Model Evaluation and Selection

Supervised ML: Classification

- Accuracy: http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

where $1(x)$ is the indicator function.

```
>>> import numpy as np
>>> from sklearn.metrics import accuracy_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> accuracy_score(y_true, y_pred)
0.5
>>> accuracy_score(y_true, y_pred, normalize=False)
2
```

Supervised ML: Classification

- Accuracy:
  http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

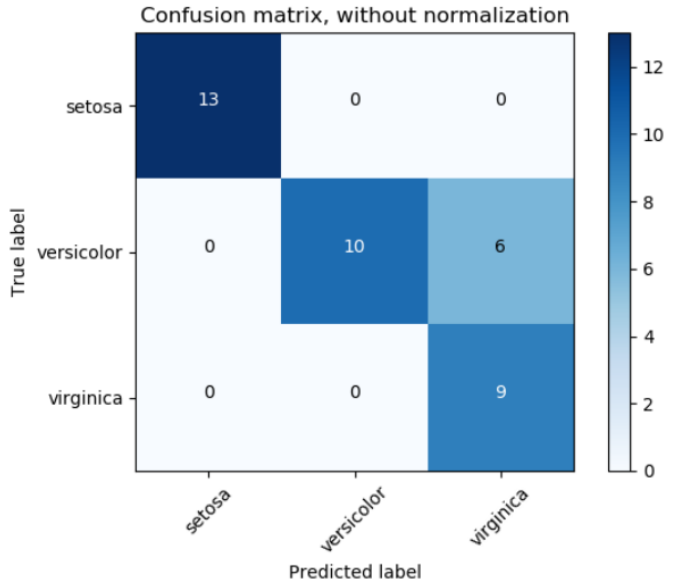
- Exercise:
  y_actual: [Win, Win, Win, Loss, Loss]
  y_predicted: [Loss, Win, Win, Loss, Win]

Supervised ML: Classification

- Confusion Matrix: http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

Here is a visual representation of such a confusion matrix (this figure comes from the Confusion matrix example):



Supervised ML: Classification

- Confusion Matrix:
  http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix

- Exercise:
  y_actual: [Win, Win, Win, Loss, Loss]
  y_predicted: [Loss, Win, Win, Loss, Win]

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

For binary problems, we can get counts of true negatives, false positives, false negatives and true positives as follows:

```
>>> y_true = [0, 0, 0, 1, 1, 1, 1, 1]
>>> y_pred = [0, 1, 0, 1, 0, 1, 0, 1]
>>> tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
>>> tn, fp, fn, tp
(2, 1, 2, 3)
```

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{N} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

precision or positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

negative predictive value (NPV)

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}}$$

miss rate or false negative rate (FNR)

$$\mathrm{FNR} = \frac{\mathrm{FN}}{P} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

fall-out or false positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{N} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

false discovery rate (FDR)

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$

false omission rate (FOR)

$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TN}} = 1 - \mathrm{NPV}$$

accuracy (ACC)

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{P + N} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

condition positive (P)

the number of real positive cases in the data

condition negative (N)

the number of real negative cases in the data

---

true positive (TP)

eqv. with hit

true negative (TN)

eqv. with correct rejection

false positive (FP)

eqv. with false alarm, Type I error

false negative (FN)

eqv. with miss, Type II error

F1 score

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Matthews correlation coefficient (MCC)

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$$

Informedness or Bookmaker Informedness (BM)

$$\mathrm{BM} = \mathrm{TPR} + \mathrm{TNR} - 1$$

Markedness (MK)

$$\mathrm{MK} = \mathrm{PPV} + \mathrm{NPV} - 1$$

Sources: Fawcett (2006), Powers (2011), and Ting (2011) [1] [2] [3]

- https://en.wikipedia.org/wiki/Confusion_matrix

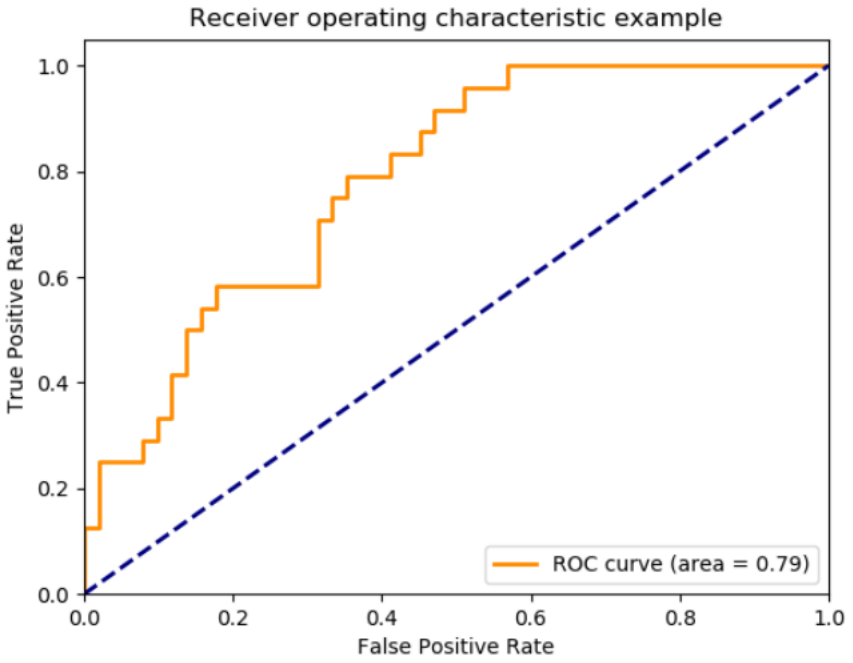
Supervised ML: Classification

- Receiver operating characteristic (ROC) curve: https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics

This function requires the true binary value and the target scores, which can either be probability estimates of the positive class, confidence values, or binary decisions. Here is a small example of how to use the `roc_curve` function:

```
>>> import numpy as np
>>> from sklearn.metrics import roc_curve
>>> y = np.array([1, 1, 2, 2])
>>> scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> fpr, tpr, thresholds = roc_curve(y, scores, pos_label=2)
>>> fpr
array([0. , 0. , 0.5, 0.5, 1. ])
>>> tpr
array([0. , 0.5, 0.5, 1. , 1. ])
>>> thresholds
array([1.8 , 0.8 , 0.4 , 0.35, 0.1 ])
```

```
>>> import numpy as np
>>> from sklearn.metrics import roc_auc_score
>>> y_true = np.array([0, 0, 1, 1])
>>> y_scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> roc_auc_score(y_true, y_scores)
0.75
```



Supervised ML: Classification

- Receiver operating characteristic (ROC) curve: https://scikit-learn.org/stable/modules/model_evaluation.html#roc-metrics

- Exercise:
  y_actual: [Win, Win, Win, Loss, Loss]
  y_predicted: [Loss, Win, Win, Loss, Win]

Supervised ML: Classification

- Precision, Recall, F-Score: http://scikit-learn.org/stable/modules/model_evaluation.html#binary-classification

Here are some small examples in binary classification:

```
>>> from sklearn import metrics
>>> y_pred = [0, 1, 0, 0]
>>> y_true = [0, 1, 0, 1]
>>> metrics.precision_score(y_true, y_pred)
1.0
>>> metrics.recall_score(y_true, y_pred)
0.5
>>> metrics.f1_score(y_true, y_pred)
0.66...
>>> metrics.fbeta_score(y_true, y_pred, beta=0.5)
0.83...
>>> metrics.fbeta_score(y_true, y_pred, beta=1)
0.66...
>>> metrics.fbeta_score(y_true, y_pred, beta=2)
0.55...
>>> metrics.precision_recall_fscore_support(y_true, y_pred, beta=0.5)
(array([0.66..., 1.        ]), array([1. , 0.5]), array([0.71..., 0.83...]), array([2, 2]))
```

$$\text{precision} = \frac{tp}{tp + fp},$$

$$\text{recall} = \frac{tp}{tp + fn},$$

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}.$$

Supervised ML: Classification

- Classification Report:
  http://scikit-learn.org/stable/modules/model_evaluation.html#binary-classification

- Exercise:
  y_actual: [Win, Win, Win, Loss, Loss]
  y_predicted: [Loss, Win, Win, Loss, Win]

Supervised ML: Classification

- Classification Report: https://scikit-learn.org/stable/modules/model_evaluation.html#classification-report

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 0]
>>> y_pred = [0, 0, 2, 1, 0]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support

     class 0       0.67      1.00      0.80         2
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.50      0.67         2

   micro avg       0.60      0.60      0.60         5
   macro avg       0.56      0.50      0.49         5
weighted avg       0.67      0.60      0.59         5
```

Supervised ML: Classification

- Classification Report:
  https://scikit-learn.org/stable/modules/model_evaluation.html#classification-report

- Exercise:
  y_actual: [Win, Win, Win, Loss, Loss, Draw, Draw]
  y_predicted: [Loss, Win, Win, Loss, Win, Draw, Win]

Supervised ML: Regression

- Mean Absolute Error: https://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| .$$

Here is a small example of usage of the `mean_absolute_error` function:

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> mean_absolute_error(y_true, y_pred)
0.5
>>> y_true = [[0.5, 1], [-1, 1], [7, -6]]
>>> y_pred = [[0, 2], [-1, 2], [8, -5]]
>>> mean_absolute_error(y_true, y_pred)
0.75
>>> mean_absolute_error(y_true, y_pred, multioutput='raw_values')
array([0.5, 1. ])
>>> mean_absolute_error(y_true, y_pred, multioutput=[0.3, 0.7])
...
0.85...
```

Supervised ML: Regression

- Mean Absolute Error:
  https://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error

- Exercise:
  y_actual: [100, 200, 210, 150, 180]
  y_predicted: [98, 180, 222, 160, 165]

Supervised ML: Regression

- Mean Squared Error: https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Here is a small example of usage of the `mean_squared_error` function:

```
>>> from sklearn.metrics import mean_squared_error
>>> y_true = [3, -0.5, 2, 7]
>>> y_pred = [2.5, 0.0, 2, 8]
>>> mean_squared_error(y_true, y_pred)
0.375
>>> y_true = [[0.5, 1], [-1, 1], [7, -6]]
>>> y_pred = [[0, 2], [-1, 2], [8, -5]]
>>> mean_squared_error(y_true, y_pred)
0.7083...
```

Supervised ML: Regression

- Mean Squared Error:
  https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error

- Exercise:
  y_actual: [100, 200, 210, 150, 180]
  y_predicted: [98, 180, 222, 160, 165]

Supervised ML: Regression

- Mean squared logarithmic error: https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-logarithmic-error

$$\text{MSLE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (\log_e(1 + y_i) - \log_e(1 + \hat{y}_i))^2.$$

Where $\log_e(x)$ means the natural logarithm of x. This metric is best to use when targets having exponential growth, such as population counts, average sales of a commodity over a span of years etc. Note that this metric penalizes an under-predicted estimate greater than an over-predicted estimate.

Here is a small example of usage of the `mean_squared_log_error` function:

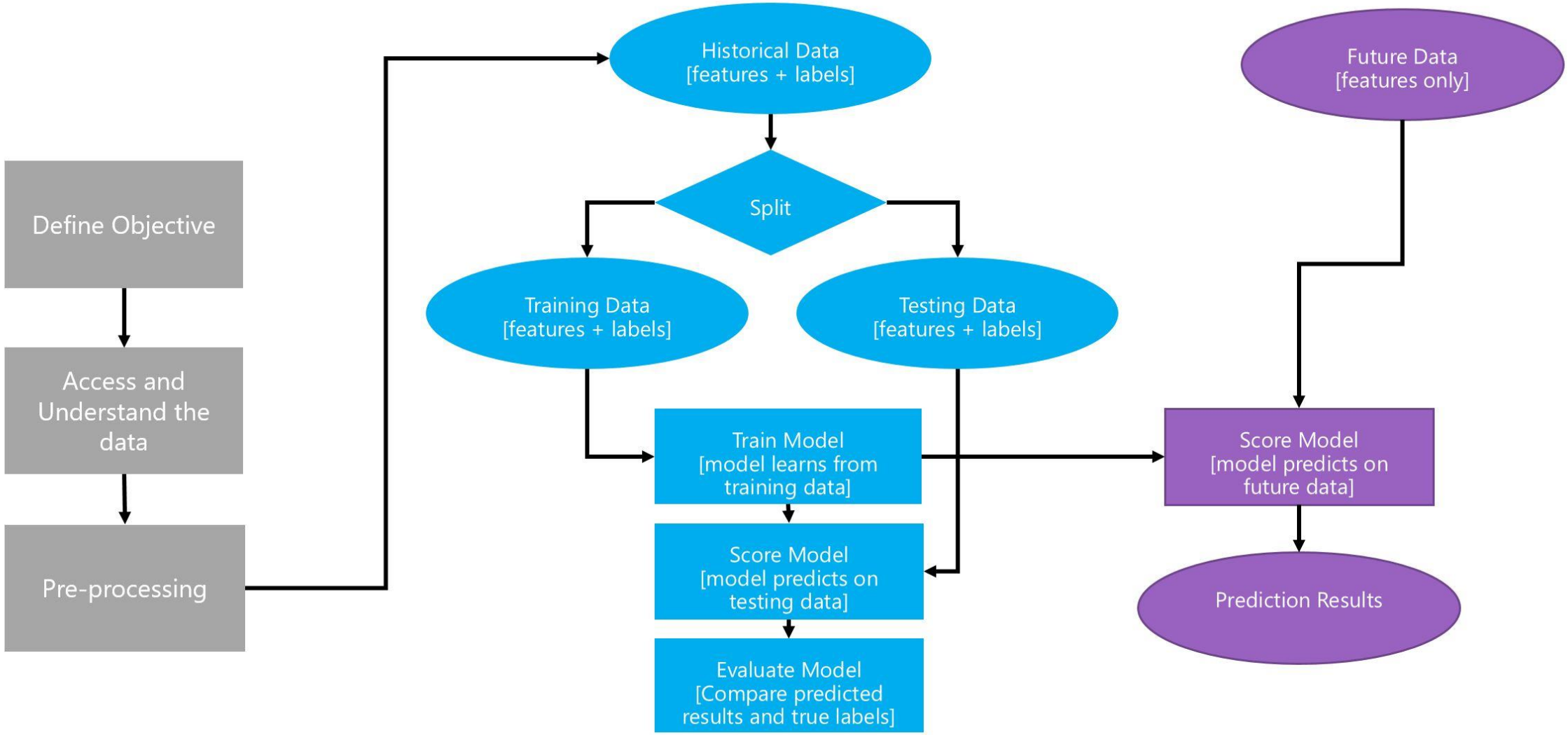
```
>>> from sklearn.metrics import mean_squared_log_error
>>> y_true = [3, 5, 2.5, 7]
>>> y_pred = [2.5, 5, 4, 8]
>>> mean_squared_log_error(y_true, y_pred)
0.039...
>>> y_true = [[0.5, 1], [1, 2], [7, 6]]
>>> y_pred = [[0.5, 2], [1, 2.5], [8, 8]]
>>> mean_squared_log_error(y_true, y_pred)
0.044...
```

Supervised ML: Regression

- Mean squared logarithmic error:
  https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-logarithmic-error

- Exercise:
  y_actual: [100, 200, 210, 150, 180]
  y_predicted: [98, 180, 222, 160, 165]

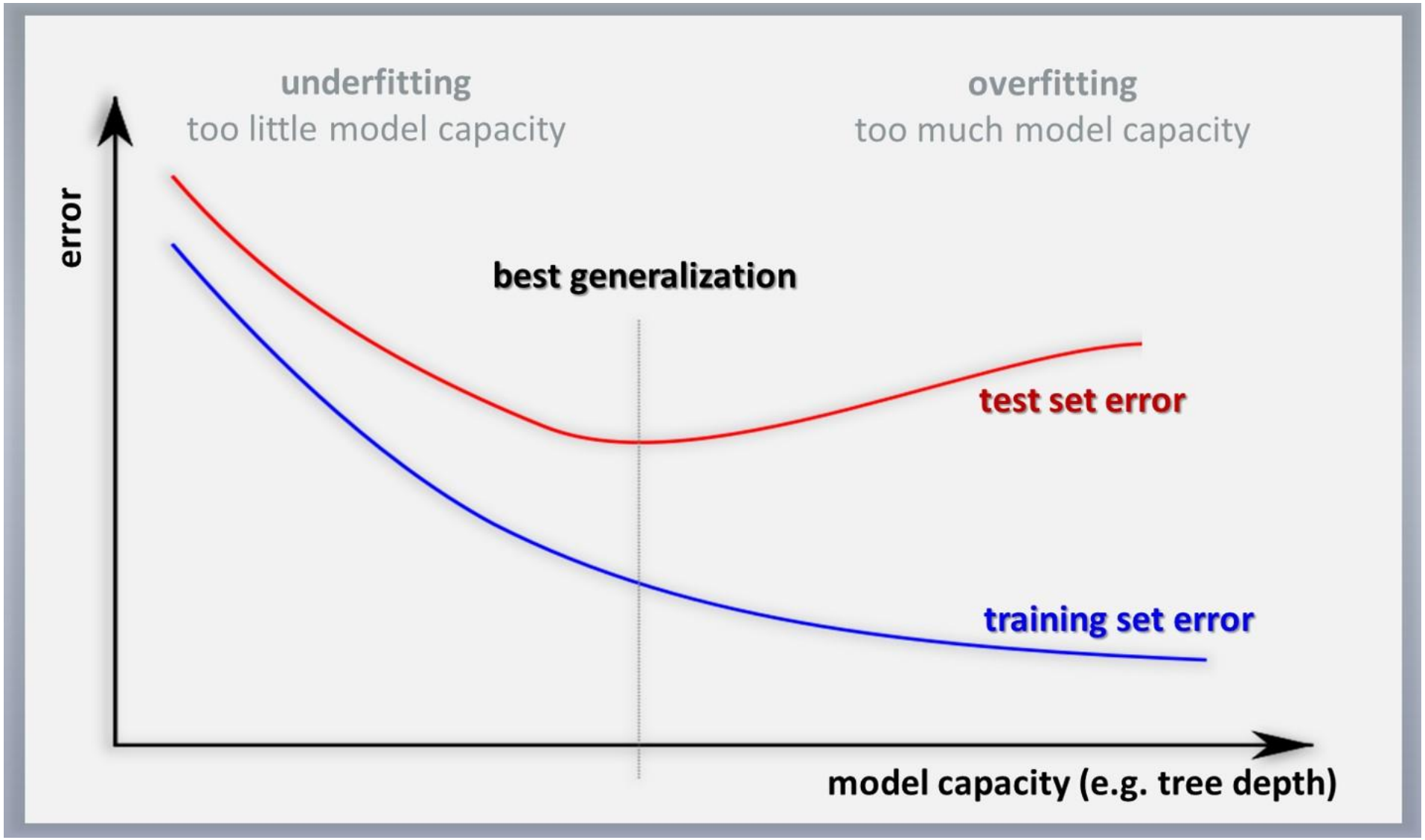
Machine Learning Process



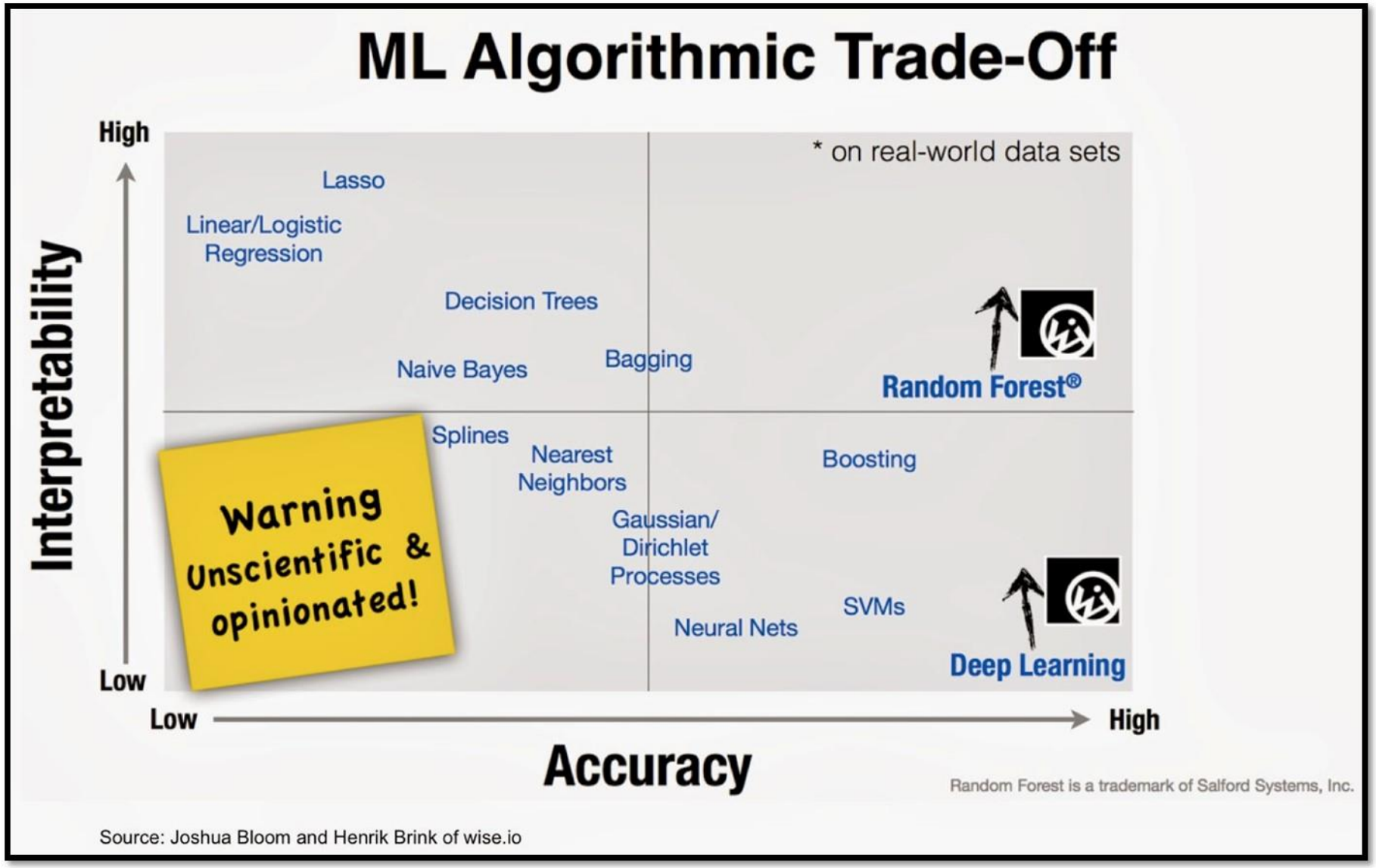
Overfitting

- A similar situation to "memorizing the answers" can happen with models.
- A good model should capture meaningful trends (signals) in the training data, meaning trends that generalize to data outside the training data (out-of-sample as statisticians call it).
- An apparent trend that fails to generalize to out-of-sample data (noise) should be ignored by the model. Otherwise the model is said to overfit.
- Without a test data, we can't distinguish the signal from the noise.
- The more "complex" models are more likely to overfit.

Complexity for the same model (e.g. RF)



Complexity for different models



Models should be simple, but not simplistic

don't do this	why?	do this instead
simply throw all our variables at the model	multi-collinearity (feature redundancy)	variable selection or feature engineering
simply pick the most accurate model	more likely to overfit, harder to interpret, maybe less efficient	decide on the right trade-off between accuracy and complexity

Short Quiz

- True or false:
  1. When building models, there is no point extracting less granular features from more granular features (for example, month from a datetime column). More information is always better. (T/F)

  2. overfitting affects more simple models. (T/F)

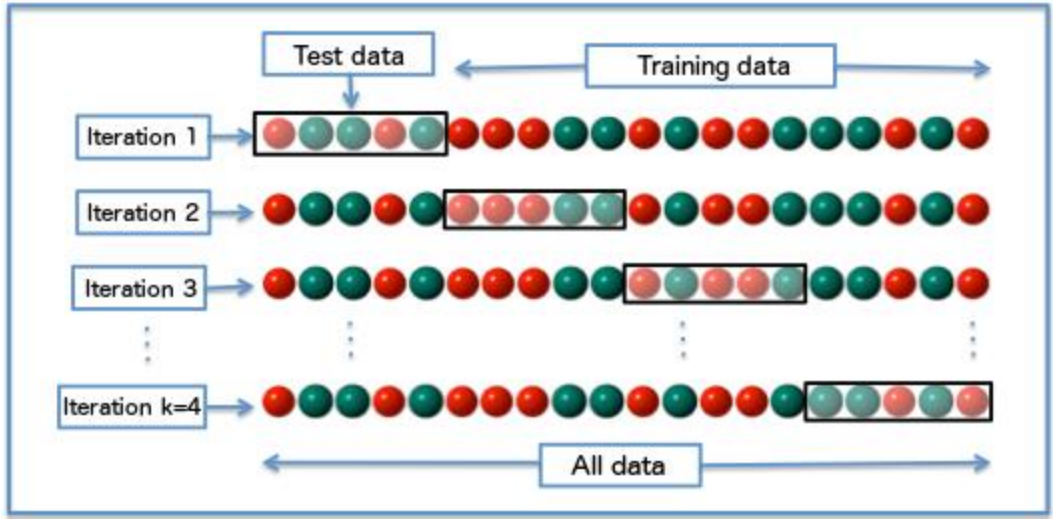
- Why does having more features not necessarily result in better models?

Model Selection: K-fold Cross Validation

A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV. In the basic approach, called *k*-fold CV, the training set is split into *k* smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the *k* "folds":

- A model is trained using $k - 1$ of the folds as training data;
- the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure reported by *k*-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as inverse inference where the number of samples is very small.



https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Model Parameter Turning: GridSearchCV

Examples

```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC(gamma="scale")
>>> clf = GridSearchCV(svc, parameters, cv=5)
>>> clf.fit(iris.data, iris.target)
...
GridSearchCV(cv=5, error_score=...,
       estimator=SVC(C=1.0, cache_size=..., class_weight=..., coef0=...,
                     decision_function_shape='ovr', degree=..., gamma=...,
                     kernel='rbf', max_iter=-1, probability=False,
                     random_state=None, shrinking=True, tol=...,
                     verbose=False),
       fit_params=None, iid=..., n_jobs=None,
       param_grid=..., pre_dispatch=..., refit=..., return_train_score=...,
       scoring=..., verbose=...)
>>> sorted(clf.cv_results_.keys())
...
['mean_fit_time', 'mean_score_time', 'mean_test_score',...
 'mean_train_score', 'param_C', 'param_kernel', 'params',...
 'rank_test_score', 'split0_test_score',...
 'split0_train_score', 'split1_test_score', 'split1_train_score',...
 'split2_test_score', 'split2_train_score',...
 'std_fit_time', 'std_score_time', 'std_test_score', 'std_train_score'...]
```

- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

Exercise 1: Tutorial on scikit learn

- Cross-validation on diabetes Dataset Exercise

- https://scikit-learn.org/stable/auto_examples/exercises/plot_cv_diabetes.html#sphx-glr-auto-examples-exercises-plot-cv-diabetes-py

Exercise 2: Feature Importance

- Fitting random forest model in the automobile price data

- 1. Print the sorted table of feature importance of random forest https://towardsdatascience.com/running-random-forests-inspect-the-feature-importances-with-this-code-2b00dd72b92e

- 2. plotting the feature importance https://stackoverflow.com/questions/44101458/random-forest-feature-importance-chart-using-python