W4995 Applied Machine Learning
Fall 2021

Lecture 5

Dr. Vijay Pappu

Announcements

- HW2 is released on 10/07 (please start early)
- HW2 is due on 10/21 11:59 PM EST
- Midterm 10/27 in-class
- Next class - Shouvik Mani

In today’s lecture, we will cover...

- Model Evaluation
- Calibration
- Automatic Machine Learning

Model Evaluation

Evaluation Metrics

- Evaluation metrics are generally used to measure the performance of an ML model
- Evaluation metrics indicate of how well the models *would* do when deployed
- The choice of metrics is very task-specific and determines what the model *learns*
- It is important to know what you are willing to *trade off* when training ML models for a task

Model Evaluation Metrics

- Binary Classification
- Multi-class Classification
- Regression

Evaluation Metrics for Binary Classification

Evaluation metrics for Binary Classification

- Threshold-based metrics
  - Classification Accuracy
  - Precision, Recall & F1-score
- Ranking-based metrics
  - Average Precision (AP)
  - Area Under Curve (AUC)

Classification Accuracy

$$Acc = \frac{\sum_{i=1}^{n} I_{\hat{y}_i = y_i}}{n}$$

$\hat{y}_i$ - predicted label

y_i - actual label

Confusion Matrix

		Predicted Values	
		Negative	Positive
Actual Values	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

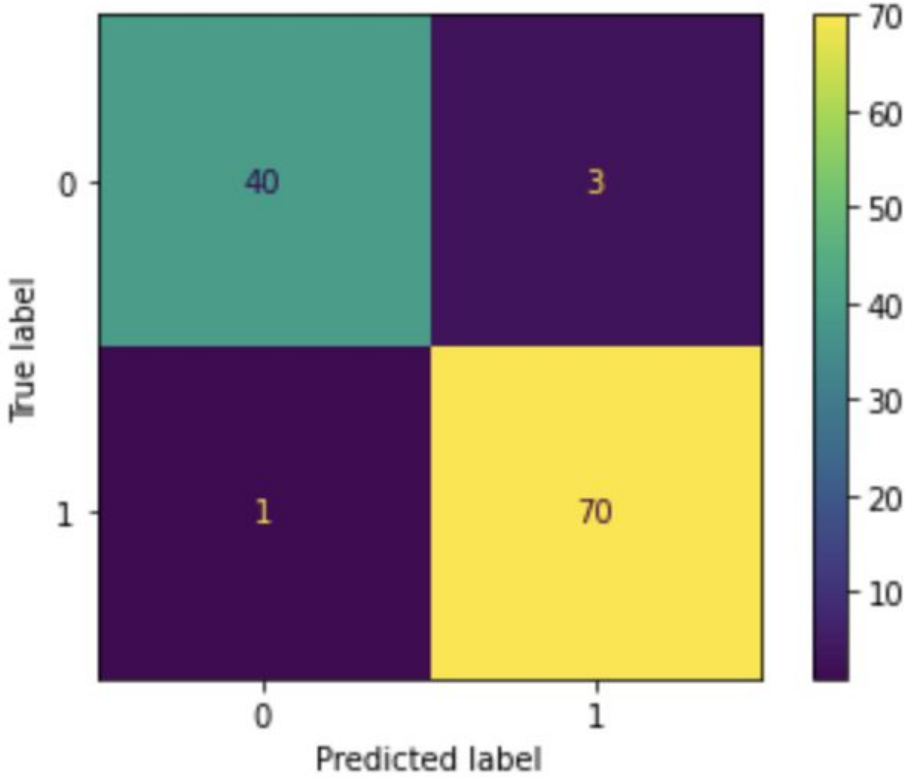
$$\textbf{Accuracy} = \frac{\textbf{TN+TP}}{\textbf{TN+TP+FP+FN}}$$

Confusion Matrix - Example

```
data, target = load_breast_cancer(return_X_y=True)
X_dev, X_test, y_dev, y_test = train_test_split(data, target,
                                                test_size=0.2,
                                                random_state=42)
model = LogisticRegression().fit(X_dev, y_dev)
pred_y = model.predict(X_test)
plot_confusion_matrix(model, X_test, y_test)
print(model.score(X_test, y_test))
```

```
0.9649122807017544
```

Confusion Matrix - Example



$$\textbf{Accuracy} = \frac{TN+TP}{TN+TP+FP+FN}$$

Accuracy = 110/114 = 0.9649

Confusion Matrix - Normalization

```
confusion_matrix(y_true,y_pred)
```

Actual Negative	True Negative	False Positive
Actual Positive	False Negative	True Positive
	Predicted Negative	Predictive Positive

```
confusion_matrix(y_true,y_pred,normalize='true')
```

```
array([[0.84, 0.16],
       [0.48, 0.52]])
```

Actual Negative	True Negative Rate Specificity	False Positive Rate Fall -Out
Actual Positive	False Negative Rate Miss-Rate	True Positive Rate Recall Sensitivity
	Predicted Negative	Predictive Positive

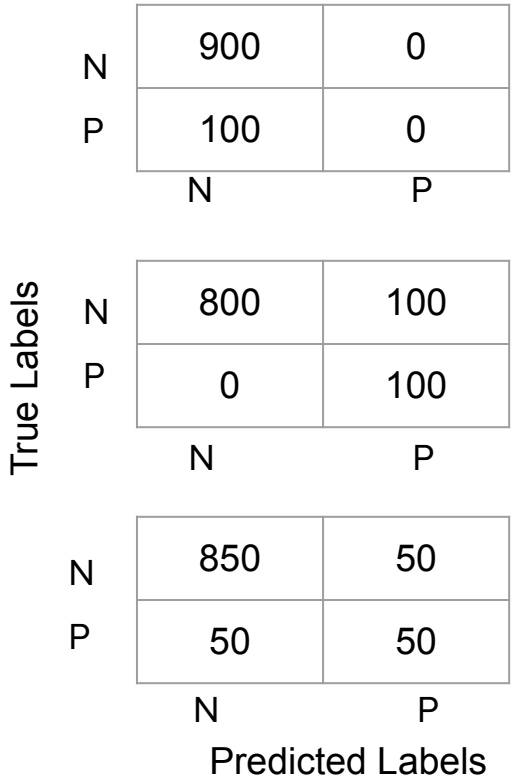
```
confusion_matrix(y_true,y_pred,normalize='pred')
```

```
array([[0.77, 0.36],
       [0.23, 0.64]])
```

Actual Negative	Negative Predictive Value	False Discovery Rate
Actual Positive	False Omission Rate	Precision Positive Predictive Value
	Predicted Negative	Predictive Positive

Classification Accuracy - Limitations

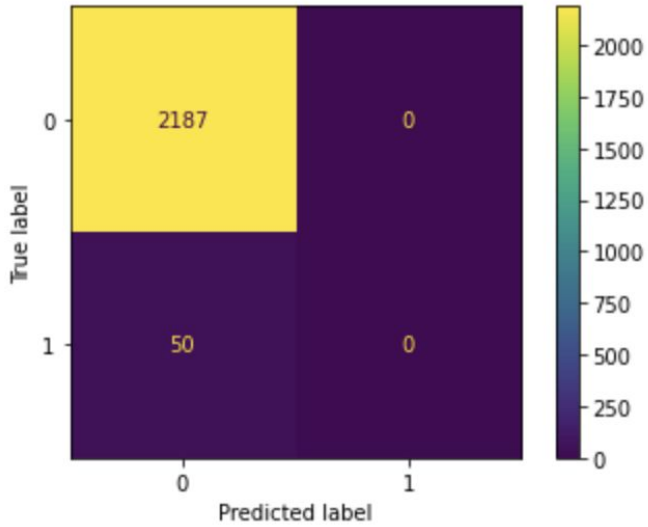
- Classification accuracy could be misleading in case of imbalance datasets
- Accuracy Paradox



Classification Accuracy - Limitations

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
print(X.shape)
print(y.value_counts())
rows = X_test.shape[0]
const_model_pred = ['-1'] * rows
cm = confusion_matrix(y_test, const_model_pred)
print(accuracy_score(y_test, const_model_pred))
disp = ConfusionMatrixDisplay(cm)
disp.plot()
```

```
(11183, 6)
-1    10923
1       260
Name: class, dtype: int64
0.9776486365668305
```



Precision, Recall & F1-score

- Precision, Recall & F1-score are better metrics for imbalance datasets
- Precision is defined as the fraction of relevant instances among retrieved instances
- Recall is defined as the fraction of relevant instances that were retrieved
- F1-score is the harmonic mean of precision & recall

Precision, Recall & F1-score

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall (TPR)} = \frac{TP}{TP+FN}$$

$$\text{F1-Score} = 2\,\frac{Precision*Recall}{Precision+Recall}$$

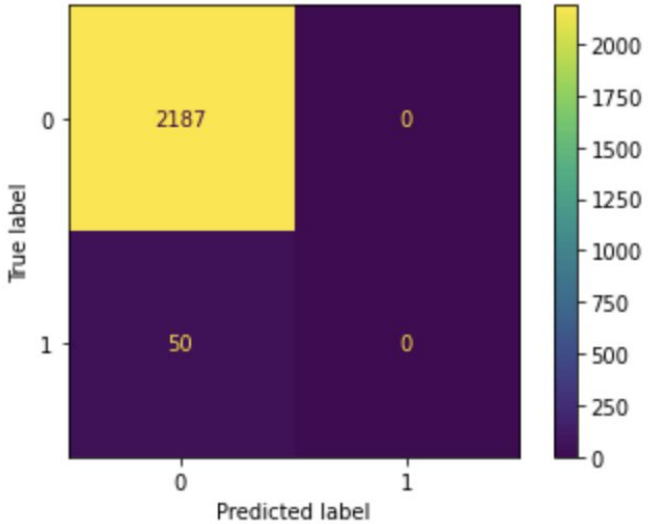
		Predicted Values	
		Negative	Positive
Actual Values	Negative	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)

Minority class is considered positive as best practice

Precision, Recall & F1-score - Example

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
print(X.shape)
print(y.value_counts())
rows = X_test.shape[0]
const_model_pred = ['-1'] * rows
print(f"Accuracy:",
      accuracy_score(y_test, const_model_pred))
print(f"Recall:",
      recall_score(y_test, const_model_pred, pos_label='1'))
print(f"Precision:",
      precision_score(y_test, const_model_pred, pos_label='1'))
print(f"F1-score:",
      f1_score(y_test, const_model_pred, pos_label='1'))
```

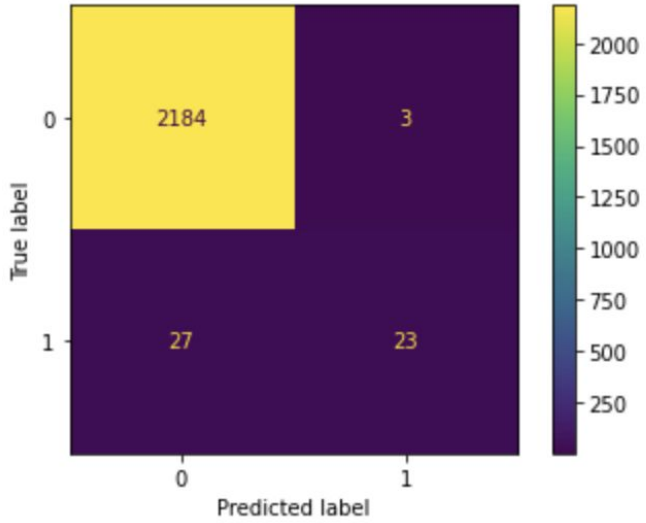
```
(11183, 6)
-1    10923
1       260
Name: class, dtype: int64
Accuracy: 0.9776486365668305
Recall: 0.0
Precision: 0.0
F1-score: 0.0
```



Precision, Recall & F1-score - Example

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
rows = X_test.shape[0]
model = LogisticRegression().fit(X_dev, y_dev)
y_pred = model.predict(X_test)
print(f"Accuracy:",
      accuracy_score(y_test, y_pred))
print(f"Recall:",
      recall_score(y_test, y_pred, pos_label='1'))
print(f"Precision:",
      precision_score(y_test, y_pred, pos_label='1'))
print(f"F1-score:",
      f1_score(y_test, y_pred, pos_label='1'))
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
```

```
Accuracy: 0.9865891819400984
Recall: 0.46
Precision: 0.8846153846153846
F1-score: 0.605263157894737
```



Averaging Metrics - Macro & Weighted

- Precision, Recall & F1-score are generally calculated (by default) for the minority class
- In case of multiple classes, metrics per class can be combined in several ways to report an average metric

$$Macro = \frac{1}{|L|} \sum_{l \in L} R\left(y_l, \hat{y}_l \right)$$

Average over recall per class

$$Weighted = \frac{1}{n} \sum_{l \in L} n_l R\left(y_l, \hat{y}_l \right)$$

Weighted (by class size) average over recall per class

Averaging Metrics - Example

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
rows = X_test.shape[0]
model = LogisticRegression().fit(X_dev, y_dev)
y_pred = model.predict(X_test)
print(f"Accuracy:",
      accuracy_score(y_test, y_pred))
print(f"Binary Recall:",
      recall_score(y_test, y_pred, pos_label='1', average="binary"))
print(f"Macro Recall:",
      recall_score(y_test, y_pred, pos_label='1', average="macro"))
print(f"Weighted Recall:",
      recall_score(y_test, y_pred, pos_label='1', average="weighted"))
```

```
Accuracy: 0.9865891819400984
Binary Recall: 0.46
Macro Recall: 0.7293141289437586
Weighted Recall: 0.9865891819400984
```

Averaging Metrics - Balanced Accuracy

- Average of recall obtained on each class (same as macro-averaged recall)
- Same as accuracy for balanced datasets

$$balanced\ accuracy = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

Averaging Metrics - Balanced Accuracy

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
rows = X_test.shape[0]
model = LogisticRegression().fit(X_dev, y_dev)
y_pred = model.predict(X_test)
print(f"Accuracy:",
      accuracy_score(y_test, y_pred))
print(f"Binary Recall:",
      recall_score(y_test, y_pred, pos_label='1', average="binary"))
print(f"Macro Recall:",
      recall_score(y_test, y_pred, pos_label='1', average="macro"))
print(f"Weighted Recall:",
      recall_score(y_test, y_pred, pos_label='1', average="weighted"))
print(f"Balanced Accuracy:",
      balanced_accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9865891819400984
Binary Recall: 0.46
Macro Recall: 0.7293141289437586
Weighted Recall: 0.9865891819400984
Balanced Accuracy: 0.7293141289437586
```

Averaging Metrics - Example

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
rows = X_test.shape[0]
model = LogisticRegression().fit(X_dev, y_dev)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          -1       0.99      1.00      0.99      2187
           1       0.88      0.46      0.61        50

    accuracy                           0.99      2237
   macro avg       0.94      0.73      0.80      2237
weighted avg       0.99      0.99      0.98      2237
```

Choosing the Right Metric

- Problem-specific
- Balanced accuracy better than accuracy (most of the times)
- Cost associated with misclassification
    - Predicting that an individual has no cancer when he/she has cancer (false negative) is far more costlier than the other way round
    - Predicting an email as spam when it is not (false positive) has higher cost than predicting email as not spam
- Choose precision when cost of false positives is high (Type I error)
- Choose recall when cost of false negatives is high (Type II error)

Choosing the Right Metric - Example

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
rows = X_test.shape[0]
model = LogisticRegression().fit(X_dev, y_dev)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          -1       0.99      1.00      0.99      2187
           1       0.88      0.46      0.61        50

    accuracy                           0.99      2237
   macro avg       0.94      0.73      0.80      2237
weighted avg       0.99      0.99      0.98      2237
```

Choosing the Right Metric - Example

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
rows = X_test.shape[0]
model = LogisticRegression().fit(X_dev, y_dev)
y_pred = model.predict_proba(X_test)[:, 1] >= 0.3
print(classification_report(y_test == '1', y_pred))
```

	precision	recall	f1-score	support
False	0.99	1.00	0.99	2187
True	0.74	0.56	0.64	50
accuracy			0.99	2237
macro avg	0.86	0.78	0.81	2237
weighted avg	0.98	0.99	0.98	2237

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
rows = X_test.shape[0]
model = LogisticRegression().fit(X_dev, y_dev)
y_pred = model.predict_proba(X_test)[:, 1] >= 0.7
print(classification_report(y_test == '1', y_pred))
```

	precision	recall	f1-score	support
False	0.98	1.00	0.99	2187
True	1.00	0.28	0.44	50
accuracy			0.98	2237
macro avg	0.99	0.64	0.71	2237
weighted avg	0.98	0.98	0.98	2237

Precision-Recall (PR) Curve

- A precision-recall curve shows the relationship between precision and recall at every cut-off point.
- Visualize effect of selected threshold on performance.

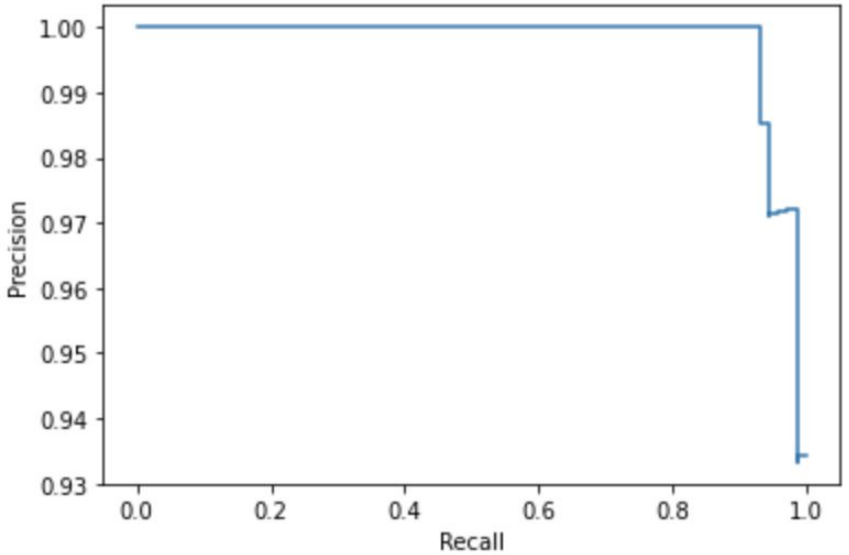
PR Curve - Example

```
data, target = load_breast_cancer(return_X_y=True)
X_dev, X_test, y_dev, y_test = train_test_split(data, target,
                                                test_size=0.2,
                                                random_state=42)
model = LogisticRegression().fit(X_dev, y_dev)
y_pred_prob = model.predict_proba(X_test)[:, 1]
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob,
                                                       pos_label = 1)
disp = PrecisionRecallDisplay(precision=precision, recall=recall)
disp.plot()
```

PR Curve - Example

```
thresholds
```

```
array([0.14860909, 0.39616821, 0.41720957, 0.65787024, 0.69361918,
       0.80632541, 0.81114673, 0.86164509, 0.86360292, 0.8742218 ,
       0.89569695, 0.91343602, 0.92476118, 0.93965188, 0.94186253,
       0.94490585, 0.9487397 , 0.95145943, 0.95709663, 0.96268462,
       0.9630544 , 0.96596123, 0.96857305, 0.9738155 , 0.97411169,
       0.97603704, 0.98164287, 0.98257776, 0.98306774, 0.98436737,
       0.98554496, 0.98599898, 0.98657668, 0.98733158, 0.98756045,
       0.98830699, 0.98908157, 0.98949783, 0.98985665, 0.99080722,
       0.99092516, 0.99137281, 0.99272312, 0.992795  , 0.99294971,
       0.99353043, 0.99466452, 0.99520786, 0.99523613, 0.99548719,
       0.99564478, 0.99578087, 0.99638044, 0.99646681, 0.99653845,
       0.99726527, 0.99728913, 0.9972898 , 0.99730236, 0.99751813,
       0.99753612, 0.99757115, 0.99770176, 0.99778441, 0.99804583,
       0.99817105, 0.99839012, 0.99879504, 0.99881537, 0.99921133,
       0.99925662, 0.99945534, 0.99963201, 0.99967188, 0.99967857,
       0.99983227])
```



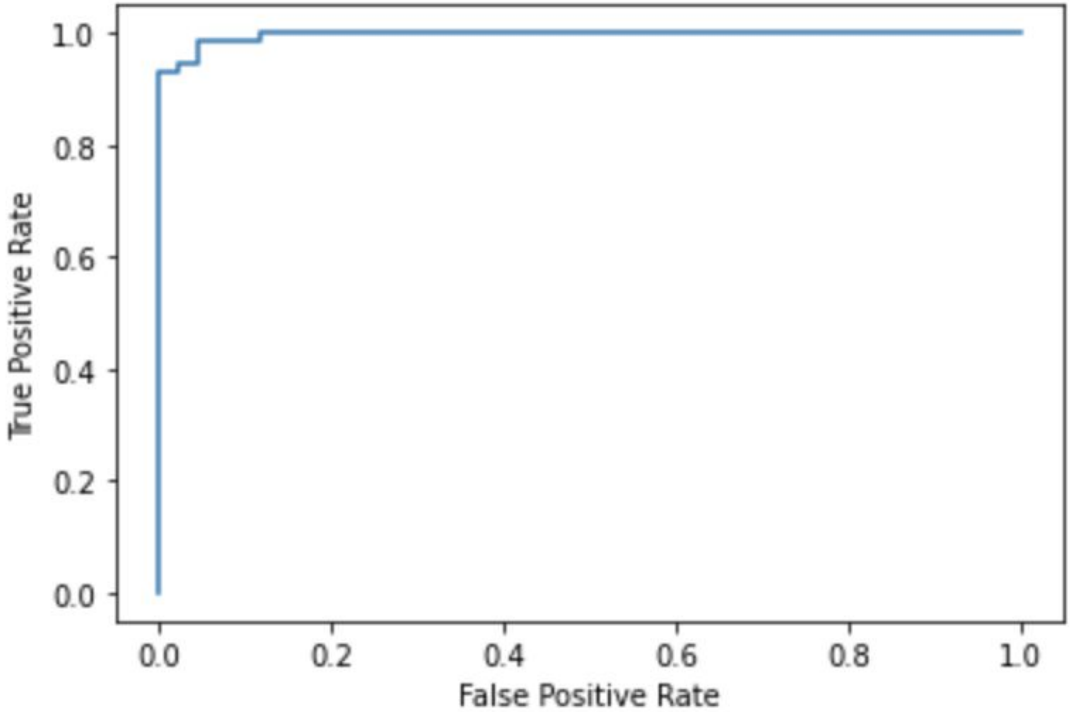
Receiver Operating Curve (ROC)

- Another useful tool to visualize the performance of a classification model
- ROC depicts the relationship between False Positive Rate (FPR) and True Positive Rate/Recall (TPR)

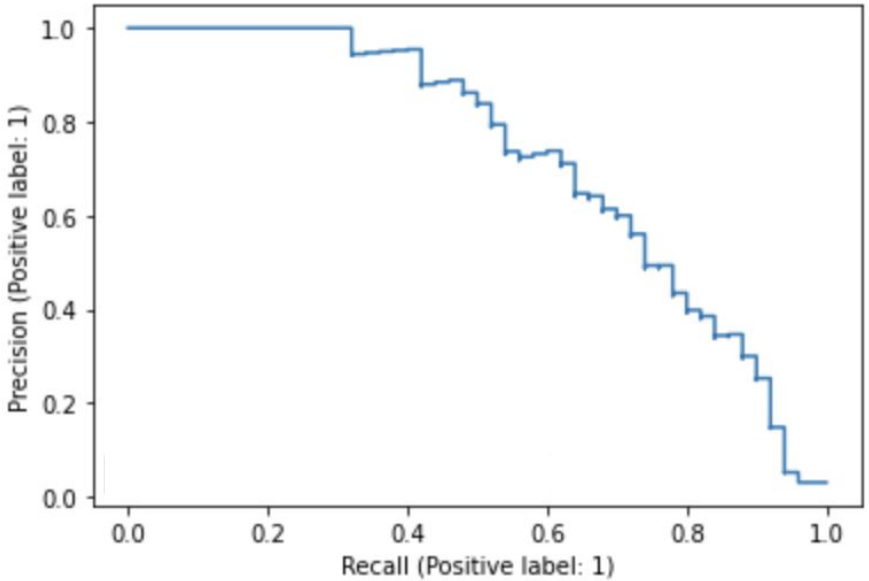
Receiver Operating Curve (ROC) - Example

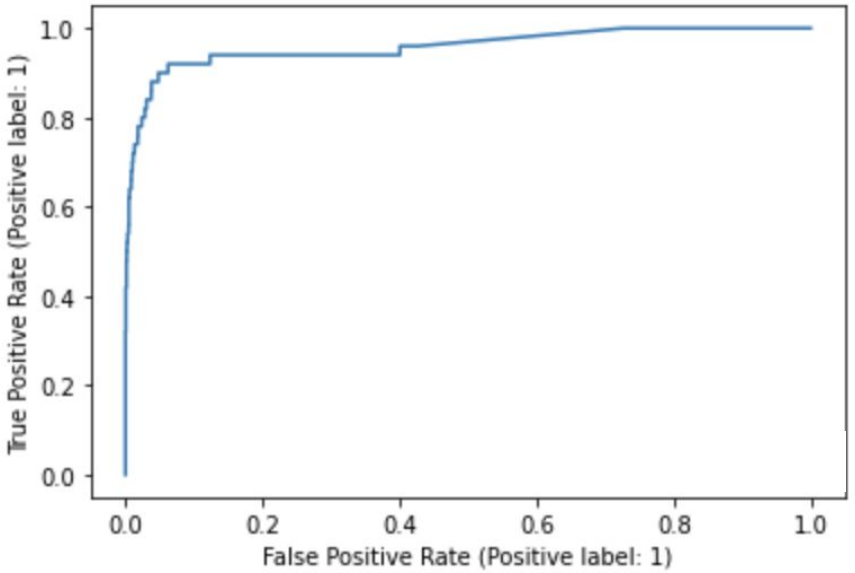
```
data, target = load_breast_cancer(return_X_y=True)
X_dev, X_test, y_dev, y_test = train_test_split(data, target,
                                                test_size=0.2,
                                                random_state=42)
model = LogisticRegression().fit(X_dev, y_dev)
y_pred_prob = model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob, pos_label=1)
disp = RocCurveDisplay(fpr=fpr, tpr=tpr)
disp.plot()
```

Receiver Operating Curve (ROC) - Example



PR Curve v.s. ROC





$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{FPR} = \frac{FP}{TN+FP}$$

PR Curve v.s. ROC

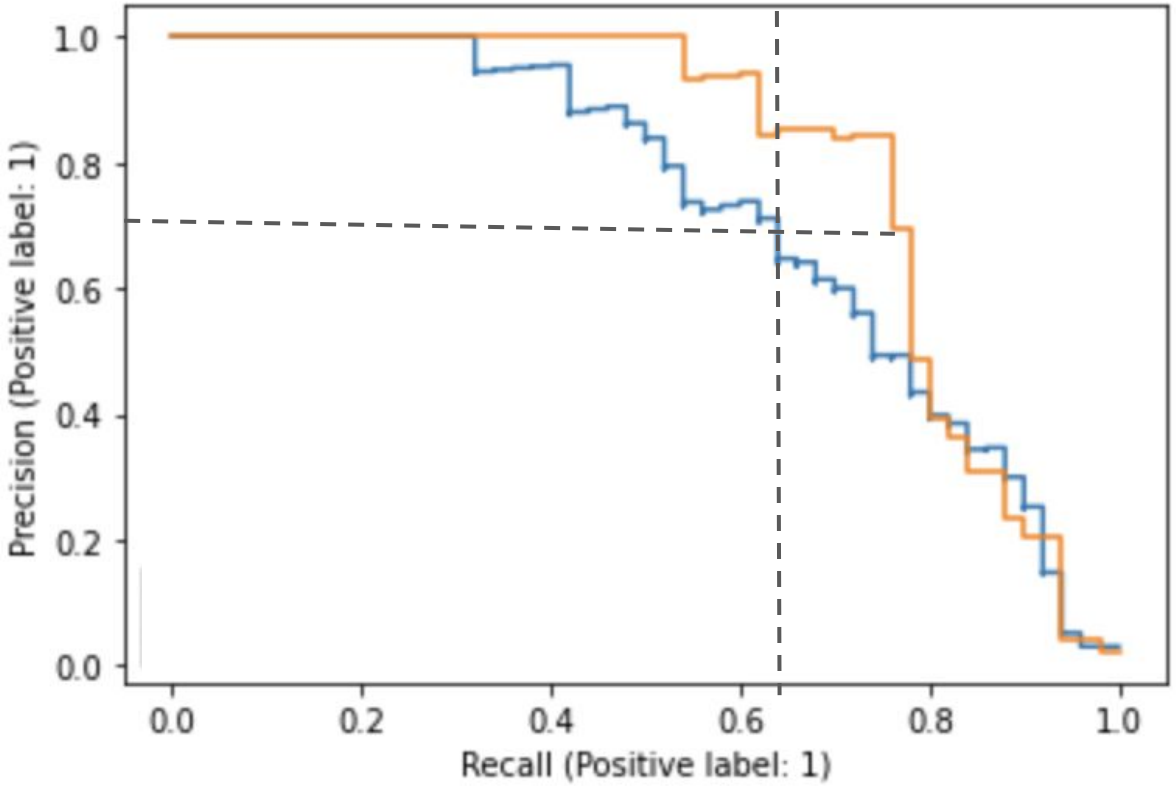
		Actual: TRUE +, 1	Actual: FALSE -, 0
Predicted	TRUE +, 1	100 (TP)	1000 (FP)
Predicted	FALSE -, 0	100 (FN)	10000 (TN)
Class count		P = 200	N = 11000

$$Recall = \frac{100}{200} = 0.5$$

$$FPR = \frac{1000}{11000} = 0.09$$

$$Precision = \frac{100}{1100} = 0.09$$

Comparing models using PR-curve



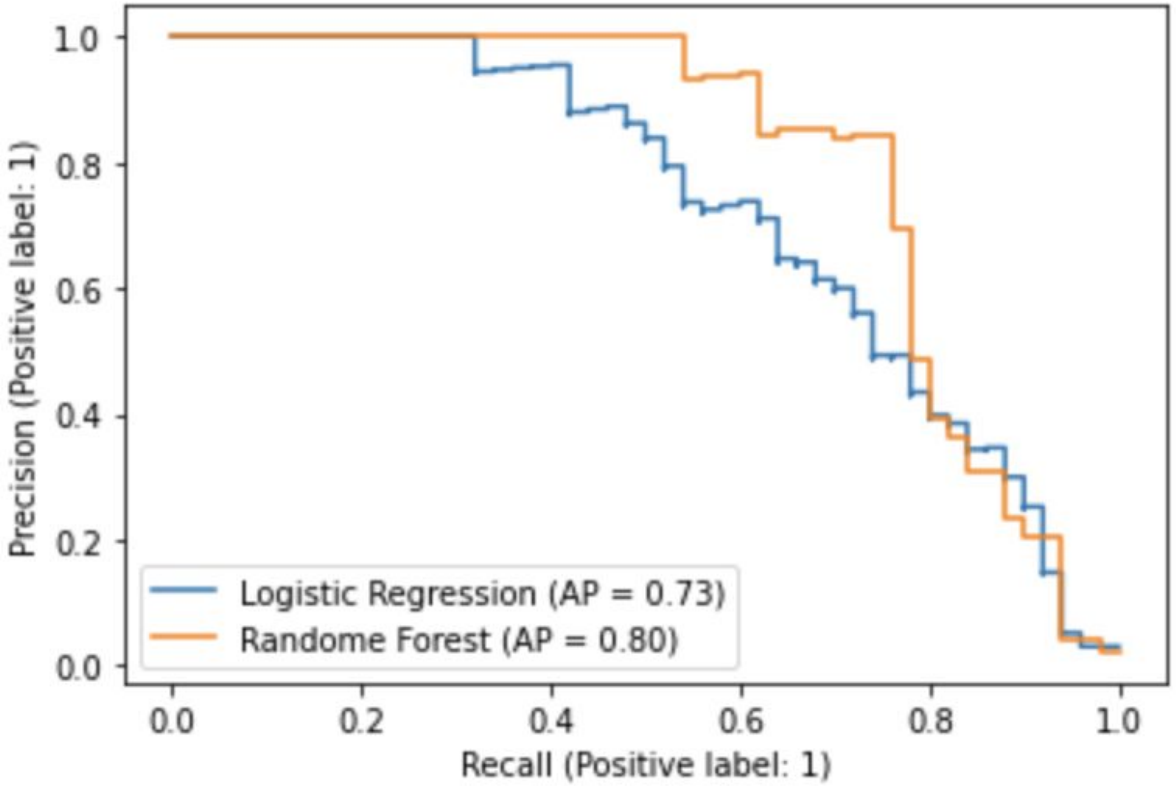
Average Precision (AP)

$$AveP = \sum_{k=1}^{n} P(k)\ \Delta\ r(k)$$

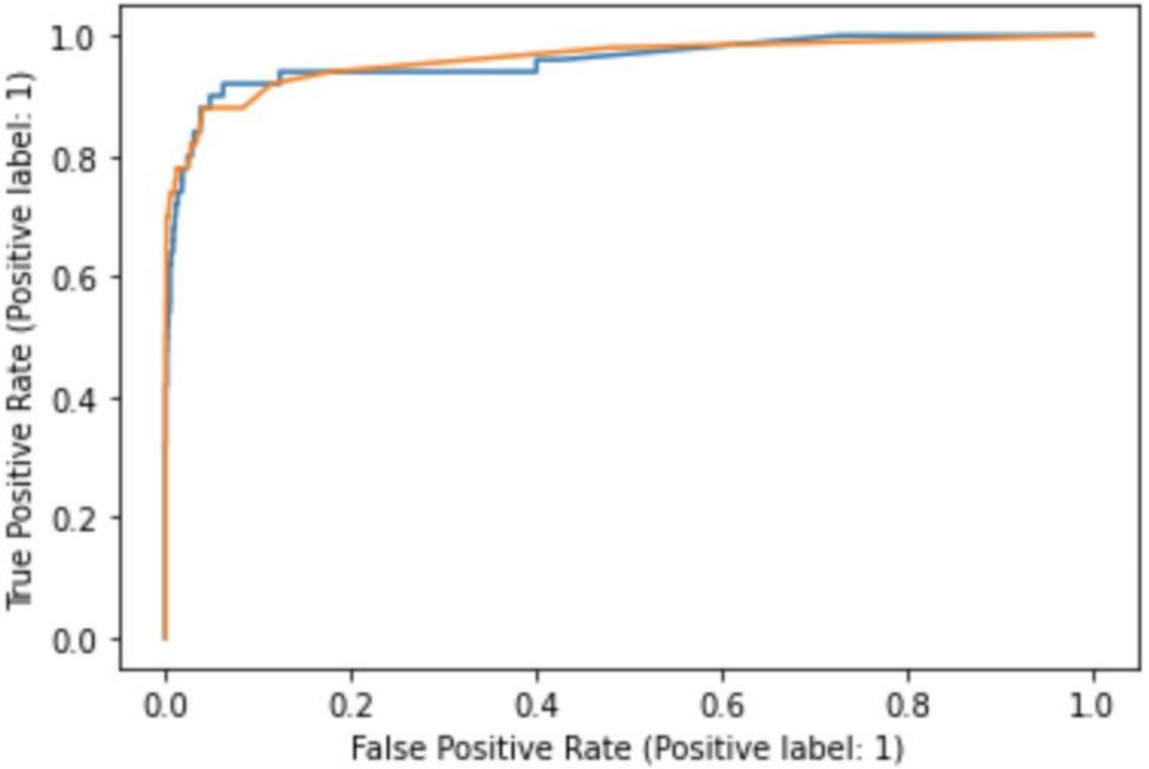
$$P(k)\ -\ precision\ at\ cutoff\ k$$

$$\Delta\ r(k) - change\ in\ recall\ from\ items\ k-1\ to\ k$$

Comparing models using AP



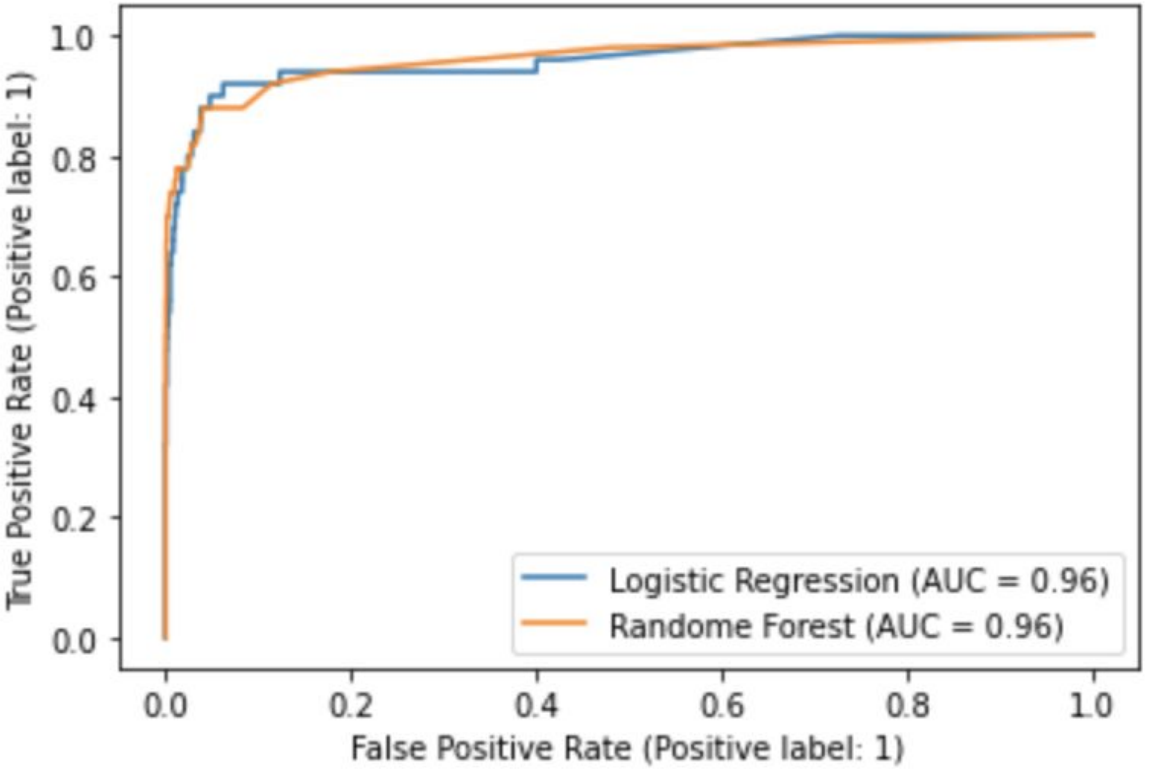
Comparing models using ROC-curve



Area Under ROC (AUROC)

- Area Under ROC (AUROC) provides an aggregate measure of model performance across all possible classification thresholds.
- AUROC varies between 0 and 1 and a model with random/const predictions has a value of 0.5.

Comparing models using AUROC



AP v.s. AUROC

- AP & AUROC are both ranking metrics
- AP indicates whether your model can correctly identify all positive examples without accidentally marking too many negative examples as positive
- AUROC measures whether the model is able to rank positive examples higher than negative samples
- It is *easier* to know how well the model is performing than random using AUROC than AP
- In case of imbalance datasets, AP is a better estimate indicative of model performance

Evaluation Metrics for Multi-class Classification

Evaluation metrics for Multi-class Classification

- Threshold-based metrics
  - Classification Accuracy
  - Precision, Recall & F1-score (macro & weighted)
- Ranking-based metrics
  - Average Precision (AP)
  - Area Under Curve (AUC)

Evaluation metrics for Multi-class Classification - AUROC

Hand & Till, 2001, one vs one

$$\frac{1}{c(c-1)} \sum_{j=1}^{c} \sum_{k \neq j}^{c} AUC(j,k)$$

Provost & Domingo, 2000, one vs rest

$$\frac{1}{c} \sum_{j=1}^{c} p(j) AUC(j, \text{rest}_j)$$

Evaluation Metrics for Regression

Evaluation Metrics for Regression

R^2 (Coefficient of determination) : $1 - \left(\frac{Unexplained\ Variation}{Total\ Variation} \right)$ Easy to understand scale of 0 to 1

MSE (Mean Squared Error): $\frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$ Easy to relate to input

MAE (Mean Absolute Error): $\frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$ Used to limit impact of outliers

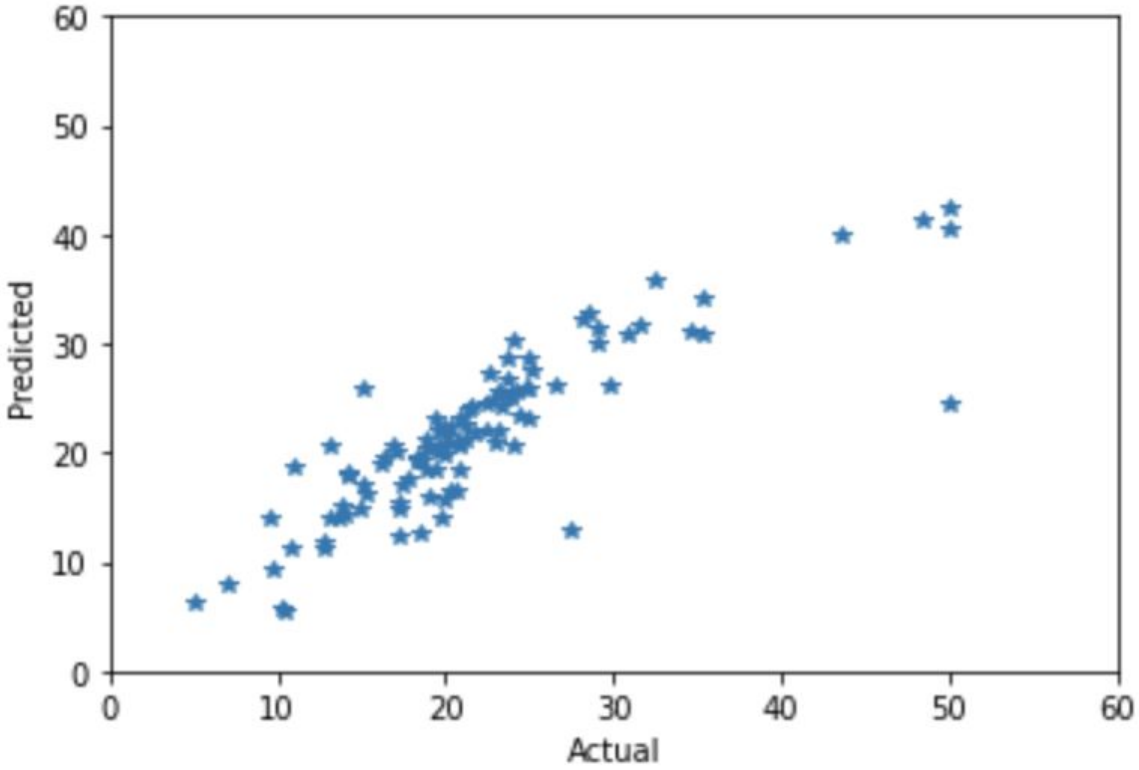
Mean Absolute Percentage Error (MAPE): $\frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$ Error relative to the value

Evaluation Metrics for Regression - Example

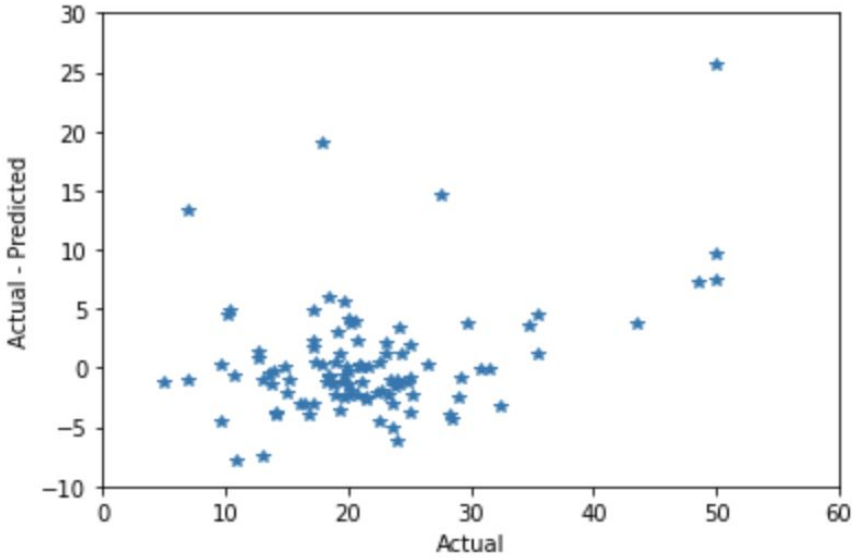
```
data = load_boston()
X_dev, X_test, y_dev, y_test = train_test_split(data.data, data.target,
                                                test_size=0.2,
                                                random_state=42)
model = Ridge().fit(X_dev, y_dev)
y_pred = model.predict(X_test)
print(f"R^2:", model.score(X_test, y_test))
print(f"MSE:", mean_squared_error(y_test, y_pred))
print(f"MAE:", mean_absolute_error(y_test, y_pred))
print(f"MAPE:", mean_absolute_percentage_error(y_test, y_pred))
```

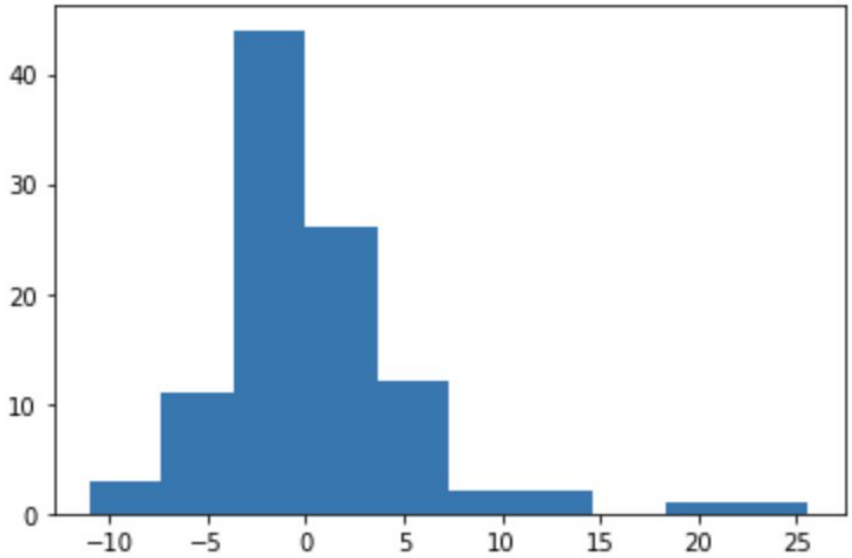
```
R^2: 0.6662221670168521
MSE: 24.477191227708662
MAE: 3.132947427805527
MAPE: 0.1660769516722134
```

Evaluation Metrics for Regression - Example



Evaluation Metrics for Regression - Example





Questions?

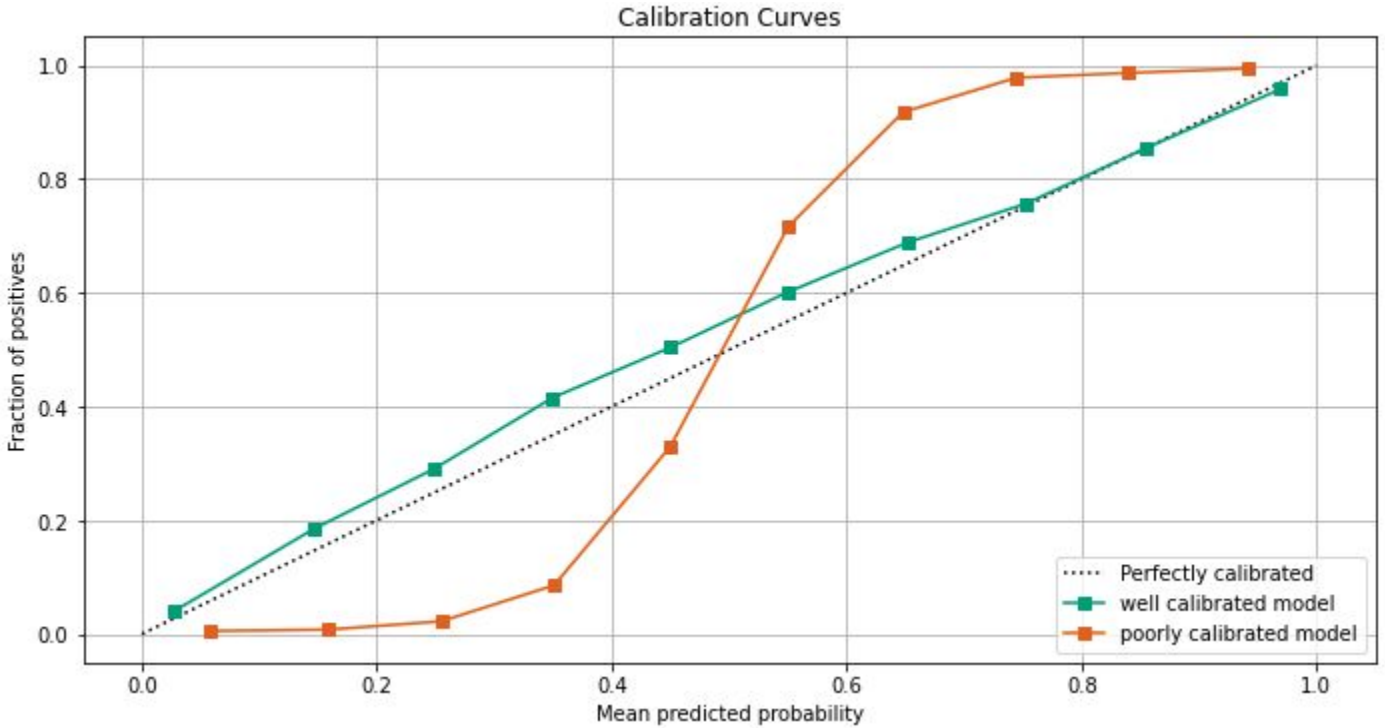
Let’s take a 10 min break!

Calibration

Motivation

- Many supervised learning algorithms predict probabilities (as a precursor to predicted labels)
- In many classification tasks, the **probability** of belonging to a class is as important as the predicted label itself
- However, we need these probabilities to be well ***calibrated***
- Calibrated probabilities means that the probability reflects the likelihood of true events

Calibration Curve (Reliability Diagram)



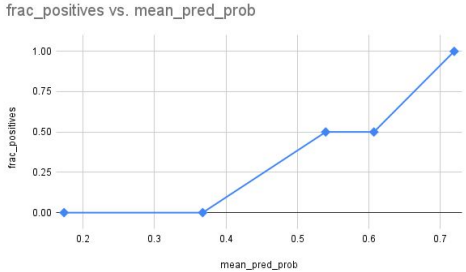
Calibration Curve (Reliability Diagram)

y_pred	y_true
0.69023332	1
0.61398155	0
0.74803338	1
0.54696659	1
0.12294483	0
0.53193647	0
0.3505869	0
0.5997642	1
0.384394	0
0.22423849	0

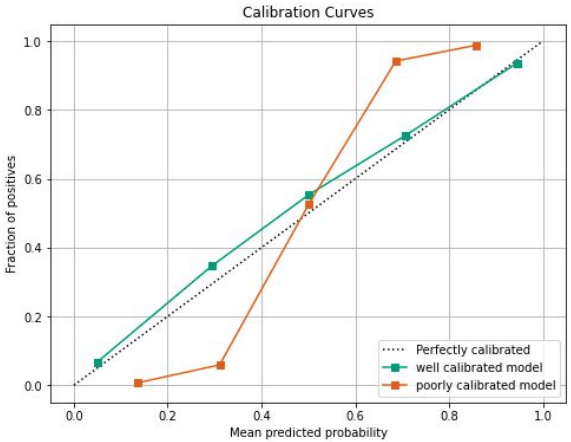
Sort by **y_pred**

y_pred	y_true	
0.12294483	0	Bin 1
0.22423849	0	
0.3505869	0	Bin 2
0.384394	0	
0.53193647	0	Bin 3
0.54696659	1	
0.5997642	1	Bin 4
0.61398155	0	
0.69023332	1	Bin 5
0.74803338	1	

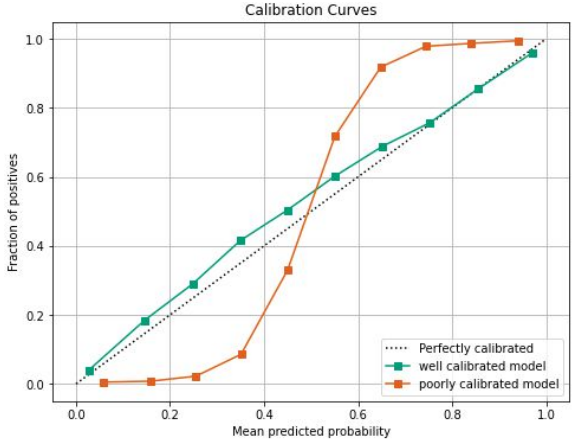
mean_pred_prob	frac_pos
0.17359166	0
0.36749045	0
0.53945153	0.5
0.606872875	0.5
0.71913335	1



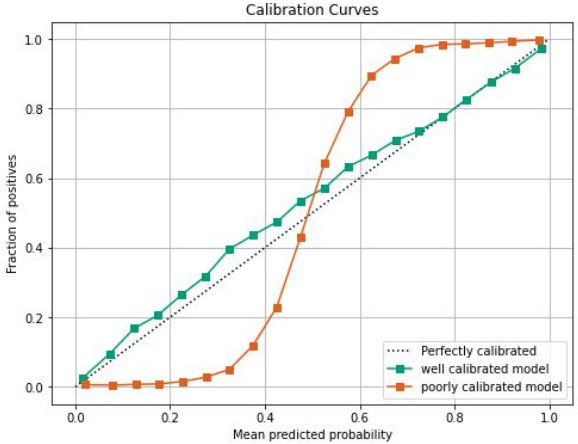
Calibration Curve - choosing # of bins



bins=5

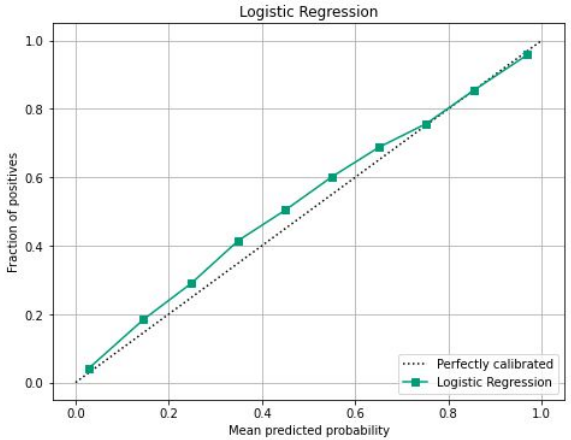


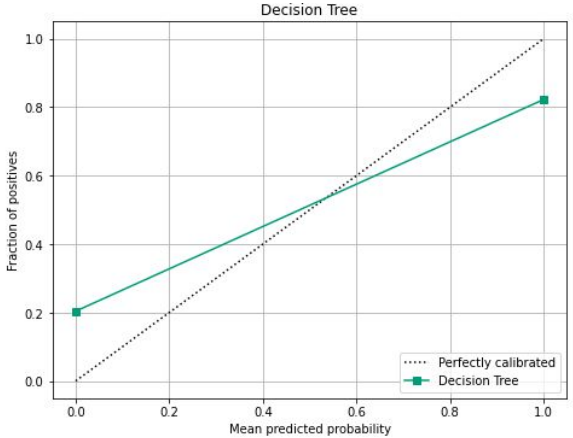
bins=10

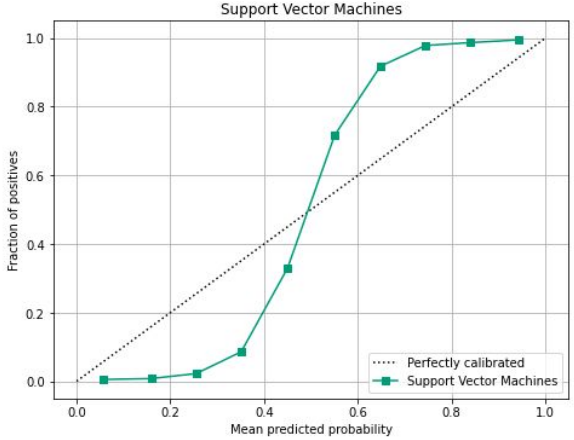


bins=20

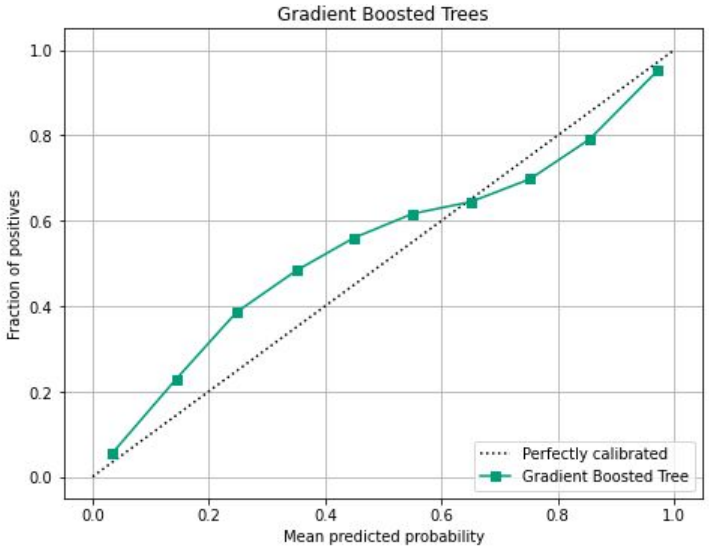
Calibration Curves - Models

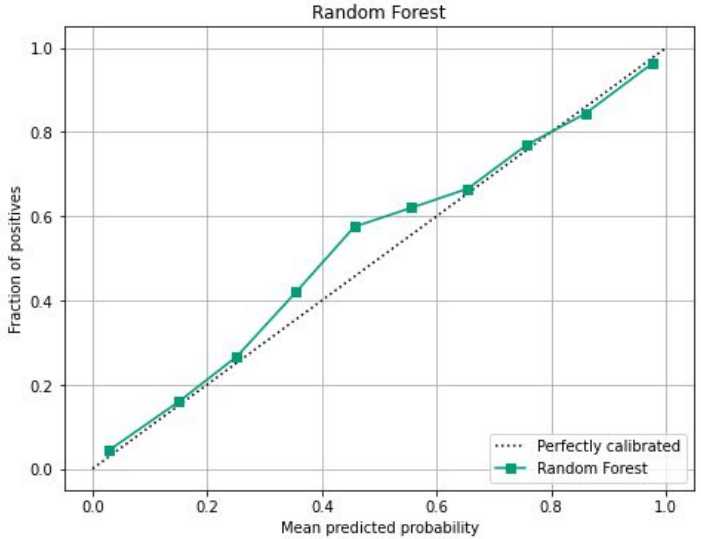






Calibration Curves - Models

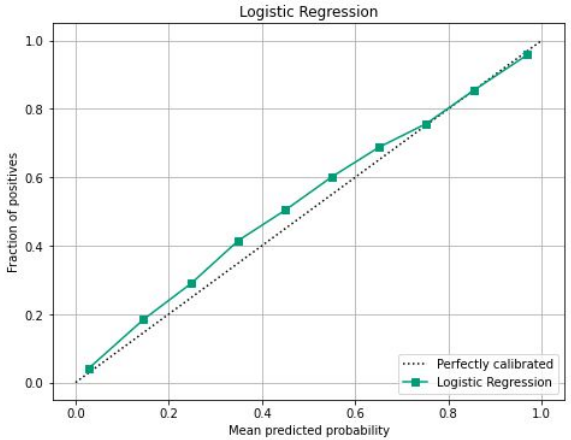




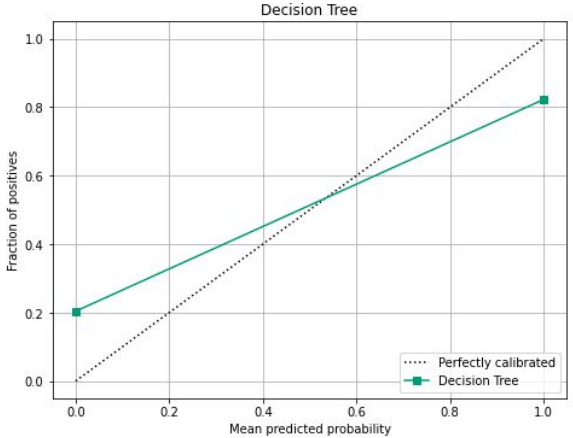
Calibration Curves - Brier Score

$$Brier\ Score\ (BS) = \frac{\sum_{i}^{n} \left(p\left(y_i \right) - y_i \right)^2}{n}$$

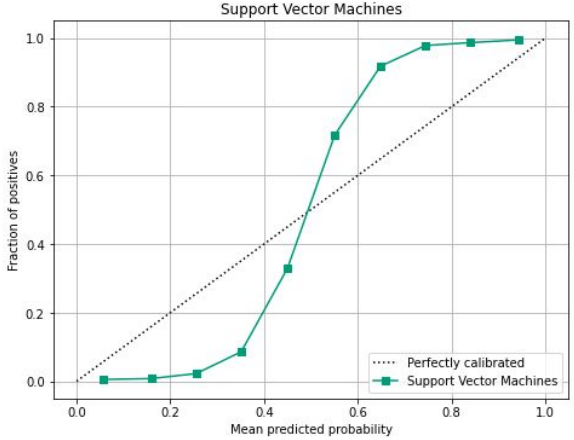
Calibration Curves - Brier Score



BS=0.09892

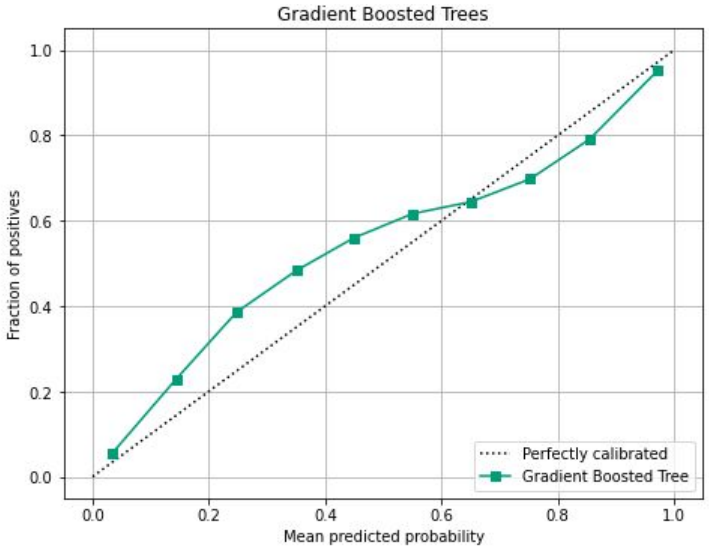


BS=0.19538

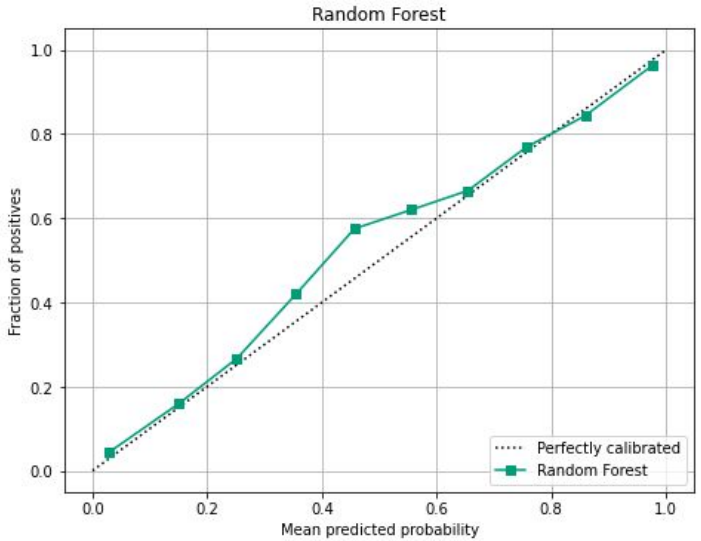


BS=0.14494

Calibration Curves - Brier Score



BS=0.10229



BS=0.09828

Calibrated Classifiers

- A calibrated classifier trains another classifier that maps model probabilities to better probabilities
- It also works on model scores (and not just probabilities)
- Typically, we are training 1D-model - $f\left(m(x)\right) \approx p(y)$
- m(x) is the model score
- Parametric/non-parametric form for f
- Training data for the calibrated classifier should be shown carefully.

Calibrated Classifiers

- Platt Scaling
- Isotonic Regression

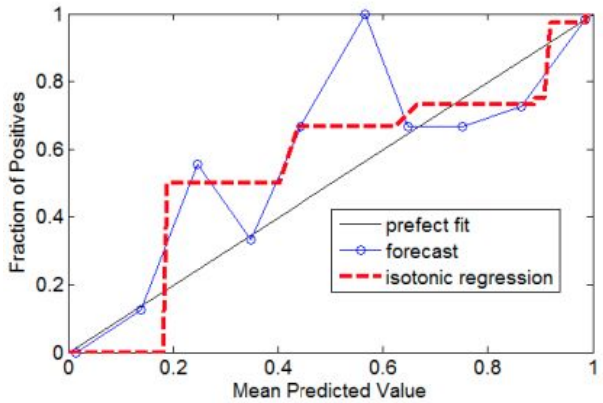
Platt Scaling

- Assume a parametric form (i.e. logistic sigmoid) for function f
- We are just learning 1-d logistic regression
- Has been applied successfully to SVMs (in theory should work for other classifiers as well)

$$f_{platt} = \frac{1}{1 + exp(-wm(x) - b)}$$

Isotonic Regression

- Assume a non-parametric form for function f
- Way more flexible than Platt Scaling
- Learns 1d step functions that approximate monotonically increasing probabilities/scores



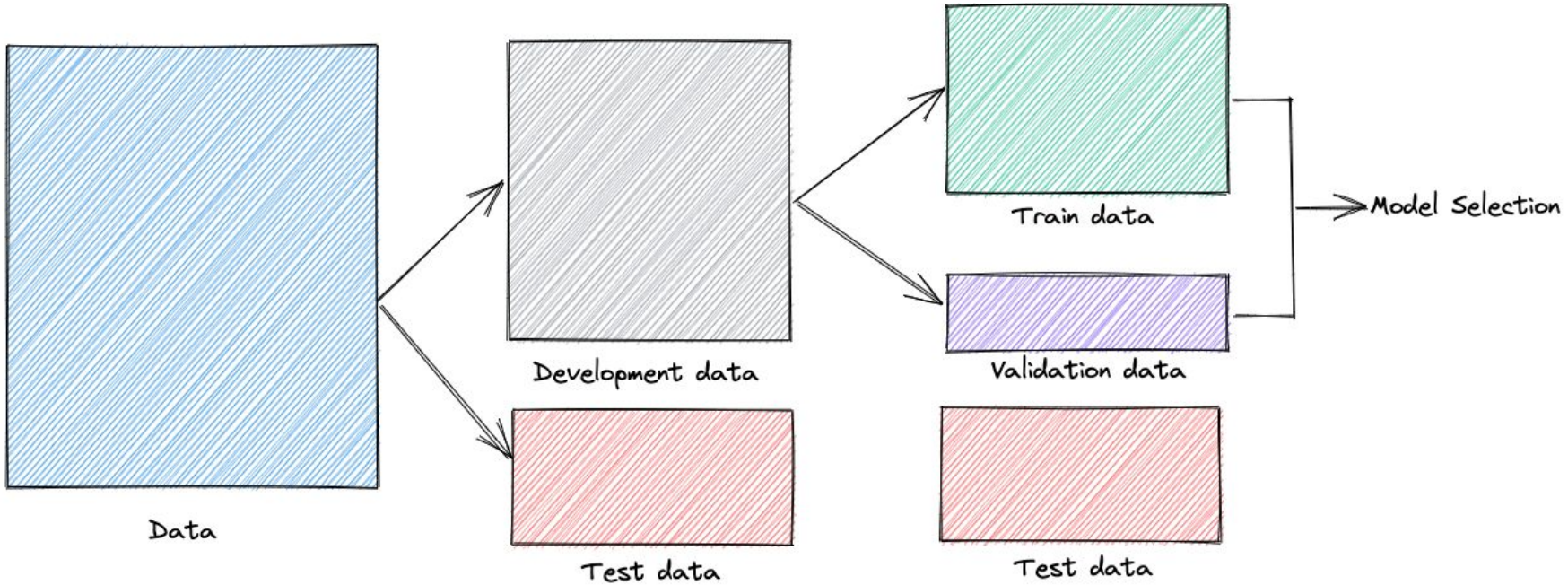
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3248752/pdf/16-cri_summit_2011.pdf

Training Calibrated Classifier

- Using training data (again) is a bad idea!
- Use holdout data to train.
- We can use cross-validation (again) to train the model

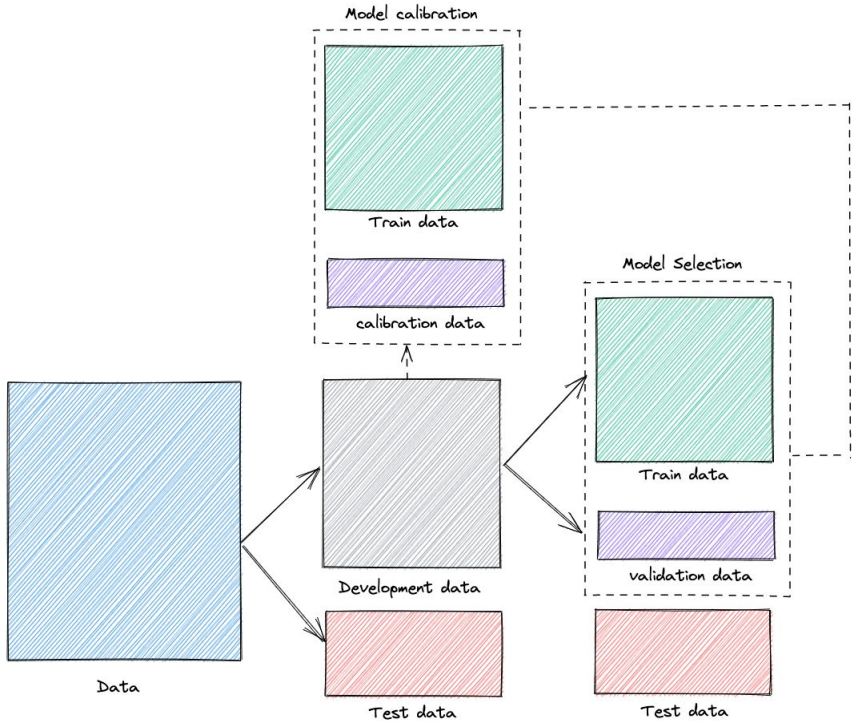
Three-way holdout

- Model selection corresponds to the hyperparameter with the best performance on validation data.



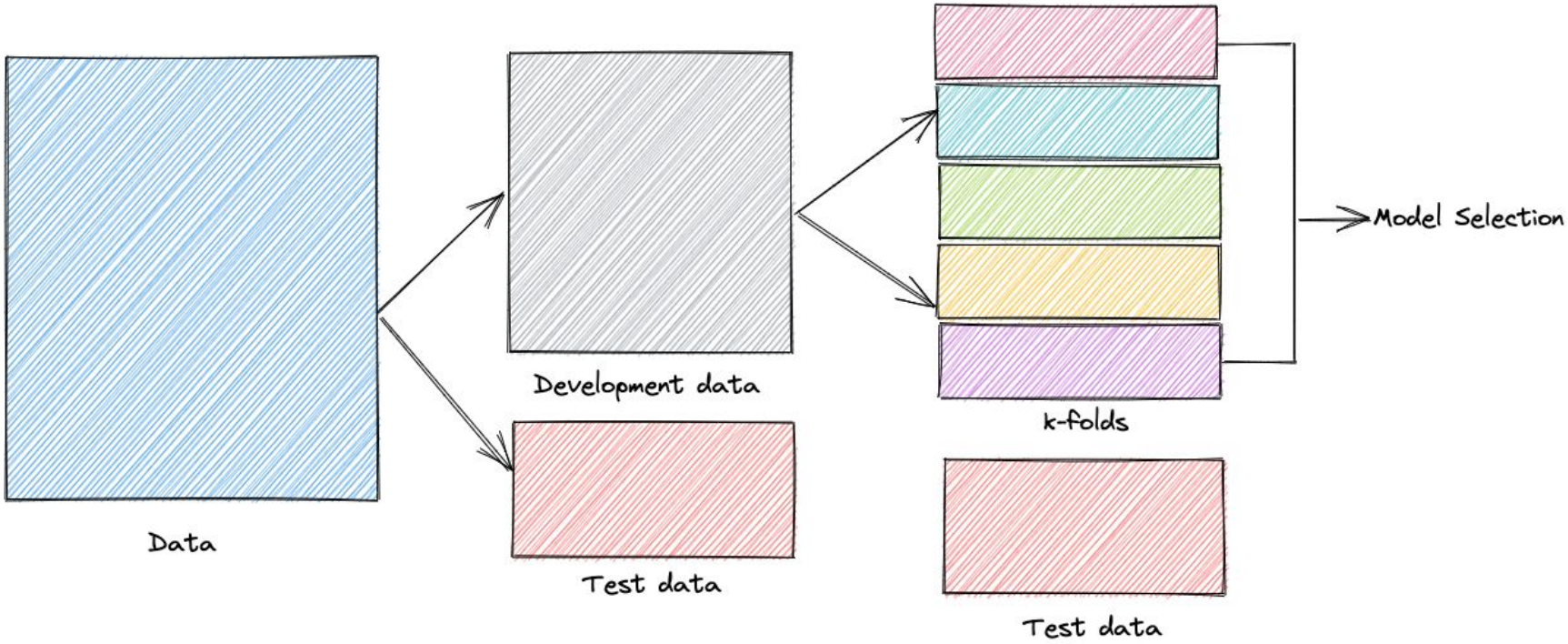
Training Calibrated Classifier - Three-way holdout

- Split the data into training data & calibration data
- Train classifier using training data with optimal hyperparameters
- Predict probabilities/scores using trained model on calibration data
- Use predicted probabilities and labels from calibration data to train the calibration model



K-fold Cross Validation

- Model selection refers to optimal hyperparameters with maximum cross validated score

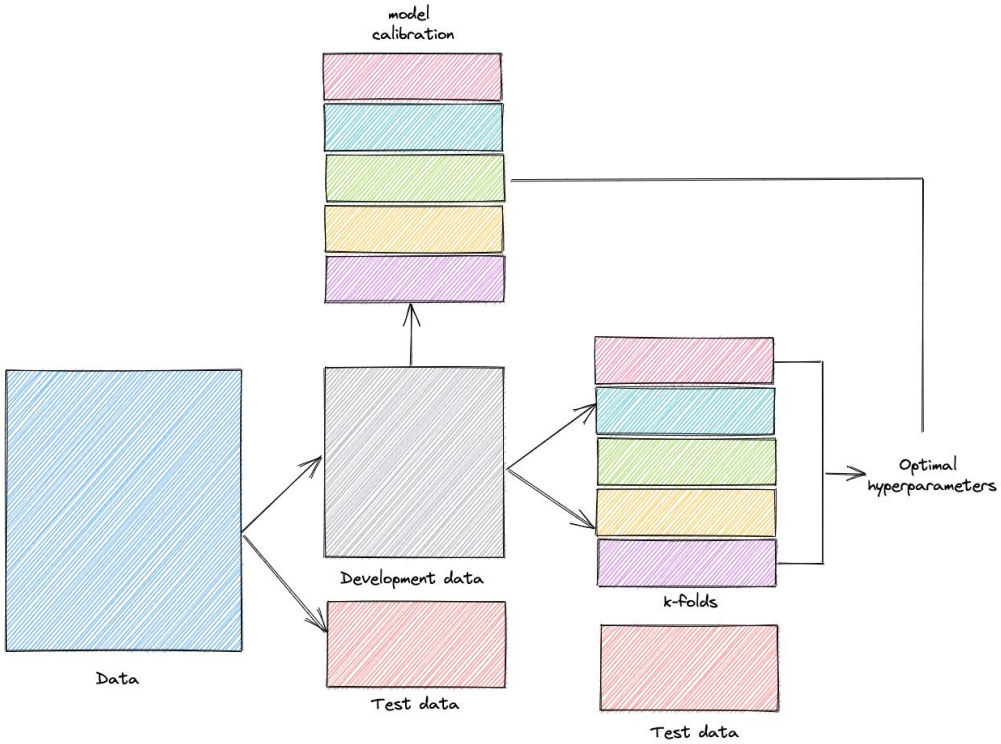


Training Calibrated Classifier - K-fold Cross Validation

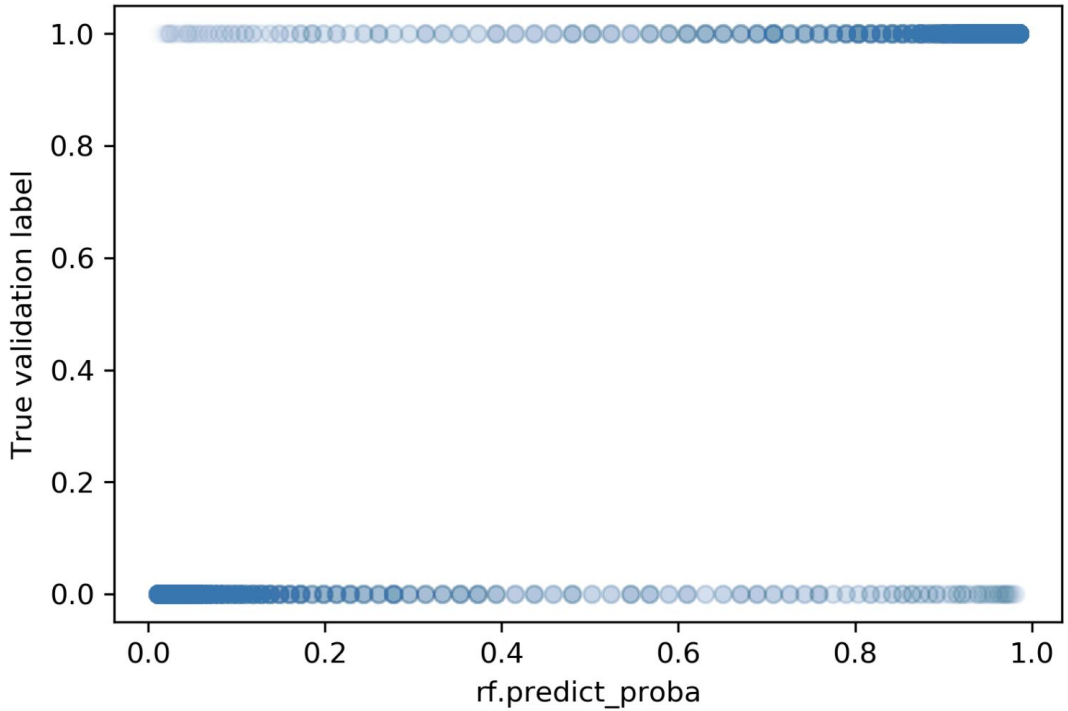
- There are several ways to train a calibrated classifier using cross-validation strategy
- We want to make sure the data used to train the model is “different” from data for training the calibration model
- In the case model selection is already done (i.e. optimal hyperparameters are found), we could use cross-validation to obtain “unbiased” probability estimates for model calibration training.
- The training process becomes messy if we combine hyperparameter tuning & model calibration (but probably will give a better performing & calibrated model)
    - May not be necessary with some validation metrics (ranking metric like AUC)

Training Calibrated Classifier - K-fold Cross Validation

- Easier way to train the calibrated classifier
- The base estimator refers to the model trained with optimal hyperparameters
- The k-fold splits are used to obtain ***unbiased*** probabilities for the training set using the base estimator
- The unbiased probabilities are then used to train the calibrated classifier

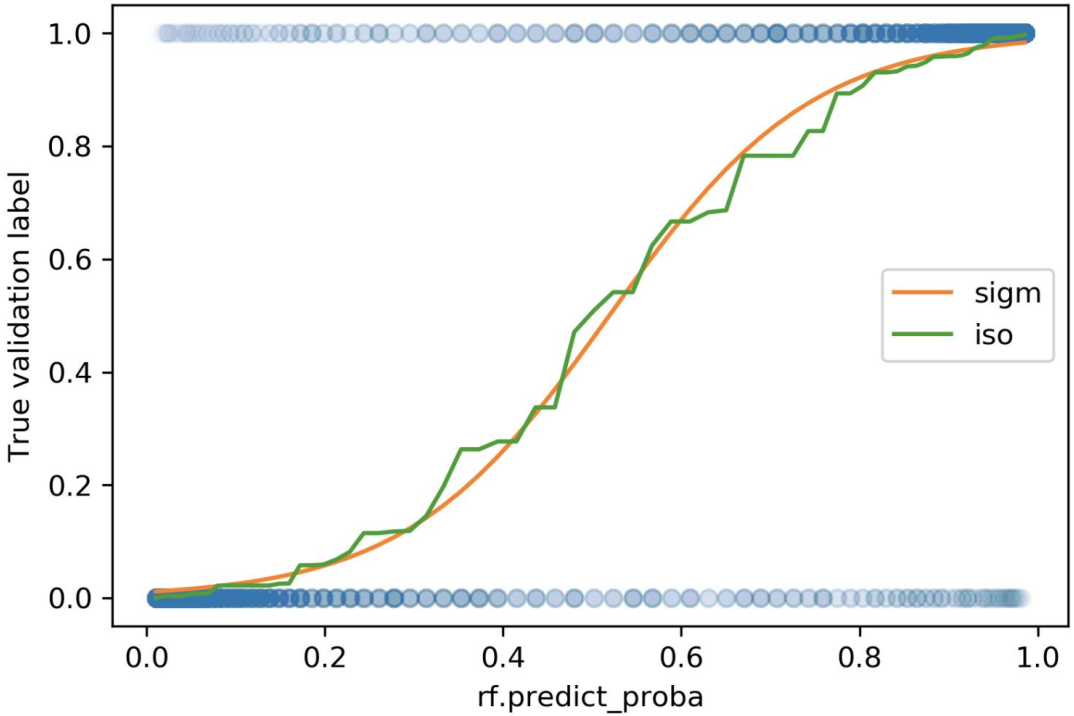


Training Calibrated Classifier



https://amueller.github.io/COMS4995-s20/slides/aml-10-calibration-imbalanced-data/#12

Training Calibrated Classifier



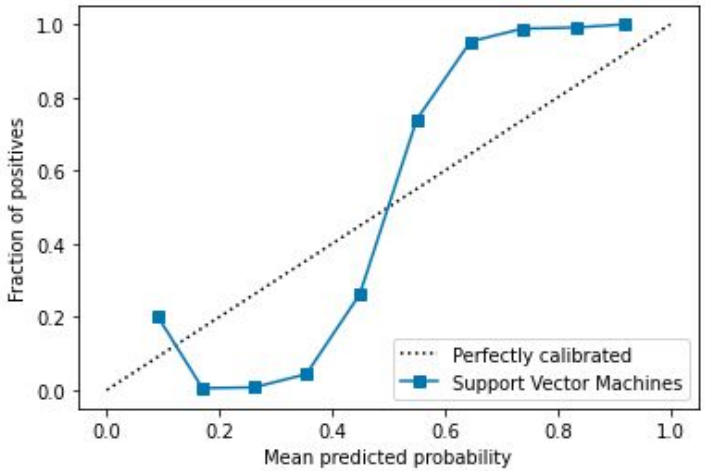
https://amueller.github.io/COMS4995-s20/slides/aml-10-calibration-imbalanced-data/#13

Training Calibrated Classifier - Example

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

X, y = make_classification(n_samples=100000, n_features=20, n_informative=2,
                           n_redundant=10, random_state=42)

X_dev, X_test, y_dev, y_test = train_test_split(X, y, test_size=0.2,
                                                random_state=42)
X_train, X_calib, y_train, y_calib = train_test_split(X_dev, y_dev, test_size=0.2,
                                                random_state=42)
```

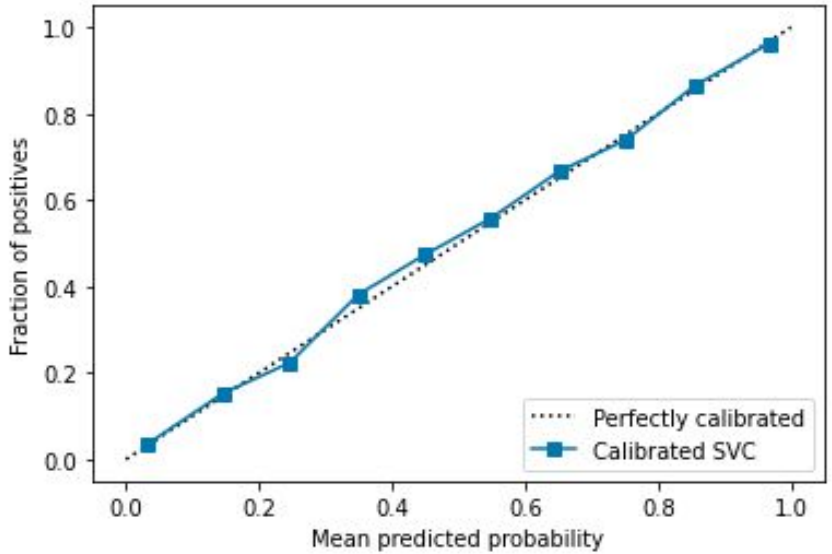
Training Calibrated Classifier - Example

```python
class NaivelyCalibratedLinearSVC(LinearSVC):
    """LinearSVC with `predict_proba` method that naively scales
    `decision_function` output for binary classification."""

    def fit(self, X, y):
        super().fit(X, y)
        df = self.decision_function(X)
        self.df_min_ = df.min()
        self.df_max_ = df.max()

    def predict_proba(self, X):
        """Min-max scale output of `decision_function` to [0, 1]."""
        df = self.decision_function(X)
        calibrated_df = (df - self.df_min_) / (self.df_max_ - self.df_min_)
        proba_pos_class = np.clip(calibrated_df, 0, 1)
        proba_neg_class = 1 - proba_pos_class
        proba = np.c_[proba_neg_class, proba_pos_class]
        return proba

svc = NaivelyCalibratedLinearSVC()
svc.fit(X_train, y_train)
display = CalibrationDisplay.from_estimator(
    svc, X_test, y_test, n_bins=10, name='Support Vector Machines')
```



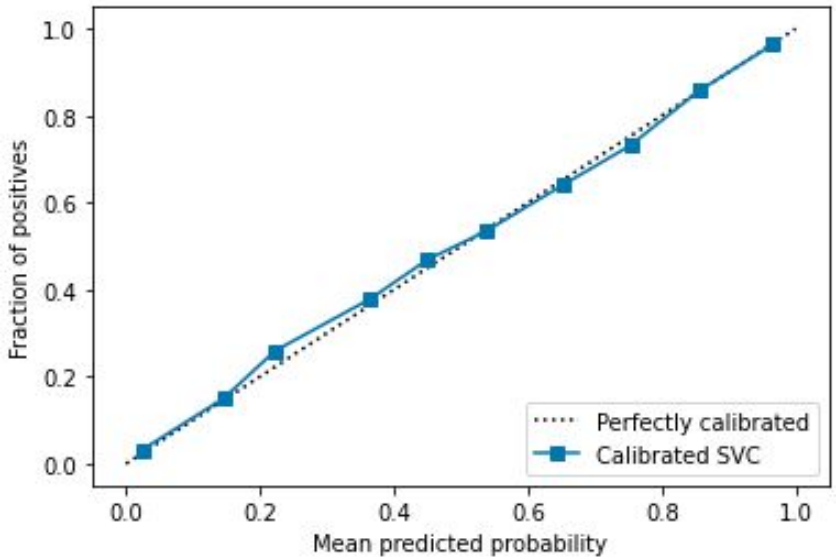
Training Calibrated Classifier - Platt Scaling

```
cal_svc = CalibratedClassifierCV(svc, cv="prefit", method="sigmoid")
cal_svc.fit(X_calib, y_calib)
display = CalibrationDisplay.from_estimator(
    cal_svc, X_test, y_test, n_bins=10, name='Calibrated SVC')
```



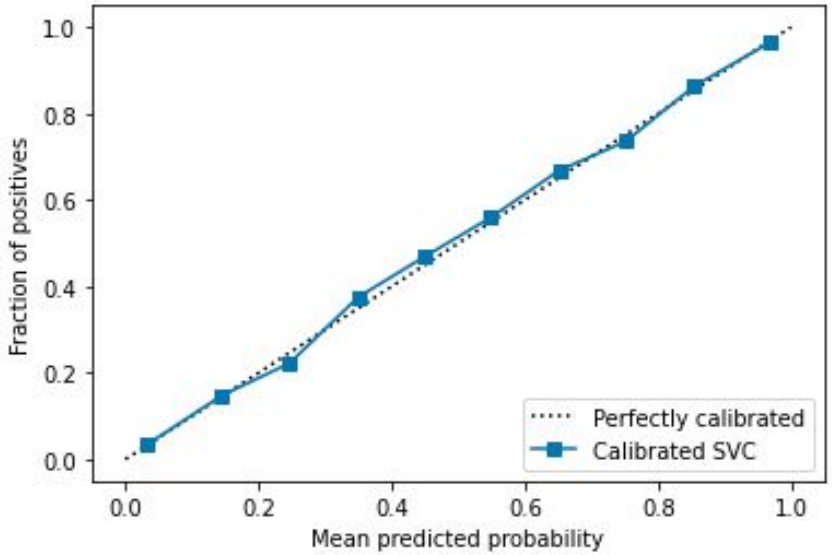
Training Calibrated Classifier - Isotonic Regression

```
cal_svc = CalibratedClassifierCV(svc, cv="prefit", method="isotonic")
cal_svc.fit(X_calib, y_calib)
display = CalibrationDisplay.from_estimator(
    cal_svc, X_test, y_test, n_bins=10, name='Calibrated SVC')
```



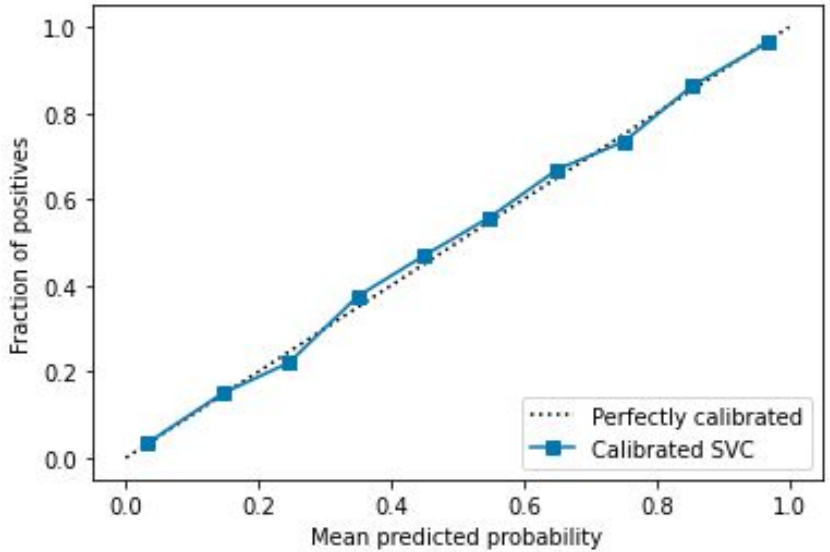
Training Calibrated Classifier - CalibratedClassifierCV

```
cal_svc = CalibratedClassifierCV(svc, method="sigmoid")
cal_svc.fit(X_dev, y_dev)
display = CalibrationDisplay.from_estimator(
    cal_svc, X_test, y_test, n_bins=10, name='Calibrated SVC')
```



Training Calibrated Classifier - CalibratedClassifierCV

```
cal_svc = CalibratedClassifierCV(svc, method="sigmoid", ensemble=False)
cal_svc.fit(X_dev, y_dev)
display = CalibrationDisplay.from_estimator(
    cal_svc, X_test, y_test, n_bins=10, name='Calibrated SVC')
```



Questions?

Hyperparameter tuning & AutoML

Motivation

- Select pre-processing methods
- Select models
- Select (conditional) hyperparameters
    - Dependent on selection of models
        - Neural nets
        - Kernel-based methods
        - Regularization-based methods
        - ...

Combined Algorithm Selection and Hyperparameter (CASH) Optimization

- Large search space
  - Categorical
  - integer
  - Continuous
- Global optimization (non-convex)
- High-dimensional space
  - # of dimensions depends on model selection
- NP-hard problem

Hyperparameter Optimization (HPO)

$$\underset{\theta}{argmax}\, f(\theta)$$

$f(\theta)$ ⟶ model evaluation

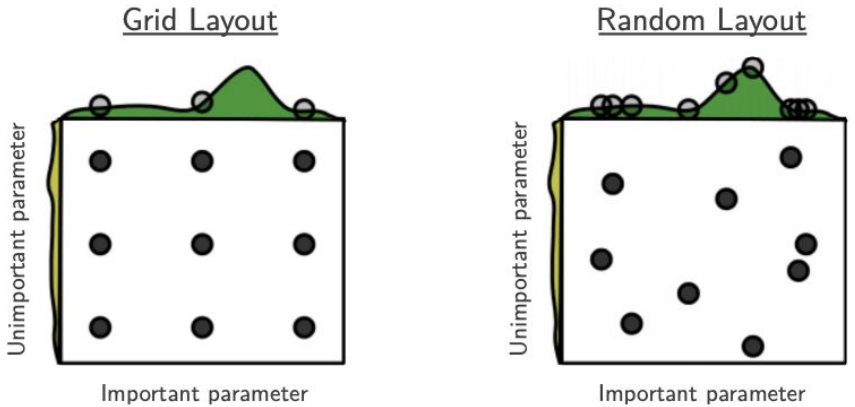
θ ⟶ hyperparameters

Black-Box Optimization Strategies for HPO

- Uninformed search strategies
  - Random Search
  - Grid Search
- Informed search strategies (Sequential Model-Based Optimization - SMBO)
  - Gaussian Process
  - Random Forest (Sequential Model Algorithm Configuration - SMAC)
  - Tree-structured Parzen Estimators (TPE)

Grid Search v.s. Random Search

- Grid search and Random search are both considered as uninformed search strategies
- Possible to combine both strategies:
  - Search a larger space using random search
  - Find promising areas
  - Perform grid search in the smaller area
  - Continue until optimal score is obtained



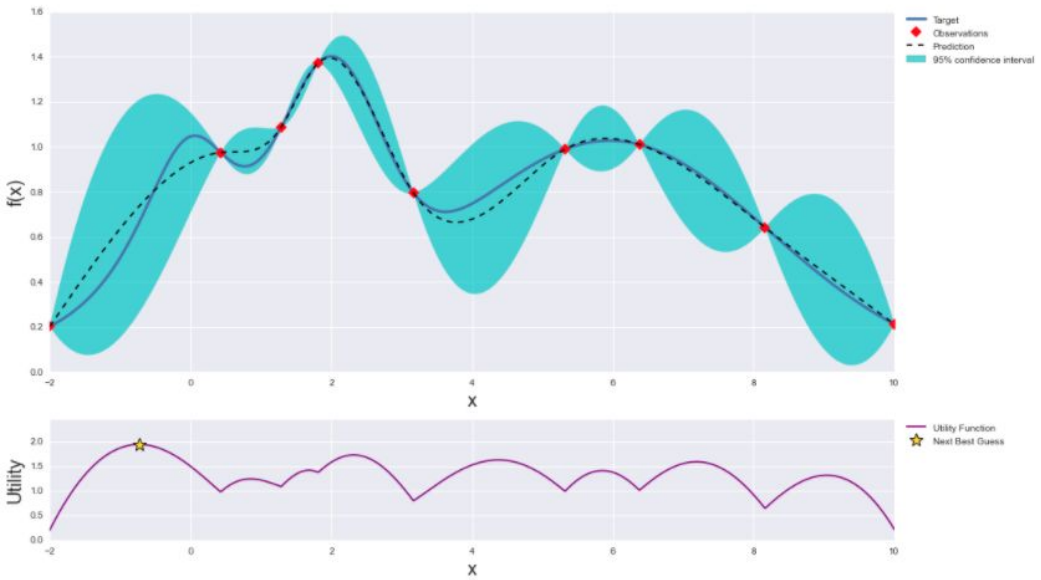
Random Search for Hyper-Parameter Optimization

Grid Search v.s. Random Search

- Grid Search
  - Manually define the grid
  - Exponential in the # of search dimensions
- Random Search
  - Typically does better than Grid Search in higher dimensions (more hyperparameters)

Gaussian Process

- Bayesian optimization works by constructing a probability distribution of possible functions (**Gaussian Process - GP**) that best describe the function you want to optimize.
- A utility function helps explore the parameter space by trading between exploration and exploitation.
- The probability distribution of functions is updated (bayesian) based on observations so far.



https://github.com/fmfn/BayesianOptimization

Random Forest Model

- First model is trained by choosing hyperparameters at random, and the evaluation metric is calculated.
- This (hyperparameter combination, metric output) pair is used to train a Random Forest model
- Several random hyperparameter configurations are chosen and predicted using the trained Random Forest model
- The configuration with the best predicted score is used to train the model

https://ml.informatik.uni-freiburg.de/wp-content/uploads/papers/11-LION5-SMAC.pdf

Tree-structured Parzen Estimator (TPE)

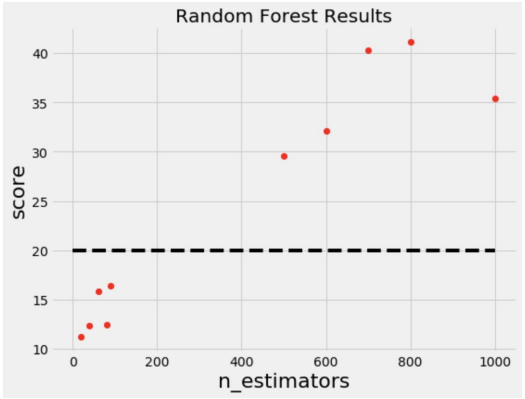
$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) p(y|x) dy$$

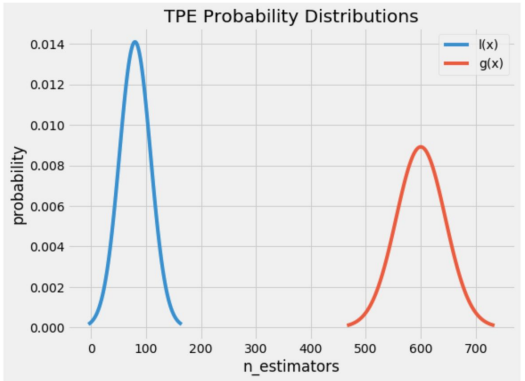
$$p(y|x) = \frac{p(x\,|y) * p(y)}{p(x)}$$

$$p(x|y) = \begin{cases} \ell(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases}$$

$$EI_{y^*}(x) = \frac{\gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y) dy}{\gamma \ell(x) + (1-\gamma) g(x)} \propto \left(\gamma + \frac{g(x)}{\ell(x)}(1-\gamma)\right)^{-1}$$

Choose $\boldsymbol{x}$ that maximizes the $\dfrac{l(x)}{g(x)}$



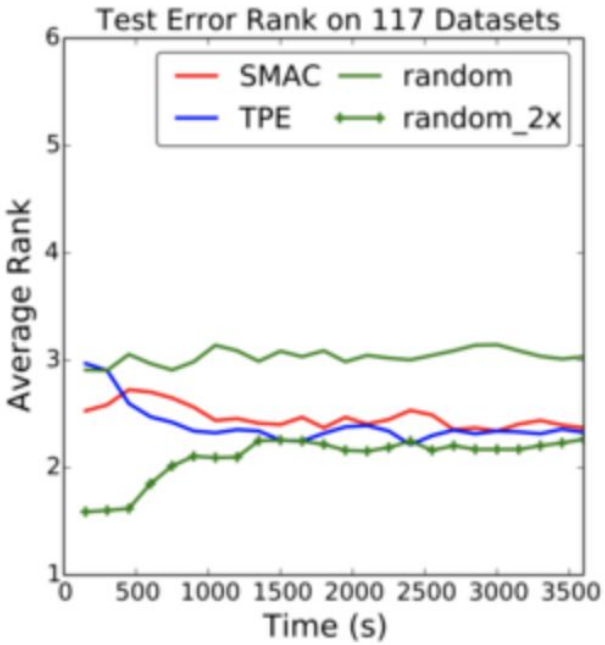


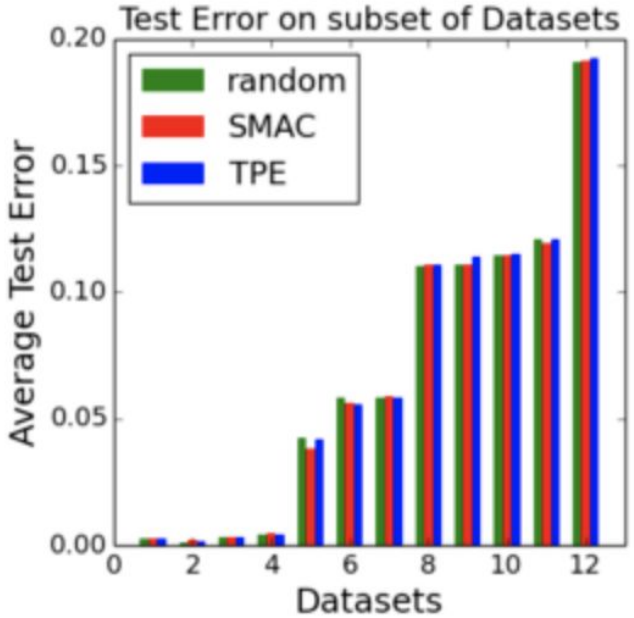
https://bit.ly/2YoQZIW

Implementations

- SMAC (RF model)
- HyperOpt (TPE)
- Scikit-optimize (RF, GP, TPE)
- RayTune (GP, TPE, BOHB, etc.)
- ...

Are all these SMBOs worth the extra computation?





http://www.argmin.net/2016/06/20/hypertuning/

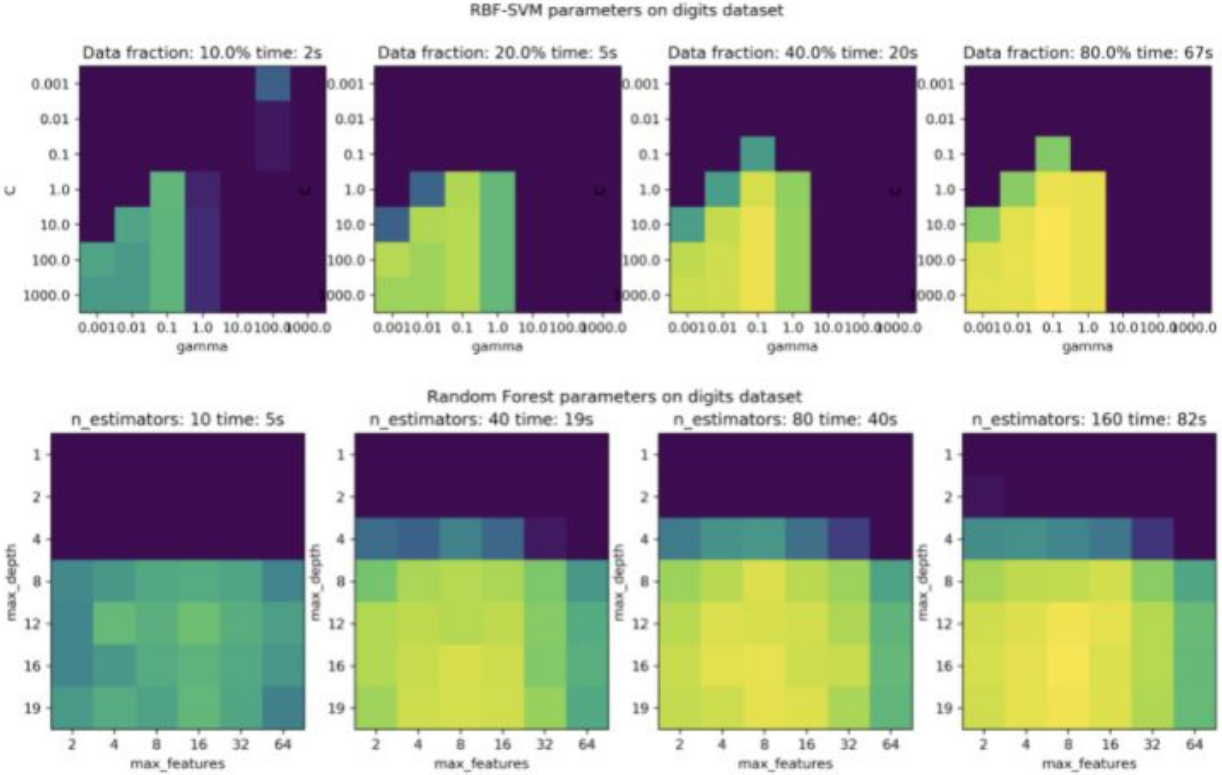
Beyond Black-Box Optimization Strategies

- Hyperparameter gradient descent optimization
- Multi-fidelity optimization
- Meta-learners
- ...

Beyond Black-Box Optimization Strategies

- Hyperparameter gradient descent optimization
- **Multi-fidelity optimization**
- Meta-learners
- ...

Multi-Fidelity Search



https://amueller.github.io/COMS4995-s19/slides/aml-13-parameter-tuning-automl/#p17

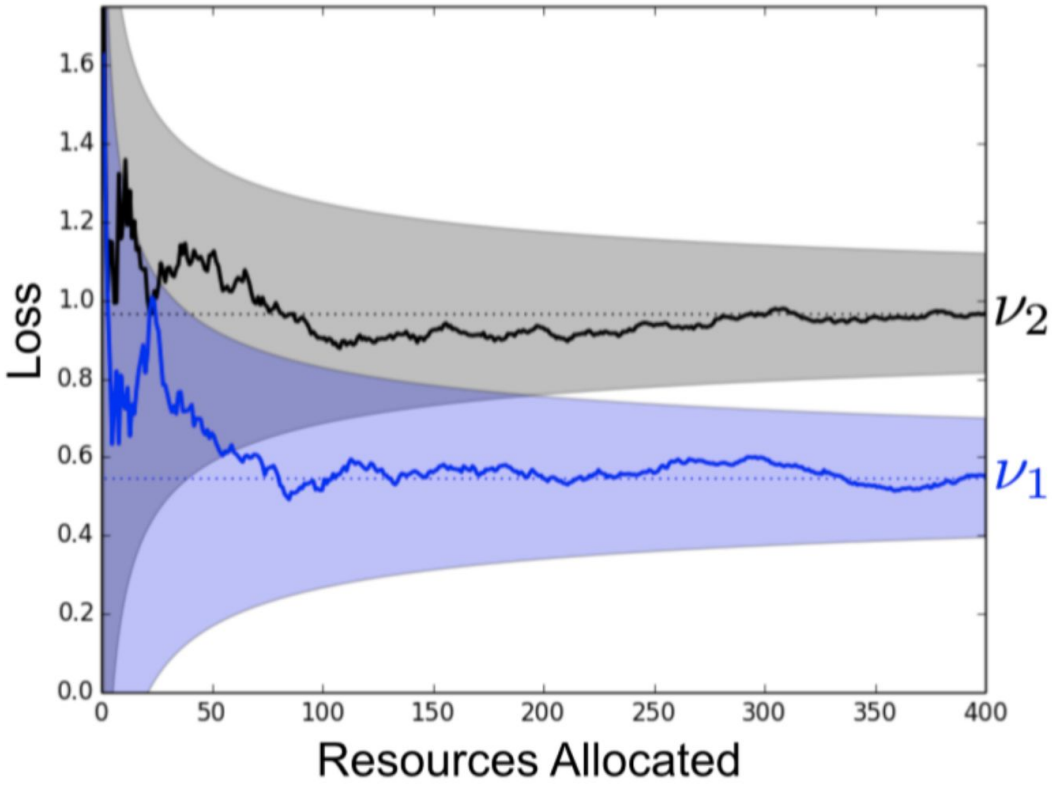
Multi-Fidelity Optimization

- Find optimal hyperparameters given a set of configurations and budget
- The set of configurations can be chosen using random search
- Choose configurations & budget at every iteration to explore and exploit
- Closely related to multi-armed bandits & A/B testing

Multi-Fidelity Optimization

- Successive Halving
- Hyperband (HB)
- Bayesian Optimization Hyperband (BOHB)

Successive Halving - Motivation

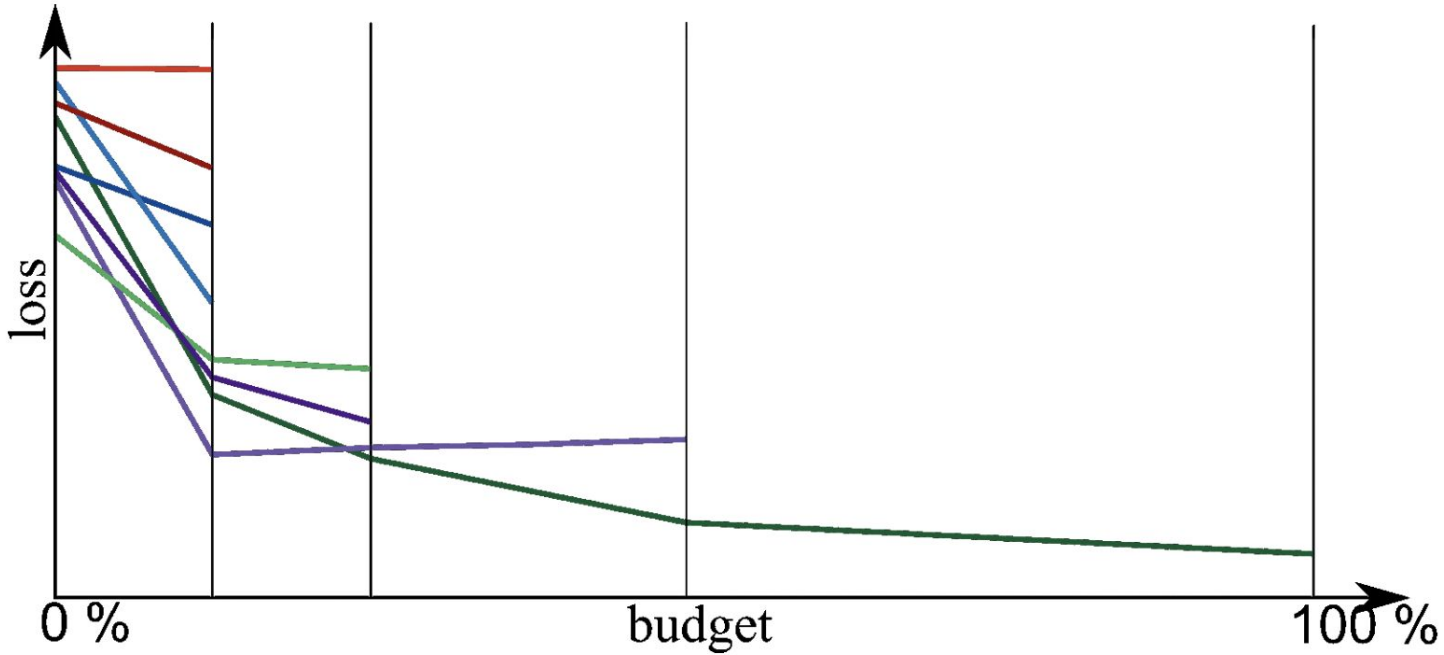


https://arxiv.org/pdf/1603.06560.pdf

Successive Halving - Algorithm

- Assume we have **n** configurations, budget **B** and η = 2 (or 3)
- Allocate curr_budget = B/(k*n) where k = $\log_{\eta}(n) + 1$
- Train **n** models using budget B_0 each and evaluate performance metric
- Retain the top performing **n**/3 configurations
- Repeat steps 2-4 using η * curr_budget every iteration until one configuration remains.

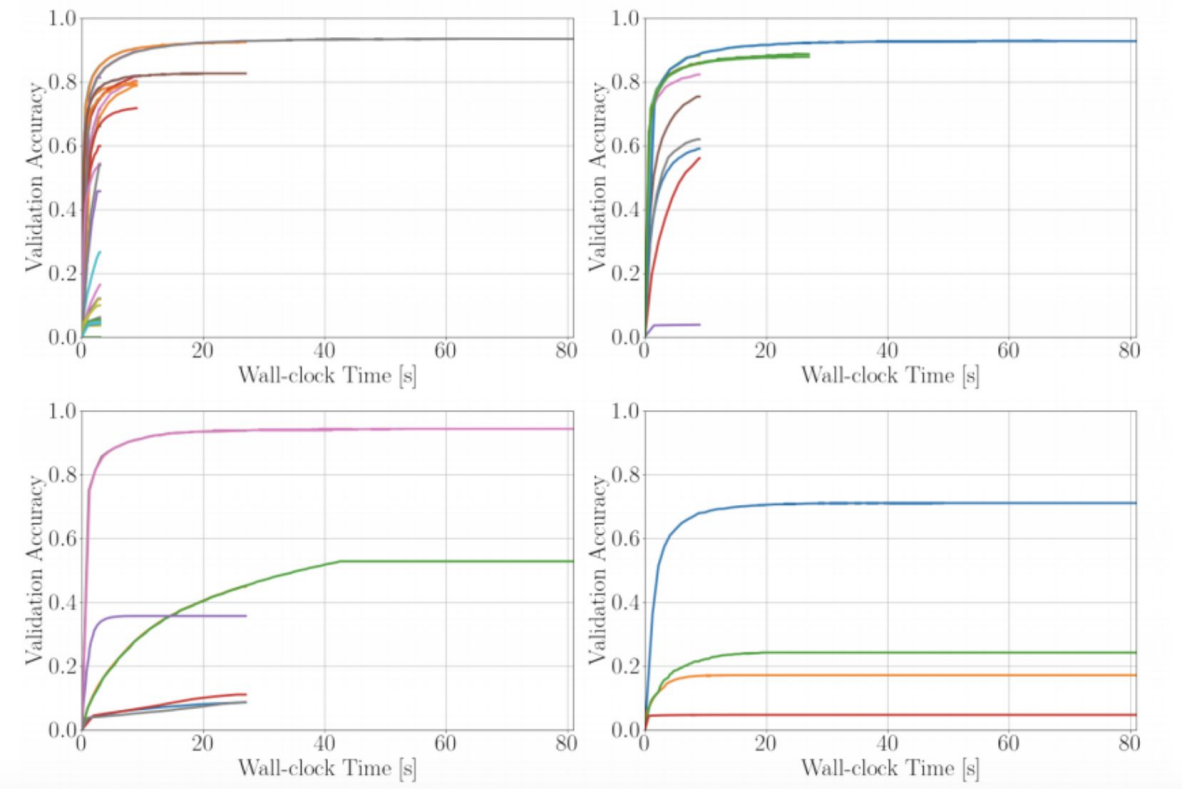
Successive Halving



Hyperband - Motivation

- Successive Halving needs budget **B** and # of configurations **n** to be selected at start
- Unclear how to choose budget **B** and # of configurations **n**
- In Gradient Boosting, generally smaller learning rates perform better when given large resources (estimators)
- HyperBand proposes to frequently perform the successive halving method with different budgets to find the best configurations.

Hyperband



https://amueller.github.io/COMS4995-s20/slides/aml-12-parameter-tuning-automl/#23

Hyperband - Algorithm

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$

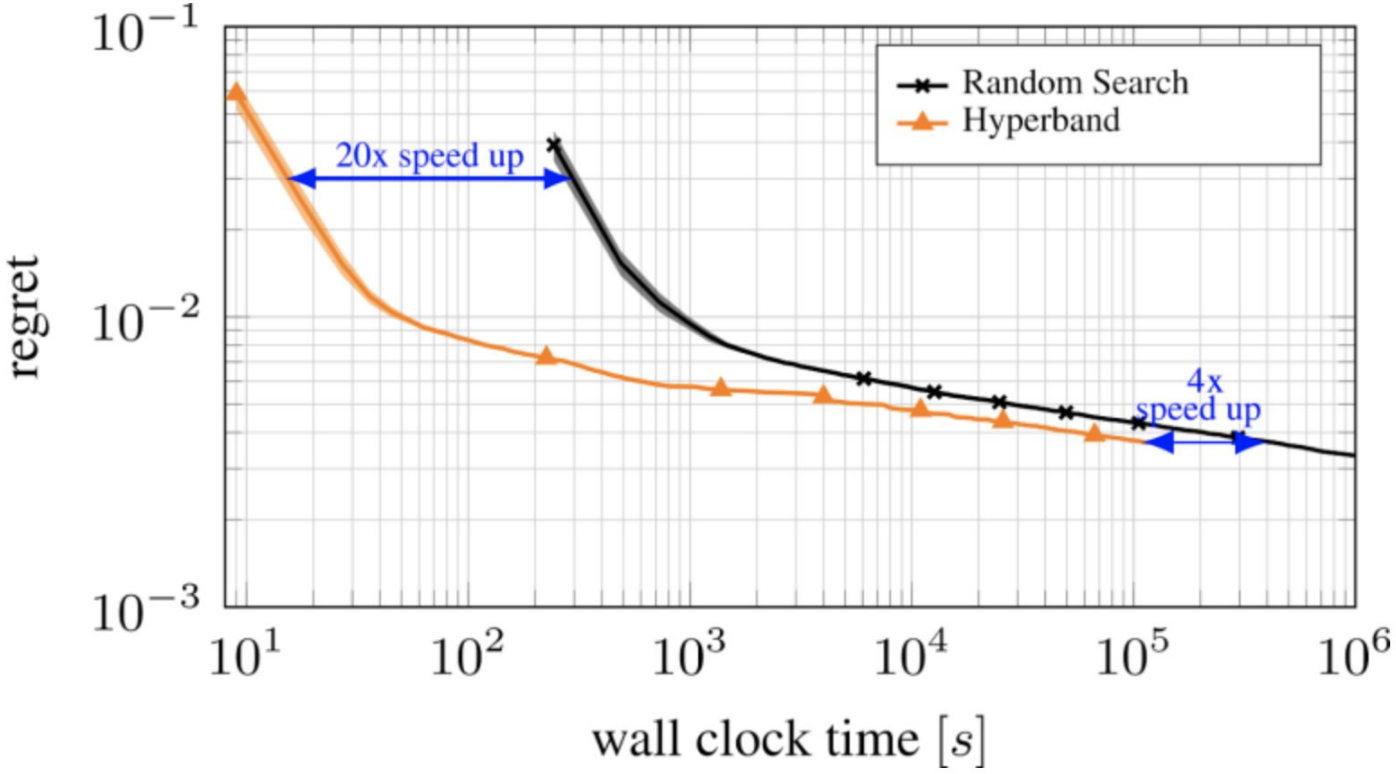
for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil$, $r = R\eta^{-s}$
  // begin SUCCESSIVEHALVING with (n, r) inner loop
  $T =$ `get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$ `top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

Hyperband

```
max_iter = 81        s=4             s=3             s=2             s=1             s=0
eta = 3              n_i   r_i       n_i   r_i       n_i   r_i       n_i   r_i       n_i   r_i
B = 5*max_iter       ---------       ---------       ---------       ---------       ---------
                      81     1        27     3         9     9         6     27        5     81
                      27     3         9     9         3     27        2     81
                       9     9         3     27        1     81
                       3     27        1     81
                       1     81
```

https://homes.cs.washington.edu/~jamieson/hyperband.html

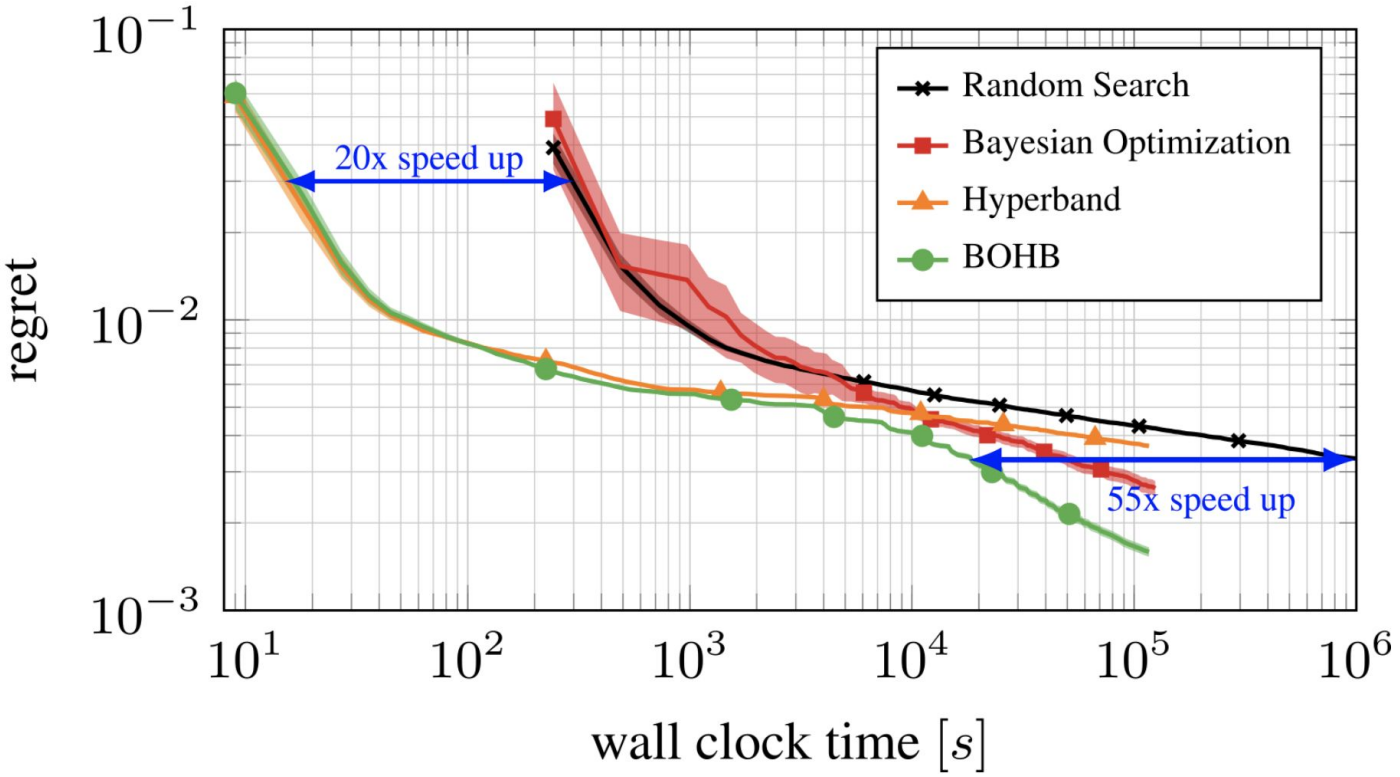
Hyperband


Random Search
Hyperband
20x speed up
4x
speed up
regret
wall clock time [s]
10^{-1}
10^{-2}
10^{-3}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}

Bayesian Optimization Hyperband (BOHB)

- This is an improvement over the Hyperband algorithm
- At each iteration, the configurations are chosen using Kernel Density Estimation (KDEs) instead of being random.
- The methodology of choosing the next best configurations is very similar to TPE algorithm.

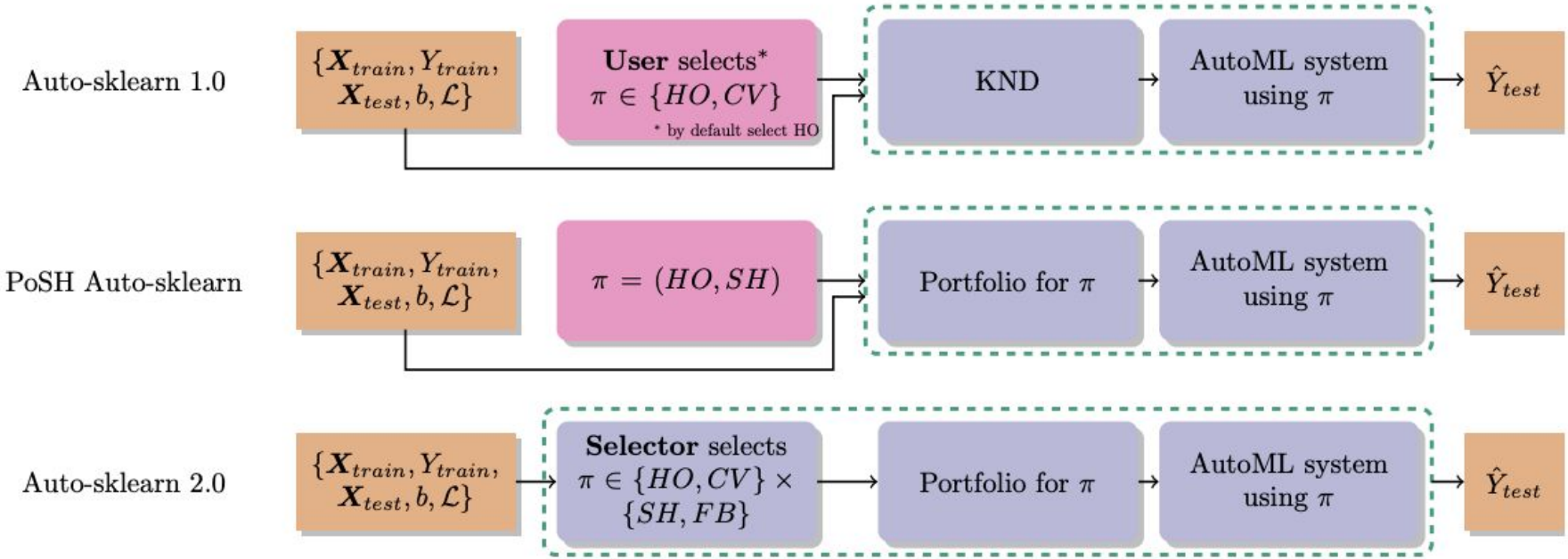
Bayesian Optimization Hyperband (BOHB)



Implementations

- HpBandSter
- RayTune
- Scikit-hyperband (HB implementation only)

AutoML



https://arxiv.org/pdf/2007.04074.pdf

Practical Considerations

- Multi-fidelity search strategies are effective and work well
- Auto-sklearn could give you a quick baseline for further improvements
- Successive halving is very easy to implement and test
- BOHB/HB tend to work well

https://arxiv.org/pdf/2007.04074.pdf

Criticisms

- Is all this complexity worth it?
- Do we really need to train 100s of classifiers?
- AutoML probably making ML too easy
- Harder to interpret

https://arxiv.org/pdf/2007.04074.pdf

Questions?