Python
Train, Validation & Test in Machine Learning

Integrated Master's in Informatics Engineering

Learning and Extraction of Knowledge

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide

Prediction Model Selecting

- Model fit & prediction allows the creation of different ML techniques
- Different ML techniques define different approaches to predict the target variables
- How to select the ML model with the best prediction performance for a specific problem?
  - Access Predictive Models Performance

Assessing Predictive Models Performance

- Different Metrics are used to assess ML models performance
- Regression Problem – Mean Squared Error (MSE) / Mean Absolute Error (MAE) / Root Mean Squared Error (RMSE)
  - $MSE = \sum_{k=0}^{n}(Target(k) - Predicted(k))$^2 / n
  - $MAE = \sum_{k=0}^{n}|Target(k) - Predicted(k)|$ / n
  - RMSE = √MSE
- Classification Problem – Misclassification Error Rates (fraction of predictions that are incorrectly classified)
  - Confusion Matrix

Confusion Matrix

- Summarize the results of prediction models in a table
- Example: Binnary Classification Problem
  - Predict if Pacient has a Disease

Total Population	**Predicted: No**	**Predicted: Yes**
Real: No	True Negative	False Positive
Real: Yes	False Negative	True Positive

Total=165	**Predicted: No**	**Predicted: Yes**	
Real: No	TN=50	FP=10	60
Real: Yes	FN=5	TP=100	105
	55	110	

Confusion Matrix

- Summarize the results of prediction models in a table
- Example: Binnary Classification Problem
  - Dog Recognition Classifier

Total=165	**Predicted: No**	**Predicted: Yes**	
Real: No	TN=50	FP=10	60
Real: Yes	FN=5	TP=100	105
	55	110	

Accuracy = (TP+TN) / Total

Misclassification/Error Rate = (FP+FN)/Total

True Positive/Recall Rate = TP/(TP+FN)

False Positive Rate = FP/(TN+FP)

True Negative/Specificity Rate = TN/(TN+FP)

Precision = TP / (FP+TP)

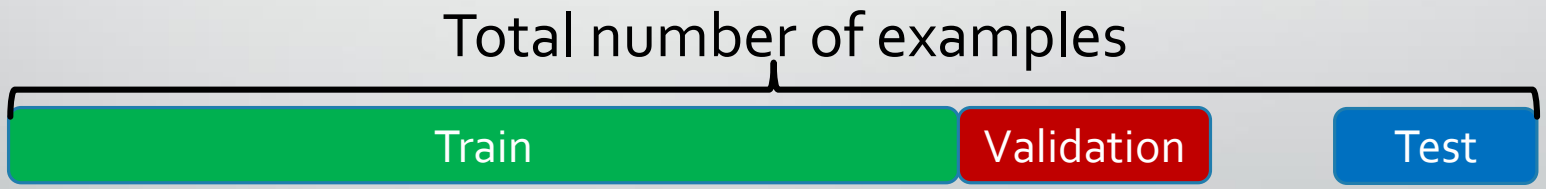
Prevalence = (FN+TP)/Total

Overfitting/Underfitting a Model

- In statistics and machine learning usually we split our data into 2/3 subsets:
  - Training dataset
  - Testing dataset
  - (sometimes Validation dataset)
- Some problems may affect the predictability of the model:
  - **Overfit model**
    - model has trained "too well"
    - model very accurate on the training dataset but is inaccurate on untrained or new data - model not generalized
  - **Underfit model**
    - model fails to fit the training dataset and is inaccurate to predict - model not generalized
    - usually the result from not presenting enough predictors/independent variables or to fit a linear model to data that is not linear

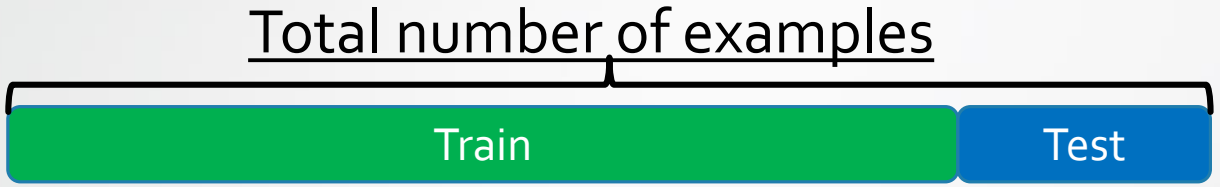
Dataset in Machine Learning

- **Training Dataset**: The sample of data used to fit the model
- **Validation Dataset**: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters
- **Test Dataset**: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset



Train/Test Split

- Training set contains features and outputs used to train the model
- Test set used to test the model's prediction




```
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt

# Load the Diabetes Housing dataset
columns = "age sex bmi map tc ldl hdl tch ltg glu".split() # Declare the columns names
diabetes = datasets.load_diabetes() # Call the diabetes dataset from sklearn
df = pd.DataFrame(diabetes.data, columns=columns) # load the dataset as a pandas data frame
y = diabetes.target # define the target variable (dependent variable) as y
```

```
# create training and testing vars
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.2)

# fit a model
lm = linear_model.LinearRegression()

model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)

# Score from model
print "Score:", model.score(X_test, y_test)
```

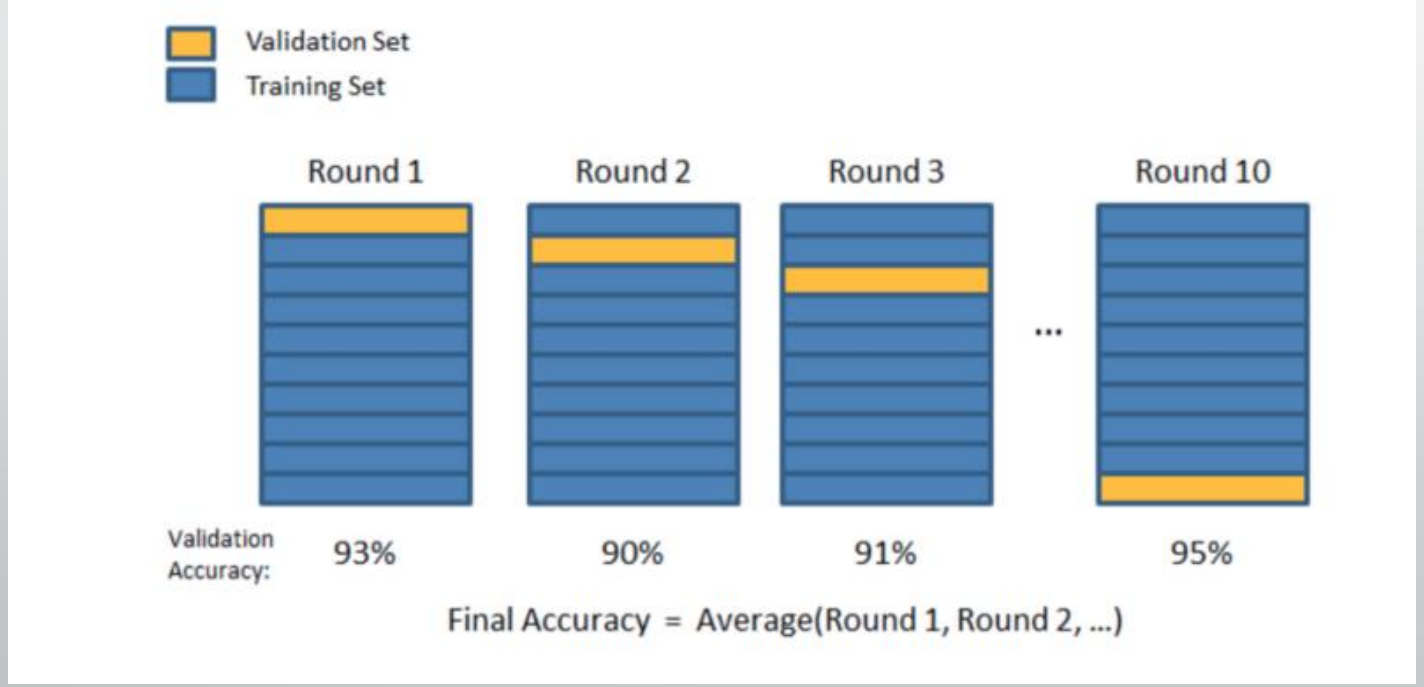
Train/Test Split

- **Advantages**:
  - Easy to apply
  - Easy to understand
- **Disadvantages**:
  - Danger of spliting data non-randomly
  - Risk of overfitting

- **Solution**: K-Fold Cross-validation

K-Fold Cross Validation

- Split data into k different subsets
- Train on K-1 subsets
- Hold last subset for validation
- Average the model's performance agaisn't each subsets



K-Fold Cross Validation

- Split Study Cases in 10-Fold

```
kf = KFold(n_splits=10) # Define the split - into 10 folds
kf.get_n_splits(X_train) # returns the number of splitting iterations in the cross-validator

print(kf)
```

- Score & Predict with 10-Fold Cross Validation

```
# Necessary imports:
from sklearn.cross_validation import cross_val_score, cross_val_predict
from sklearn import metrics

# Perform 10-fold cross validation
scores = cross_val_score(model, df, y, cv=kf)
print("Cross-validated scores:", scores)
print("Baseline Accuracy: %.2f%% (%.2f%%)" % (scores.mean()*100, scores.std()*100))
# Make cross validated predictions
predictions = cross_val_predict(model, df, y, cv=10)
plt.scatter(y, predictions)
```

K-Fold Cross Validation

- Higher number of folds implies:
  - Lower Bias Error
  - Higher Variance Error
  - Higher Computational Power
- Lower number of folds implies:
  - Higher Bias Error
  - Lower Variance Error
  - Lower Computational Power

Python
Train, Validation & Test in Machine Learning

Integrated Master's in Informatics Engineering

Learning and Extraction of Knowledge

2018/2019

Synthetic Intelligence Lab

Filipe Gonçalves

César Analide