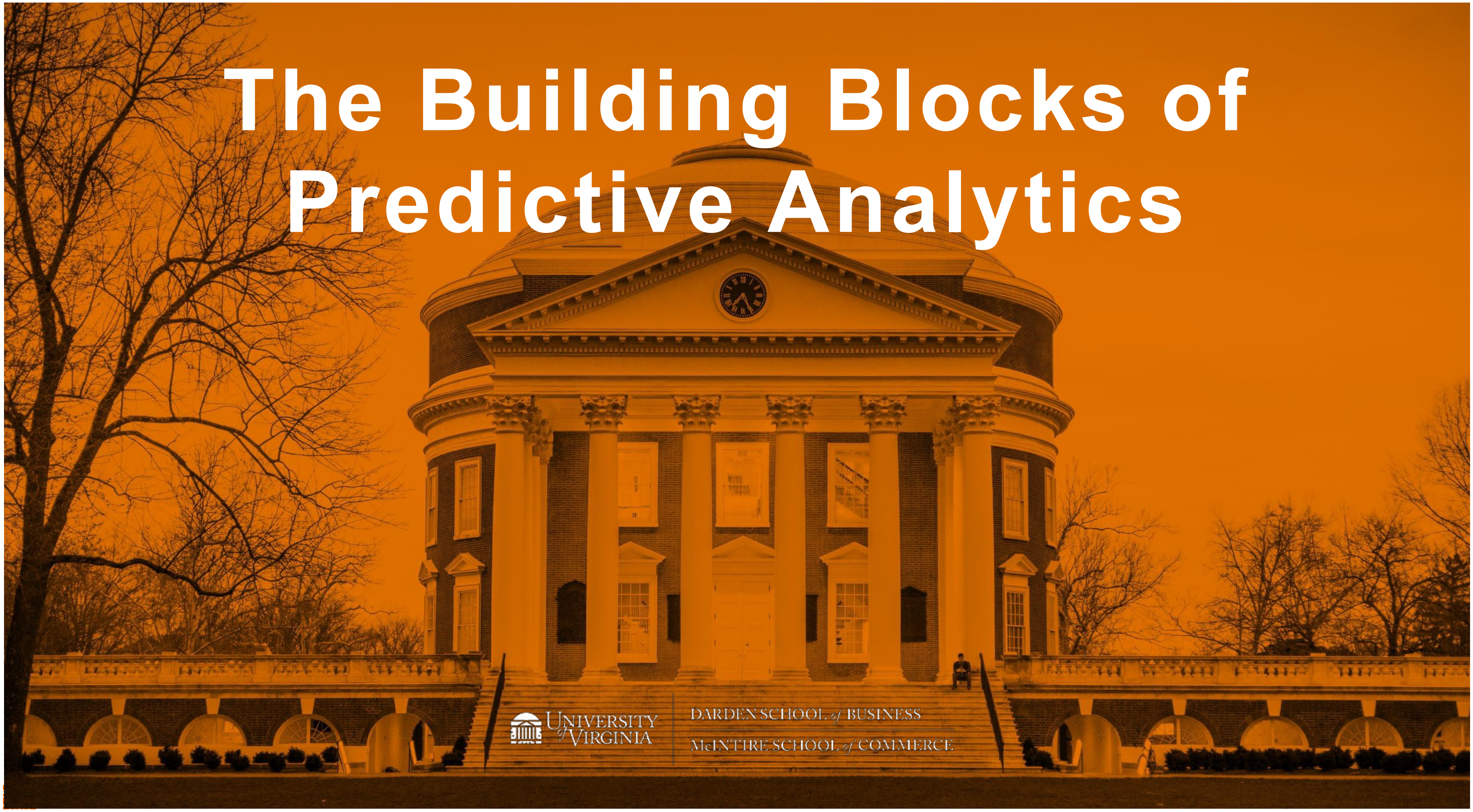
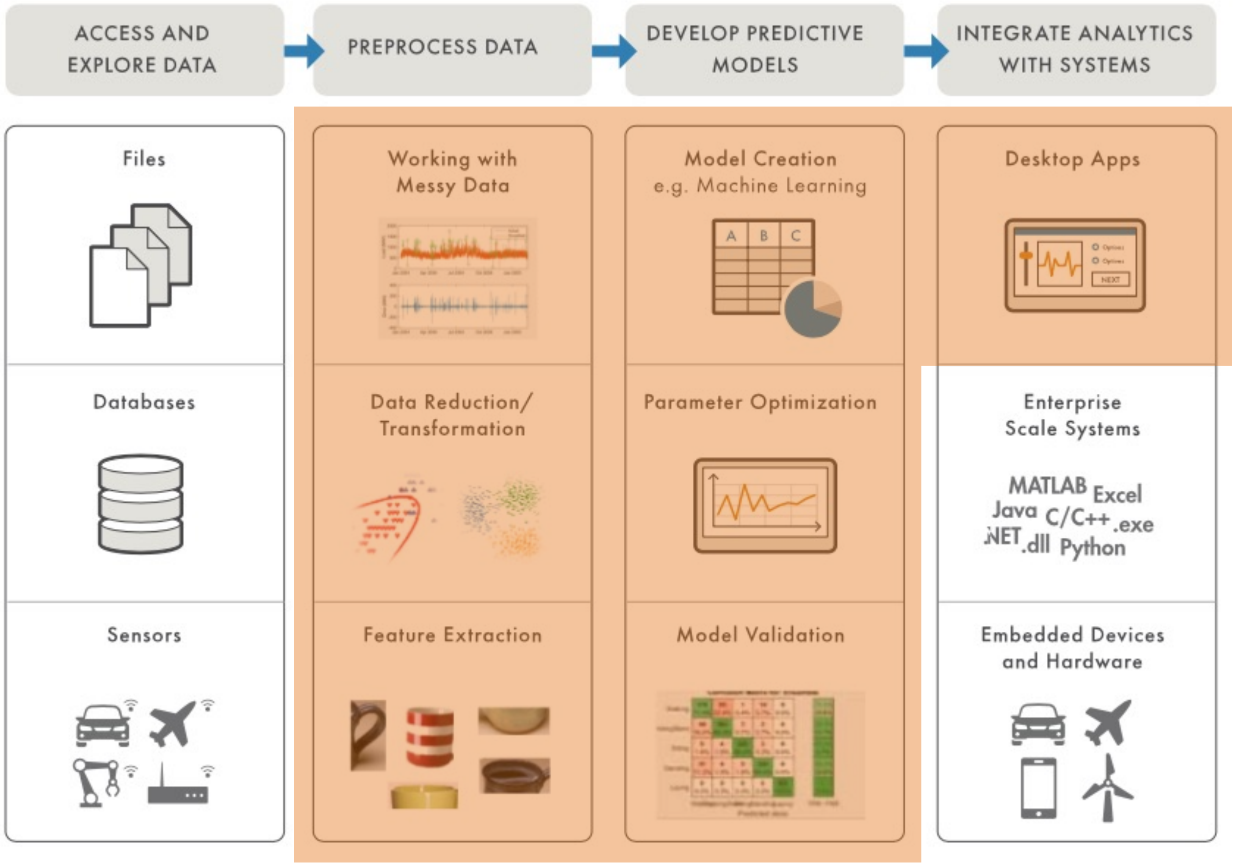
The Building Blocks of Predictive Analytics



Predictive Analytics Workflow



Develop Predictive Models

Machine learning models cheat sheet

Supervised learning

Data scientists provide input, output and feedback to build model (as the definition)

EXAMPLE ALGORITHMS:

Linear regressions
- sales forecasting
- risk assessment

Support vector machines
- image classification
- financial performance comparison

Decision tree
- predictive analytics
- pricing

Unsupervised learning

Use deep learning to arrive at conclusions and patterns through unlabeled training data.

EXAMPLE ALGORITHMS:

Apriori
- sales functions
- word associations
- searcher

K-means clustering
- performance monitoring
- searcher intent

Semi-supervised learning

Builds a model through a mix of labeled and unlabeled data, a set of categories, suggestions and exampled labels.

EXAMPLE ALGORITHMS:

Generative adversarial networks
- audio and video manipulation
- data creation

Self-trained Naïve Bayes classifier
- natural language processing

Reinforcement learning

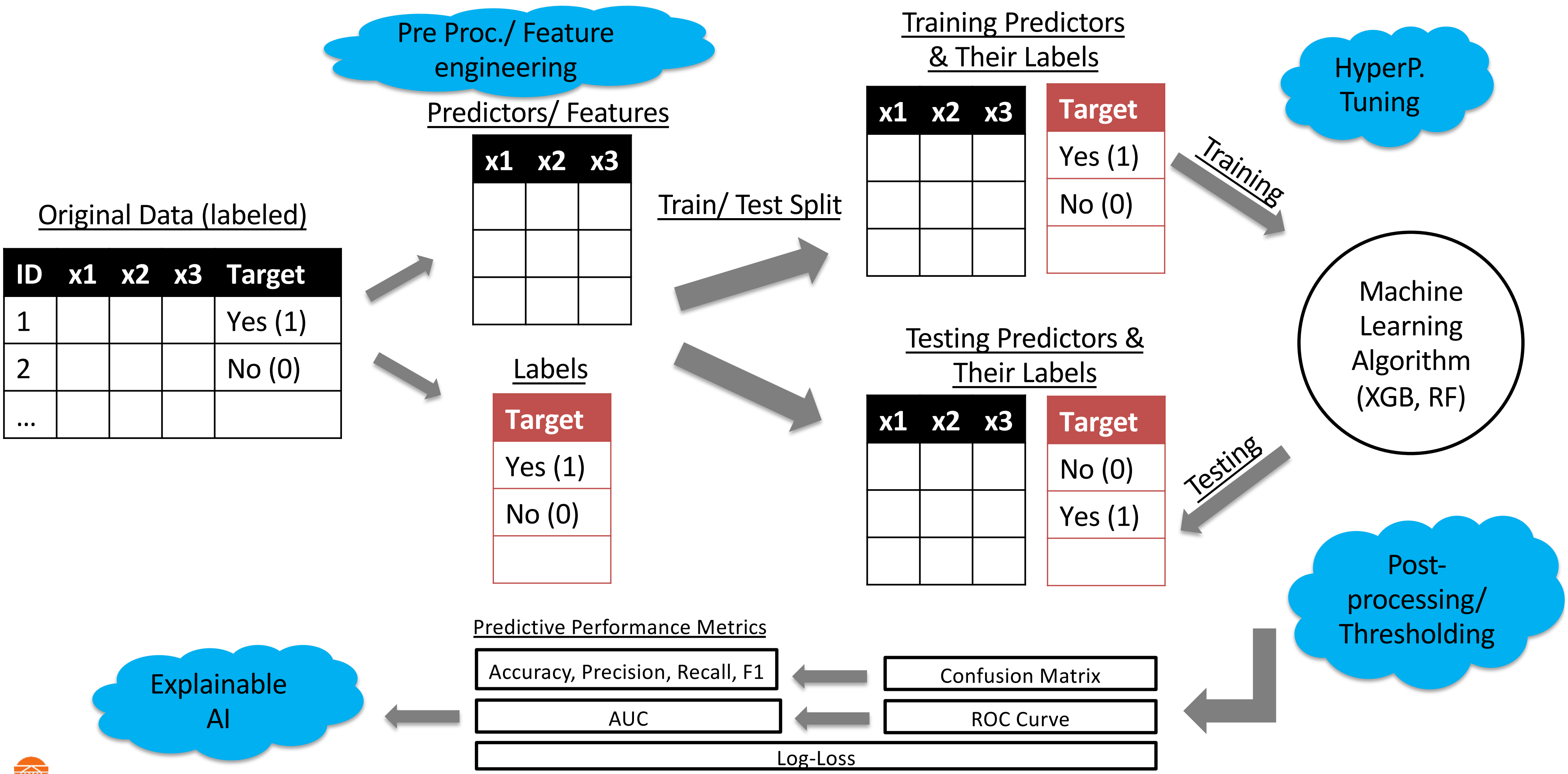
Self-interpreting but based on a system of rewards and punishments learned through trial and error, seeking maximum reward.

EXAMPLE ALGORITHMS:

Q-learning
- policy creation
- consumption reduction

Model-based value estimation
- linear tasks
- estimating parameters

How to Develop a Classifier?



PREDICTION SCORES AND CONFIDENCE

Observation	Actual Training Label/Class	Predicted Label/Class	Classification Type	Prediction Score Model A	Prediction Score Model B
1	Yes	Yes	TP	0.9	1.0
2	No	No	TN	0.2	0.1
3	Yes	Yes	TP	0.8	0.9
4	Yes	No	FN	0.1	0.4
5	Yes	Yes	TP	0.7	0.8
6	Yes	No	FN	0.3	0.4
7	No	No	TN	0.4	0.3
8	No	No	TN	0.3	0.2
9	No	No	TN	0.4	0.3
10	No	Yes	FP	0.6	0.5

PRECISION AND RECALL

		Actual Class		
Predicted Class		Class = Yes	Class = No	**Precision**
	Class = Yes	3	1	$\frac{3}{4}$ 75%
	Class = No	2	4	$\frac{4}{6}$ 66.7%
	Recall	$\frac{3}{5}$ 60%	$\frac{4}{5}$ 80%	

EVALUATING MODELS

ROC CURVE

		Actual Labels	
Predicted Labels		Yes	No
	Yes	3	1
	No	2	4

FPR = 1/5 = 0.2
TPR = 3/5 = 0.6

threshold = 0.5

threshold = 0

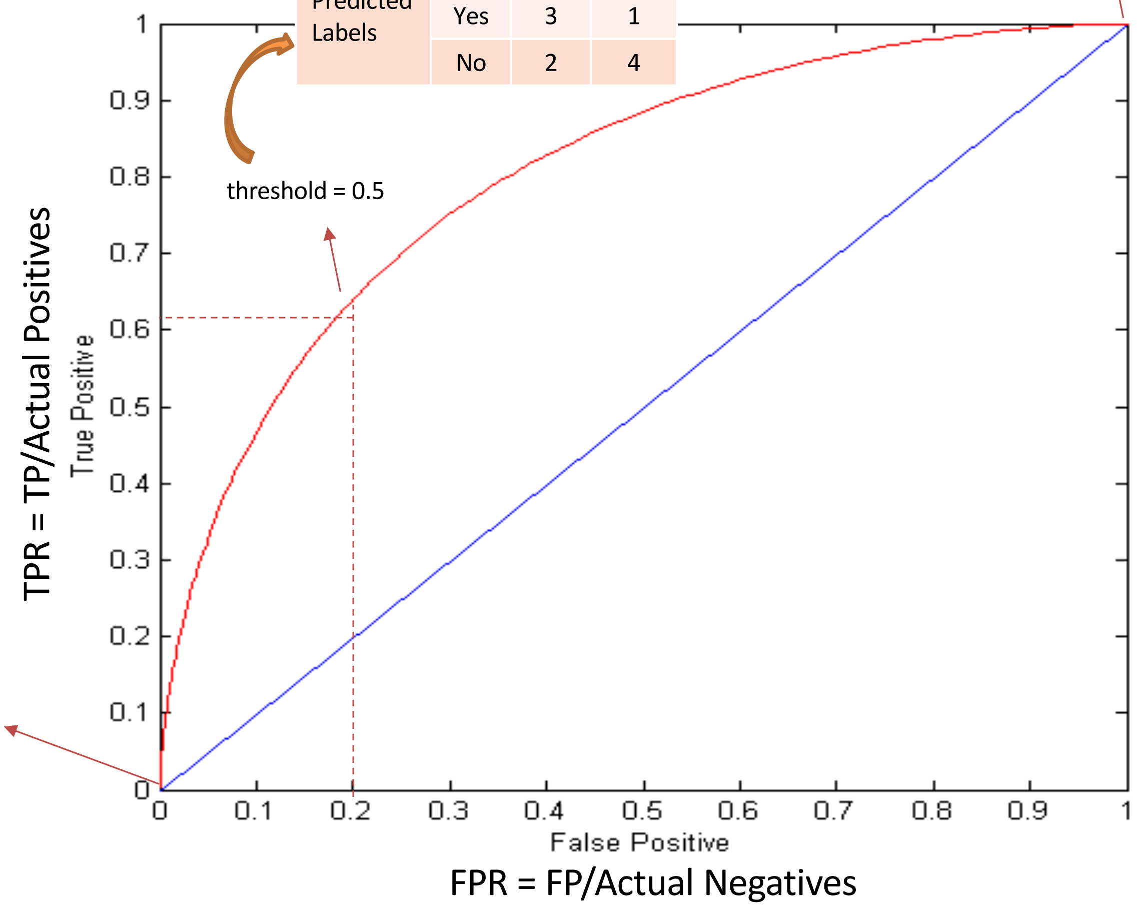
		Actual Labels	
Predicted Labels		Yes	No
	Yes	5	5
	No	0	0

FPR = 5/5 = 1
TPR = 5/5 = 1

FPR = 0/5 = 0
TPR = 0/5 = 0

		Actual Labels	
Predicted Labels		Yes	No
	Yes	0	0
	No	5	5

threshold = 1



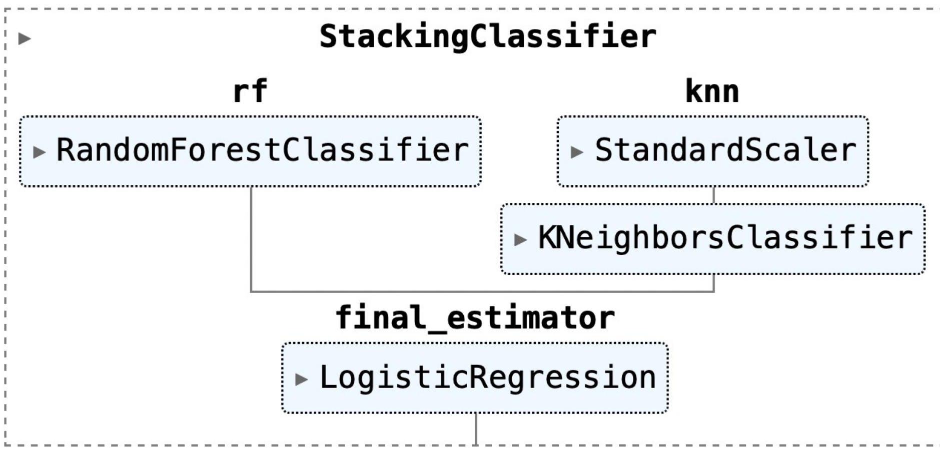
Advanced Concepts in Classification

1- Stacking classifiers

2- Voting classifiers

3- Hyperparameter optimization

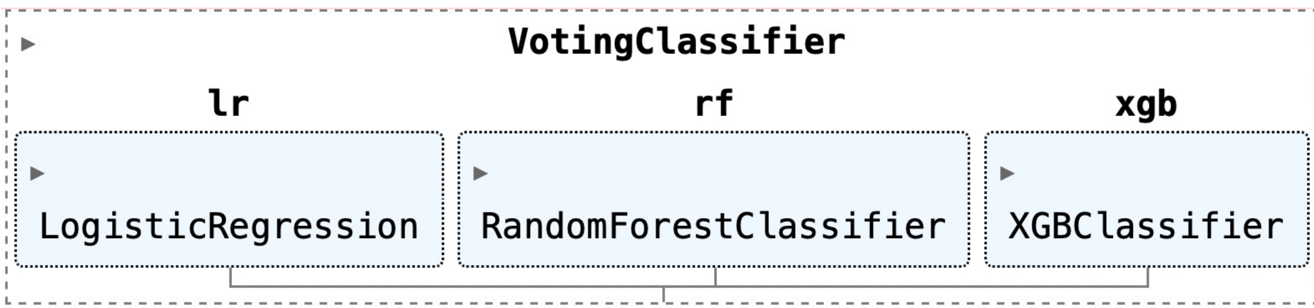
Stacking Classifiers




```python
%%time
estimators = [
    ('rf', RandomForestClassifier(n_estimators=50,
                                  random_state=42)),
    ('knn', make_pipeline(StandardScaler(),
                          KNeighborsClassifier(n_neighbors=5)))]
clf = StackingClassifier(
    estimators = estimators,
    final_estimator = LogisticRegression(),
    n_jobs = -1, verbose=True
)

clf.fit(trainData, trainLabels)
```

Voting Classifiers

```
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = XGBClassifier(random_state=1)

vclf = VotingClassifier(estimators=[
        ('lr', clf1), ('rf', clf2), ('xgb', clf3)],
                        voting='soft')

vclf.fit(trainData, trainLabels)
```



Hyperparameter Optimization

```
!pip install hyperopt
```

•••

```
# import packages for hyperparameters tuning
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
```

```
space={'max_depth': hp.quniform("max_depth", 3, 18, 1),
        'gamma': hp.uniform ('gamma', 1,9),
        'reg_alpha' : hp.quniform('reg_alpha', 40,180,1),
        'reg_lambda' : hp.uniform('reg_lambda', 0,1),
        'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
        'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1
        'n_estimators': hp.quniform('n_estimators', 100, 200, 20),
        'eta': hp.uniform('eta', 0.1,0.9),
        'seed': 0,
        'random_state': 1
    }
```

Hyperparameter Optimization- “hyperopt” Package

Step 1: Define a space for the hyperparameters.

Step 2: Create the objective function.

Step 3: Run trials.

Step 4: Export and use the best hyperparameter values in a new model.

Hyperopt

Step 1: Define a space for the hyperparameters.

```
space={'max_depth': hp.quniform("max_depth", 3, 18, 1),
        'gamma': hp.uniform ('gamma', 1,9),
        'reg_alpha' : hp.quniform('reg_alpha', 40,180,1),
        'reg_lambda' : hp.uniform('reg_lambda', 0,1),
        'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
        'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1
        'n_estimators': hp.quniform('n_estimators', 100, 200, 20),
        'eta': hp.uniform('eta', 0.1,0.9),
        'seed': 0,
        'random_state': 1
    }
```

Hyperopt

Step 2: Create the objective function.

```
def objective(space):
    clf=XGBClassifier(
        random_state = space['random_state'],
        eta = space['eta'],
        n_estimators = int(space['n_estimators']),
        max_depth = int(space['max_depth']),
        gamma = space['gamma'],
        reg_alpha = int(space['reg_alpha']),
        min_child_weight=int(space['min_child_weight']),
        colsample_bytree=int(space['colsample_bytree']))

    evaluation = [( trainData, trainLabels), ( testData, testLabels)]

    clf.fit(trainData, trainLabels,
            eval_set=evaluation, eval_metric="auc",
            early_stopping_rounds=10,verbose=False)


    pred = clf.predict(testData)
    accuracy = roc_auc_score(testLabels, pred>0.5)
    print ("SCORE:", accuracy)
    return {'loss': -accuracy, 'status': STATUS_OK }
```

Hyperopt

Step 3: Run trials.

```
trials = Trials()

best_hyperparams = fmin(fn = objective,
                        space = space,
                        algo = tpe.suggest,
                        max_evals = 100,
                        trials = trials)
```

Step 4: Export values and build a new model.

```
best_hyperparams = {
 'colsample_bytree': 0.8182902035285777,
 'eta': 0.7974676615890914,
 'gamma': 7.870558816361137,
 'max_depth': 13, # Should be int
 'min_child_weight': 4, # Should be int
 'n_estimators': 180, # Should be int
 'reg_alpha': 48.0,
 'reg_lambda': 0.8050195284705558}
```

```
best_xgb = XGBClassifier(**best_hyperparams, random_state=1)
best_xgb.fit(trainData, trainLabels)
```

Supervised Learning Cheat Sheet

S

ID	X	Y

X

Process	Method	Code
Feat. Select	Boruta	BorutaPy(…)
	sklearn.feature_selection	RFE(…)
Imputation	sklearn.impute	IterativeImputer(…)
Cat. Vars./ Scale Vars.	category_encoders	BaseNEncoder(…)
	sklearn.preprocessing	OneHotEncoder(…) StandardScaler(…)
All Feat. Eng.	feature-engine	MeanMedianImputer(…) OutlierTrimmer(…) MathematicalCombination(…) DropCorrelatedFeatures(…) DecisionTreeDiscretiser(…)

Y

Process	Method	Code
Encoding	sklearn.preprocessing	LabelEncoder(…)

Process	Method	Code
Split	sklearn.model_selection	train_test_split(…)

trainData/ Labels

X	y

Process	Method	Code
Select Algo.	mljar-supervised	AutoML(…)
	pycaret	compare_models(…)

Process	Method	Code
Hyper Param. Tuning	hyperopt	fmin(…)
	sklearn.model_selection	GridSearchCV(…)
		RandomizedSearchCV(…)

Process	Method	Code
Final Tuned Algo.	xgboost	XGBClassifier(…)
	catboost	CatBoostClassifier(…)
	lightgbm	LGBMClassifier(…)

testData/ Labels

X	y

E

Process	Method	Code
Model Eval.	sklearn.metrics	roc_auc_score() log_loss() confusion_matrix()
Interpret/ Explain	statsmodels	sm.Logit(y, X).fit()
	shap	shap.summary_plot(…) shap.plots.scatter(…) shap.force_plot(…) shap.plots.waterfall(…)
	lime	explainer.explain_instance(…)
	imodels	HSTreeClassifierCV(…)
	explainerdashboard	ClassifierExplainer(clf, testData, testLabels)

PhishCasting Competition (Leaderboard & Lessons Learned)

1- What algorithm did you use?

2- Did you do any feature engineering?

3- Explain your hyperparameter tuning approach.

4- Did you balance (resample) the training data?

5- Any best practices you would like to share?

?

VIRGINIA