost internally has parameters for cross-validation, regularization, user-defined objective functions, missing values, tree parameters, scikit-learn compatible API etc.

Install xgboost library

- Install commands

```
pip install xgboost
pip install sklearn
```

- The core part of the code.

```
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(train_X, train_y)
test_y_hat = model.predict(test_X)
```

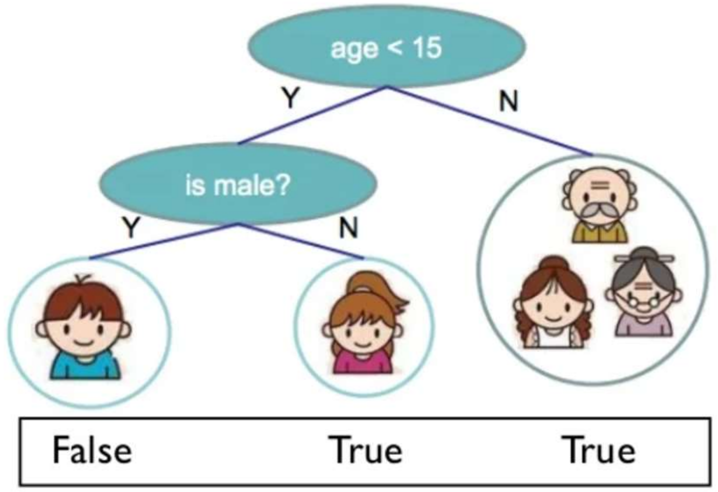
- **Very simple.** Isn’t it?

- Amazon SageMaker with XGBoost allows customers to train massive data sets on multiple machines. Just specify the number and size of machines on which you want to scale out, and Amazon SageMaker will take care of distributing the data and training process.

Decision trees

- Input: BMI, age, sex, …

- Output: Whether he / she has diabetes



True or False (1 or 0) in each leaf

Ensemble methods

- In wikipedia,
  - Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obstained from any of the constituent learning algorithms alone.
  - https://en.wikipedia.org/wiki/Ensemble_learning

- Any algorithms that integrate multiple models (algorithms) to better performance.
  - ex. bagging and boosting.

Models and parameters (optional reading)

- Model: assuming we have K trees.

space of functions containing all regression trees

$$\hat{y} = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

  - Recall: regression tree is a function that maps the attributes to the score.

- Parameters
  - Including structures of each tree, and the score in the leaf.
  - Or simply use function as parameters:

$$\Theta = \{f_1, f_2, ..., f_K\}$$

  - Instead learning weights in R^d, we are learning functions (trees).

Objective for tree ensembles of XGBoost (optional reading)

- Optimization form:

$$\min \underbrace{\boxed{\sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i)}}_{\text{Training loss}} + \underbrace{\boxed{\sum_{k=1}^{K} \Omega(f_k)}}_{\text{Complexity of the trees: Regularizer}}$$

- What is the possible ways to define Ω ?
  - The number of nodes in the tree, depth.
  - L2 norm of the leaf weights.
  - L1 norm of the lear weights.
  - Think about the role of L1 norm and L2 norm.

XGBoost for regression

```
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(train_X, train_y)
test_y_hat = model.predict(test_X)
```

XGBoost's most common parameters

- **learning_rate:** step size shrinkage used to prevent overfitting. Range is [0,1]
- **max_depth**: determines how deeply each tree is allowed to grow during any boosting round.
- **subsample:** percentage of samples used per tree. Low value can lead to underfitting.
- **colsample_bytree:** percentage of features used per tree. High value can lead to overfitting.
- **n_estimators:** number of trees you want to build.
- **objective:** determines the loss function to be used like reg:linear for regression problems, reg:logistic for classification problems with only decision,
- **binary:logistic** for classification problems with probability.

XGBoost also supports regularization parameters to penalize models as they become more complex and reduce them to simple (parsimonious) models.

- **gamma:** controls whether a given node will split based on the expected reduction in loss after the split. A higher value leads to fewer splits. Supported only for tree-based learners.
- **alpha**: L1 regularization on leaf weights. A large value leads to more regularization.
- **lambda**: L2 regularization on leaf weights and is smoother than L1 regularization.