COMP 4432 Machine Learning

Lesson 3: Classification

Agenda

- Assignment 2
- Classification versus Regression
- Classification Algorithms
- Performance Metrics

Assignment 2

- Focused on Regression
  - Linear Regression
  - Decision Tree Regression (week 6)
  - Random Forest Regression (week 7)
- SciKit Functions
  - train_test_split
    - test_size
    - random_state
  - cross_val_score
    - scoring
    - cv
  - GridSearchCV
    - estimator
    - param_grid
    - scoring
    - cv

Linear Regression

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$h_\theta(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \ldots + \theta_f x_f^{(i)} h_\theta(x^{(i)}) = \theta^T x^{(i)}$$

Find optimal values in θ by minimizing $J(\theta)$

Closed Form

$$\theta = (X^T X)^{-1} X^T y$$

Gradient Descent

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta)$$

$$\nabla_\theta J(\theta) = X^T (X\theta - y)$$

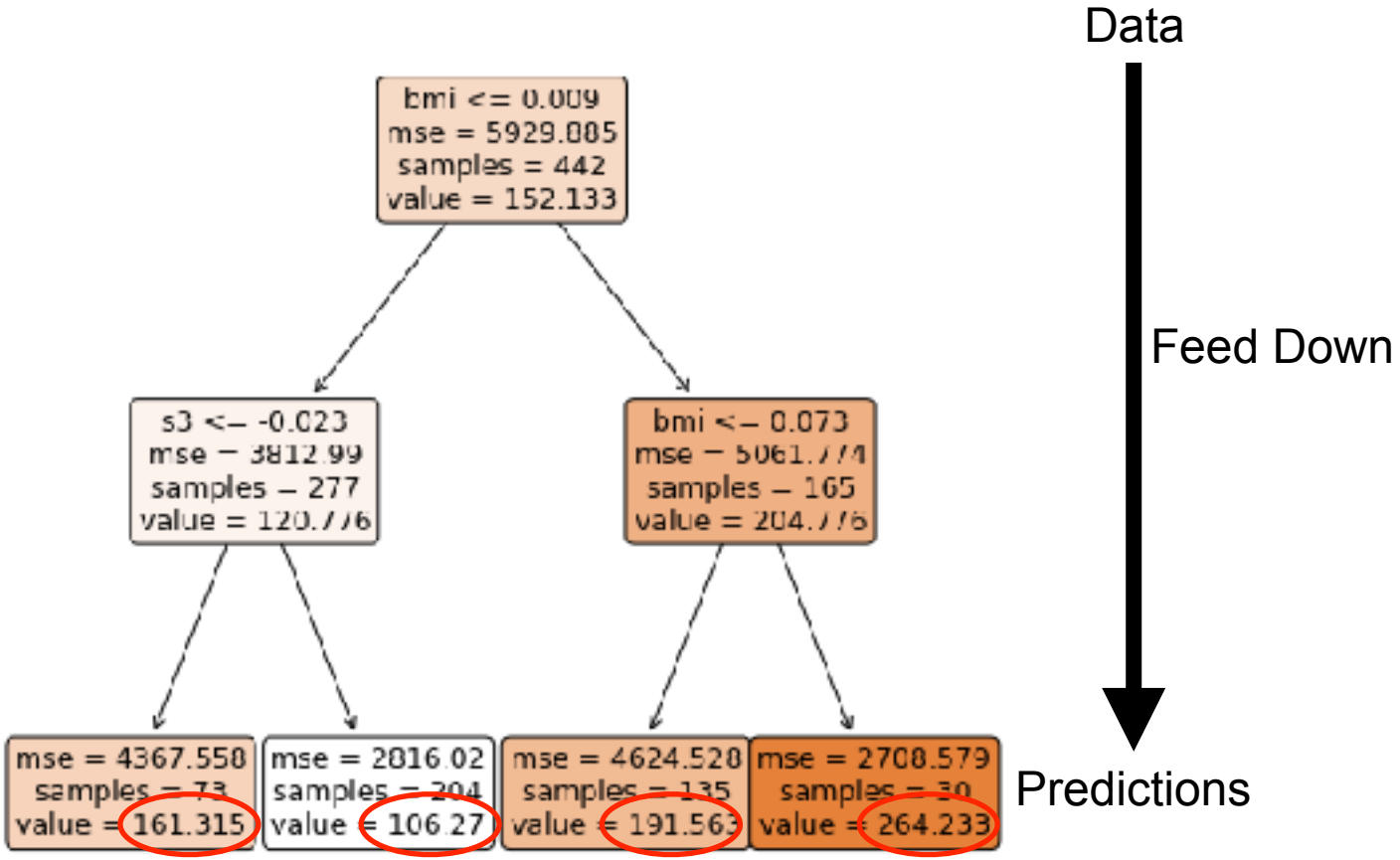
Linear Regression

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

$$h_\theta(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \ldots + \theta_f x_f^{(i)} h_\theta(x^{(i)}) = \theta^T x^{(i)}$$

Easy to interpret.

Decision Tree Regressor



Decision Tree Regressor

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Calculate the midpoint between i and i + 1 (split point)

Unique values of feature

Calculate the MSE for the target values on either side of split

RM	PRICE	
3.561	27.5	Left Node
4.138	11.9	
4.368	8.8	
4.519	7.0	
4.628	17.9	
...	...	Right Node
8.375	50.0	
8.398	48.8	
8.704	50.0	
8.725	50.0	
8.780	21.9	

Decision Tree Regressor

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Calculate the midpoint between i and i + 1 (split point)
    Unique values of feature
Calculate the MSE for the target values on either side of split
Calculate the Total Cost
Gives (Split Point, Total Cost)

RM	PRICE	
3.561	27.5	
4.138	11.9	
4.368	8.8	
4.519	7.0	
4.628	17.9	Left Node
...	...	
8.375	50.0	
8.398	48.8	
8.704	50.0	
8.725	50.0	Right Node
8.780	21.9	

Decision Tree Regressor

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Calculate the midpoint between i and i + 1 (split point)
    Unique values of feature
Calculate the MSE for the target values on either side of split
Calculate the Total Cost
Gives (Split Point, Total Cost)

Identify the Split Point with the Minimum Total Cost per Feature

	total_cost	cutpoint
age	5700.035157	0.007199
sex	5918.888900	0.003019
bmi	4279.164764	0.009422
bp	4919.231732	0.023594
s1	5572.695496	0.005999
s2	5658.358682	0.017318
s3	5046.367626	-0.015789

Decision Tree Regressor

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Calculate the midpoint between i and i + 1 (split point)
    Unique values of feature
Calculate the MSE for the target values on either side of split
Calculate the Total Cost
Gives (Split Point, Total Cost)

Identify the Split Point with the Minimum Total Cost per Feature

If multiple features are being considered, each feature follows the method above giving :
(Feature, Optimal Split Point, Minimum Total Cost)

The feature with the most minimum Total Cost is selected.

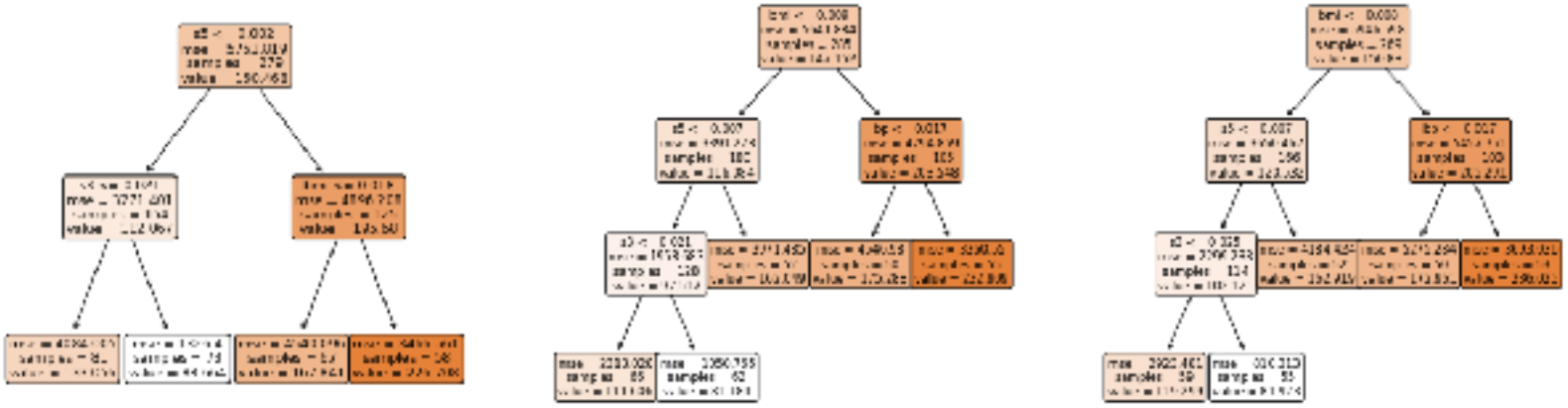
	total_cost	cutpoint
age	5700.035157	0.007199
sex	5918.888900	0.003019
bmi	4279.164764	0.009422
bp	4919.231732	0.023594
s1	5572.695496	0.005999
s2	5658.358682	0.017318
s3	5046.367626	-0.015789

Decision Tree Regressor

- Individual trees are experts to the information they were shown
- By default, decision trees are grown until the leaves are pure. Therefore, they have large variance, tend to overfit, and don't generalize well
- Limit the depth to allow better generalizability
  - Hyperparameter tuning
  - *max_depth* or *min_samples_leaf*
  - How do we know the optimal depth?
    - Hyperparameter tuning
    - SciKit Documentation

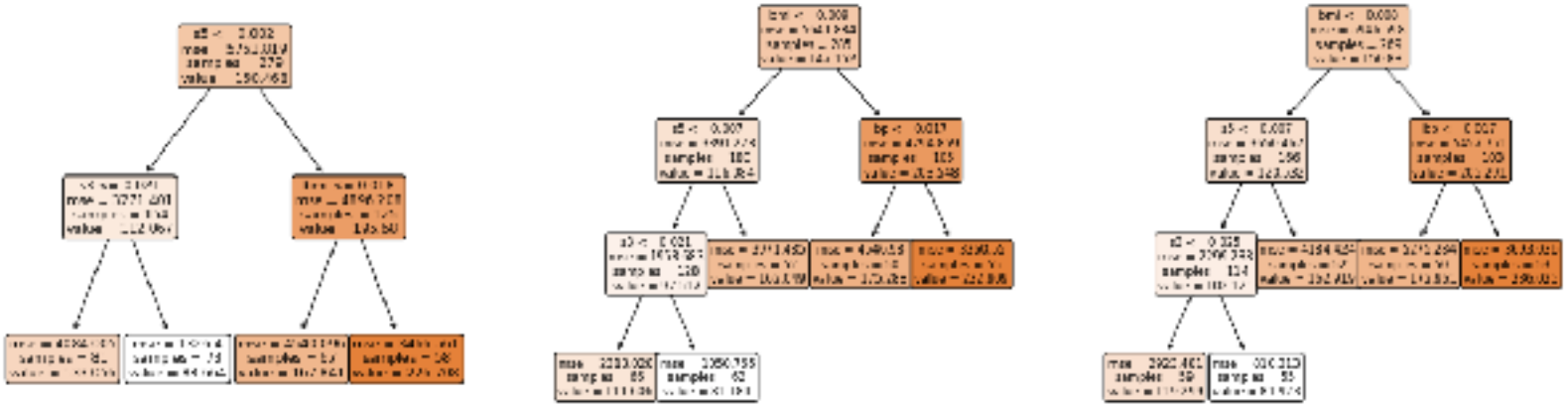
Random Forest Regressor

- Build an ensemble of decision trees



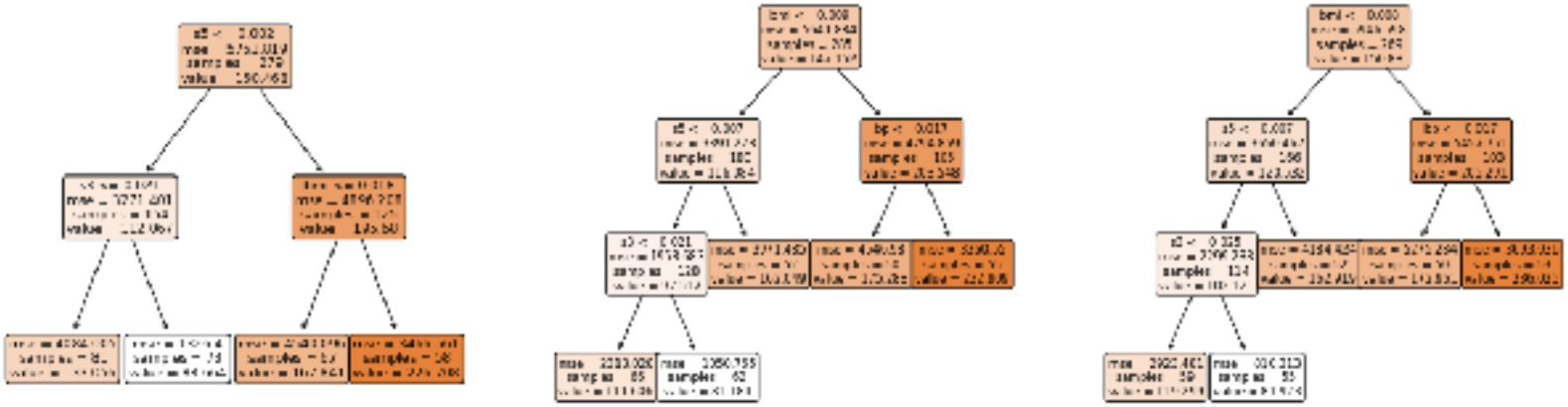
Random Forest Regressor

- Want independence and diversity amongst the constituent decision trees
  - Build each tree with a random sample of the data
  - Select the best split from a random subset of the features

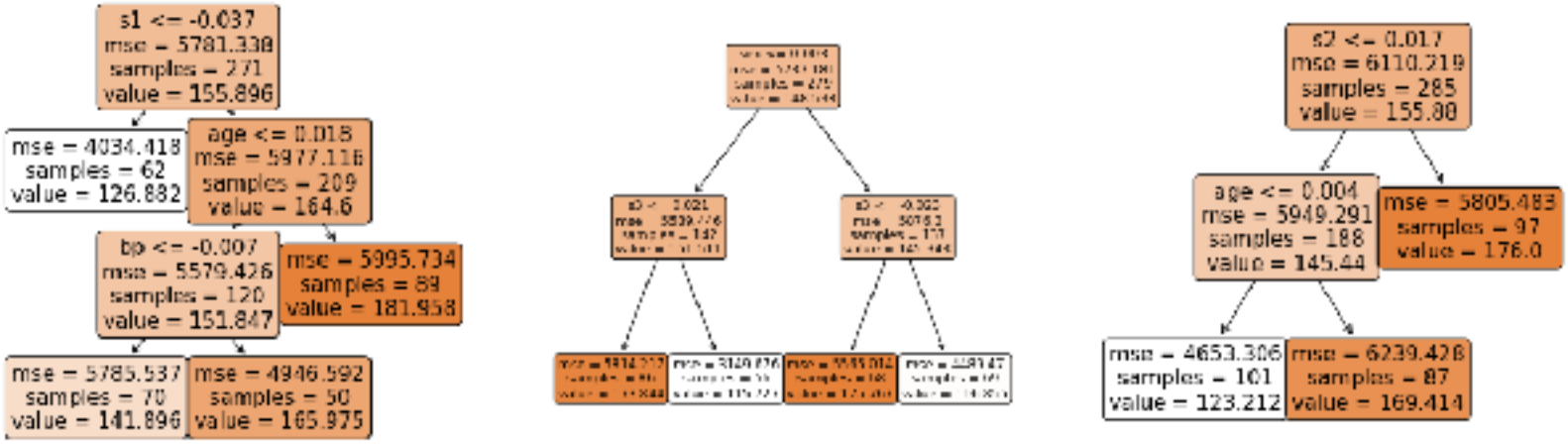


Random Forest Regressor

max_features = None (Consider all features at each split)



max_features = 0.2 (Randomly select only 20% of the features at each split)



Implementation

Theory says to build the deepest trees possible for a given data sampling.

Each tree is an *expert* at the data it has observed, and each *expert* offers their insights in the aggregation phase.

Over many participants, the sampled data can look similar to each other, and the trees begin to correlate.

Hyperparameter Tuning

n_estimators, *max_features*, *min_samples_leaf, max_samples* (maybe)
SciKit Documentation

Train Test Split

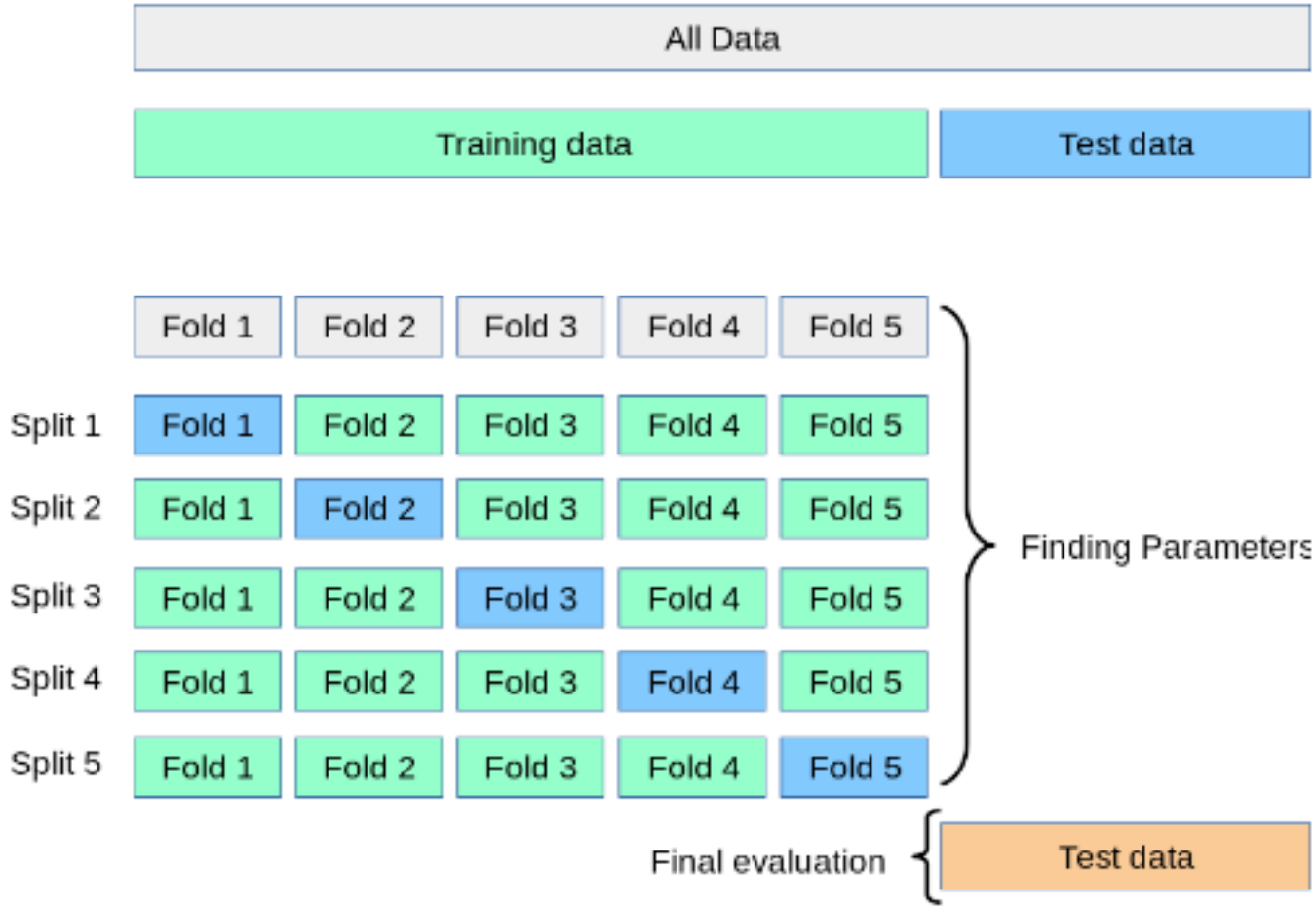
- train_test_split

```
df_train, df_test = train_test_split(diabetes_df, test_size= 0.20, random_state= 303)
```

The amount to dedicate to the Test set.
Here, 20% is set aside.

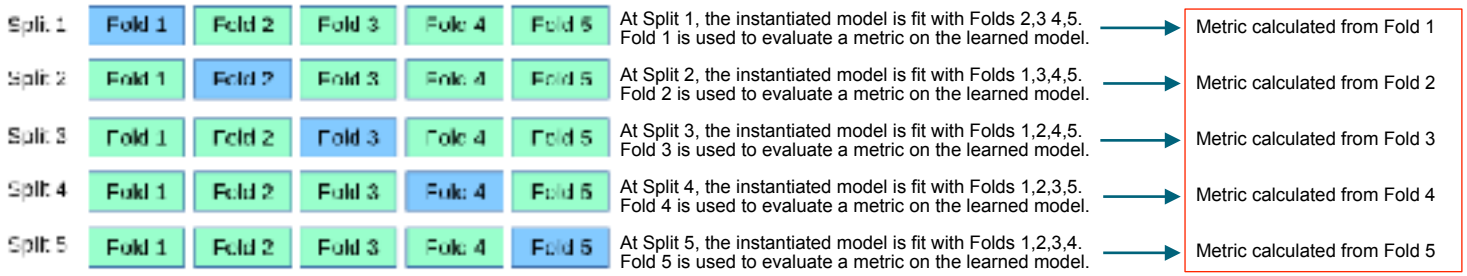
Set for reproducibility.
Tend to use the same:
303, 432, 80208

K-Fold Cross Validation



cross_val_score

- cross_val_score



cross_val_score

```
dec_tree_reg = DecisionTreeRegressor(random_state= 303)

dec_tree_reg_CV = cross_val_score(estimator= dec_tree_reg,
                                  X= X_train,
                                  y= y_train,
                                  scoring= 'neg_mean_squared_error'
                                 )
```

```
dec_tree_reg_CV
```

```
array([-5551.92957746, -7188.23943662, -6696.18309859, -5428.52857143,
       -5072.34285714])
```

scoring : ***str or callable, default=None***

A str (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)` which should return only a single value.

Similar to `cross_validate` but only a single metric is permitted.

If `None`, the estimator's default scorer (if available) is used.

`score(X, y[, sample_weight])`	Return the coefficient of determination of the prediction.

cross_val_score

```
dec_tree_reg = DecisionTreeRegressor(random_state= 303)

dec_tree_reg_CV = cross_val_score(estimator= dec_tree_reg,
                                  X= X_train,
                                  y= y_train,
                                  scoring= 'neg_mean_squared_error'
                                 )
```

```
dec_tree_reg_CV
```

```
array([-5551.92957746, -7188.23943662, -6696.18309859, -5428.52857143,
       -5072.34285714])
```

```
dec_tree_reg = DecisionTreeRegressor(random_state= 303, min_samples_leaf= 10)

dec_tree_reg_CV = cross_val_score(estimator= dec_tree_reg,
                                  X= X_train,
                                  y= y_train,
                                  scoring= 'neg_mean_squared_error'
                                 )
```

```
dec_tree_reg_CV
```

```
array([-3703.07744398, -3772.82485092, -3997.07431221, -4667.07561994,
       -4320.74800568])
```

cross_val_score

```
dec_tree_reg = DecisionTreeRegressor(random_state= 303)

dec_tree_reg_CV = cross_val_score(estimator= dec_tree_reg,
                                  X= X_train,
                                  y= y_train,
                                  scoring= 'neg_mean_squared_error'
                                 )
```

```
dec_tree_reg_CV
```

```
array([-5551.92957746, -7188.23943662, -6696.18309859, -5428.52857143,
       -5072.34285714])
```

Default scorer is coefficient of determination

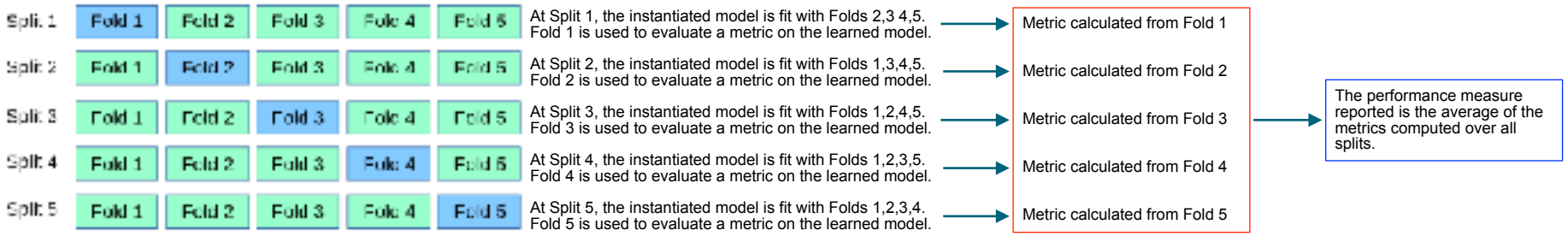
```
dec_tree_reg = DecisionTreeRegressor(random_state= 303)

dec_tree_reg_CV = cross_val_score(estimator= dec_tree_reg,
                                  X= X_train,
                                  y= y_train,
                                  #scoring= 'neg_mean_squared_error'
                                 )
```

```
dec_tree_reg_CV
```

```
array([-0.10277912, -0.12302283, -0.16843615,  0.05945196,  0.0086454 ])
```

K Fold CV Aggregates the Metrics

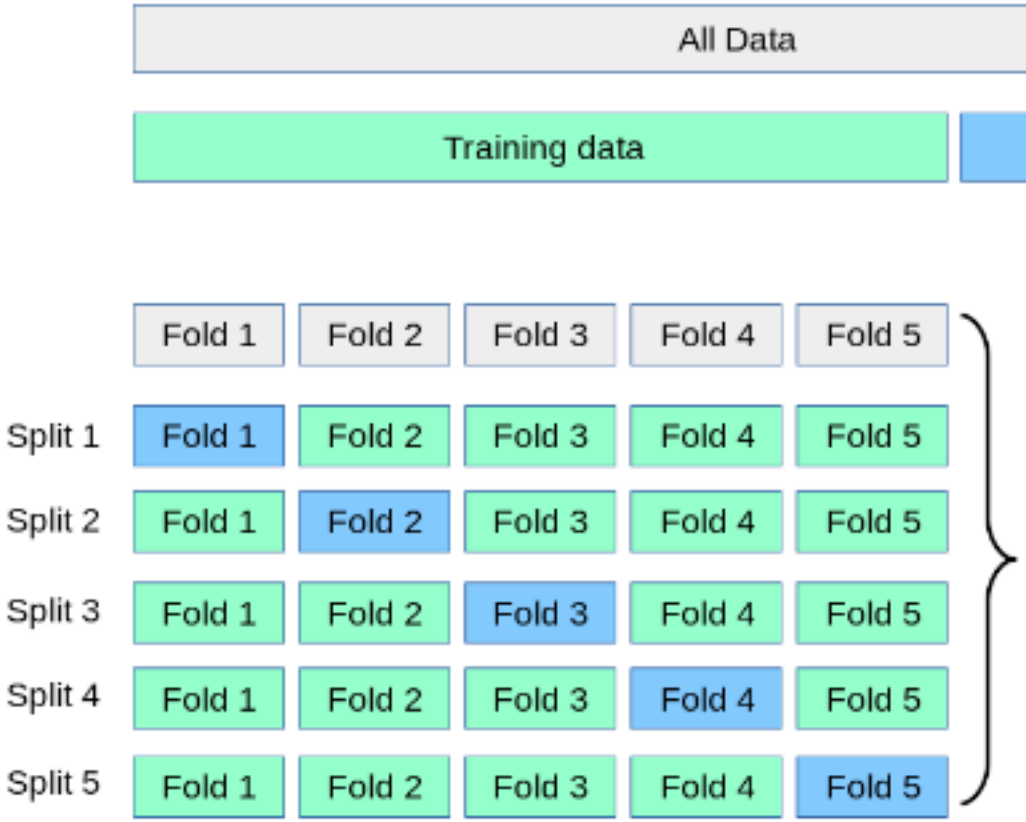


Hyperparameter Tuning

```
Total number of combinations to attempt: 50
Number of Trees          Fraction of Features
25                       [0.25, 0.5, 0.75, 1.0, 'auto']
50                       [0.25, 0.5, 0.75, 1.0, 'auto']
75                       [0.25, 0.5, 0.75, 1.0, 'auto']
100                      [0.25, 0.5, 0.75, 1.0, 'auto']
125                      [0.25, 0.5, 0.75, 1.0, 'auto']
150                      [0.25, 0.5, 0.75, 1.0, 'auto']
175                      [0.25, 0.5, 0.75, 1.0, 'auto']
200                      [0.25, 0.5, 0.75, 1.0, 'auto']
225                      [0.25, 0.5, 0.75, 1.0, 'auto']
250                      [0.25, 0.5, 0.75, 1.0, 'auto']
```

Instantiate a Random Forest Regressor with each combination (total of 50)
Execute k-Fold Cross Validation, and calculate the average error over the folds.

Grid Search with CV




```
Total number of combinations to attempt: 50
Number of Trees          Fraction of Features
25                       [0.25, 0.5, 0.75, 1.0, 'auto']
50                       [0.25, 0.5, 0.75, 1.0, 'auto']
75                       [0.25, 0.5, 0.75, 1.0, 'auto']
100                      [0.25, 0.5, 0.75, 1.0, 'auto']
125                      [0.25, 0.5, 0.75, 1.0, 'auto']
150                      [0.25, 0.5, 0.75, 1.0, 'auto']
175                      [0.25, 0.5, 0.75, 1.0, 'auto']
200                      [0.25, 0.5, 0.75, 1.0, 'auto']
225                      [0.25, 0.5, 0.75, 1.0, 'auto']
250                      [0.25, 0.5, 0.75, 1.0, 'auto']
```

For the Regression example, calculate the RMSE with the held out set.

Average the RMSE over all splits to get Total Error for the model trained.

Worked Example

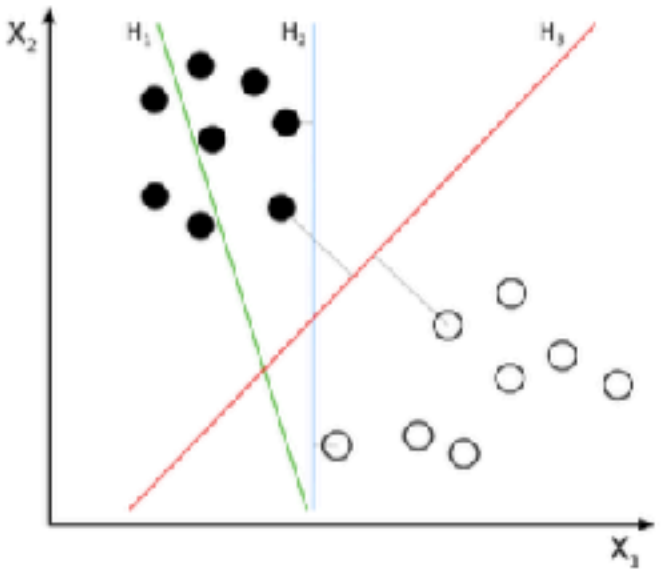
Classification versus Regression

- Predicting a discrete class label versus predicting a continuous quantity
- Linear regression can not predict label, and is replaced with logistic regression
- Introduce Support Vector Machines, but they are discussed in depth in two weeks

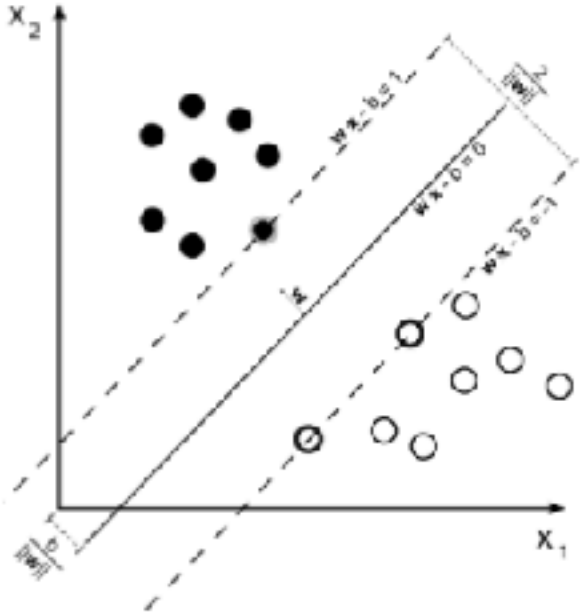
Classification

- Objective is to classify the categories of a data set
- Select the target variable which would be categorical
- Select the features from the remaining columns
  - Transform? Scale?
- Train the model to determine the categories using a subset of the data
- Test the model's effectiveness to predict using a different subset
  - How is effectiveness measured?

Support Vector Machines



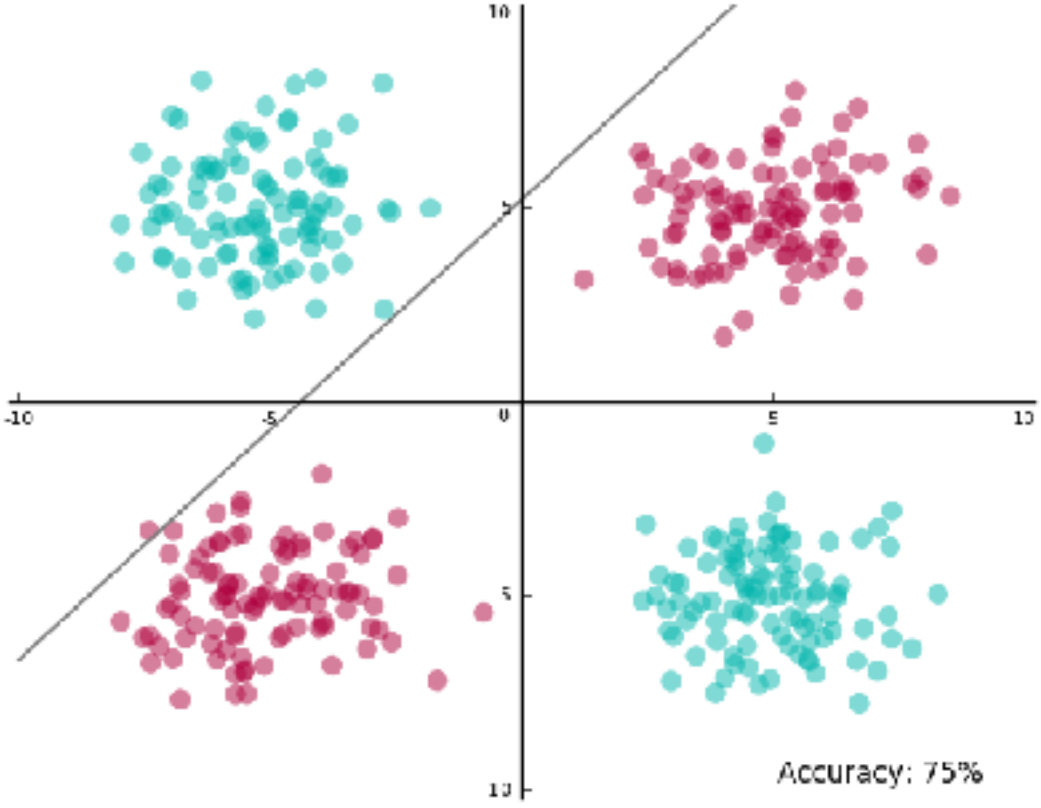
- H1 does not separate the classes
- H2 does, but only with a small margin
- H3 separates them with the max margin



- Maximum-margin hyperplane for an SVM trained with samples from 2 classes
- Samples on the margin are called the support vectors

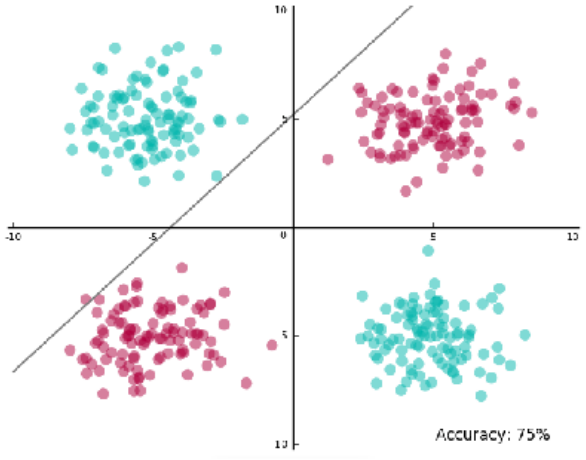
Normalize predictors between [-1, +1]

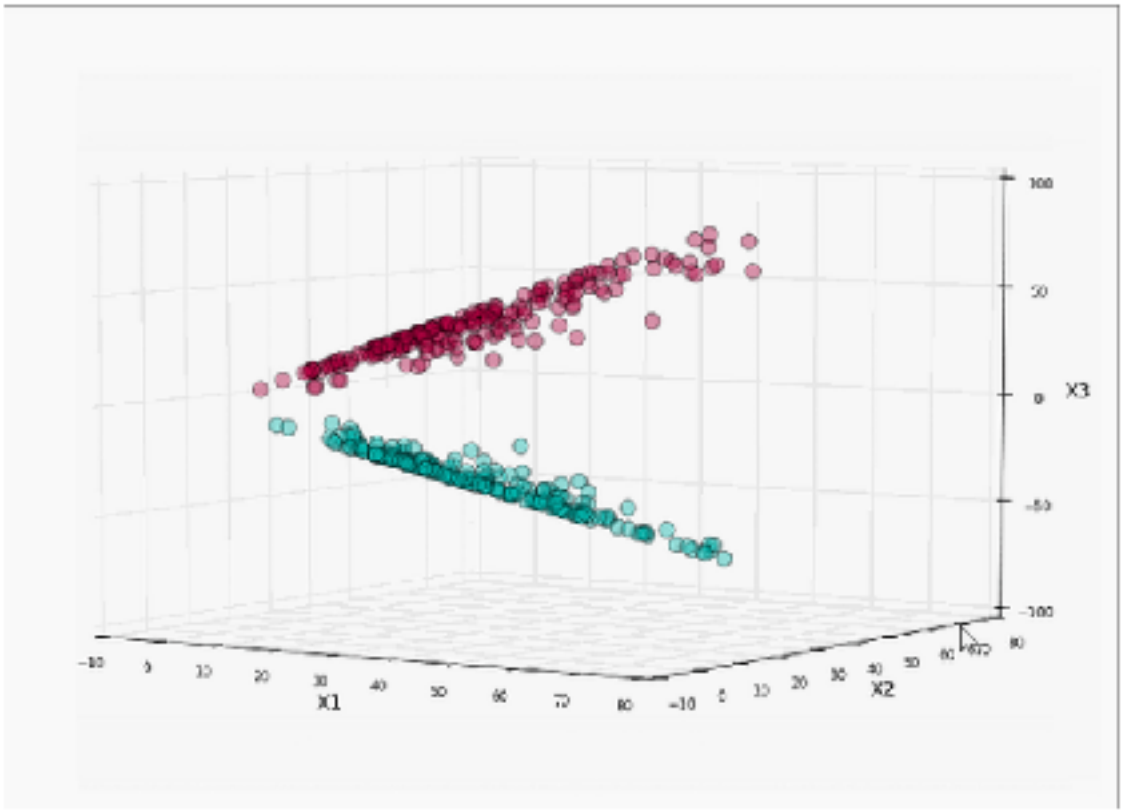
Support Vector Machines



Week 5 is dedicated to SVMs

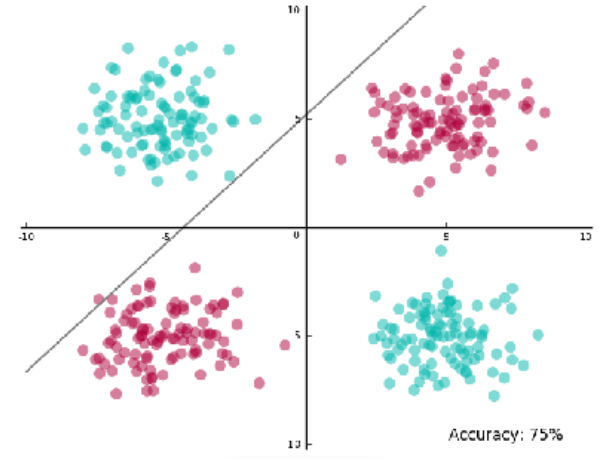
Support Vector Machines

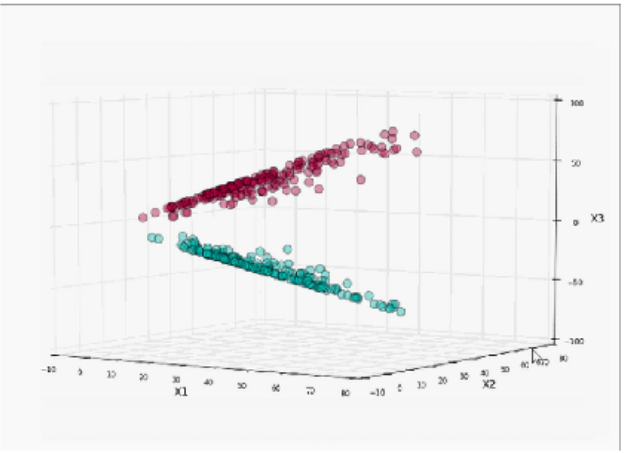


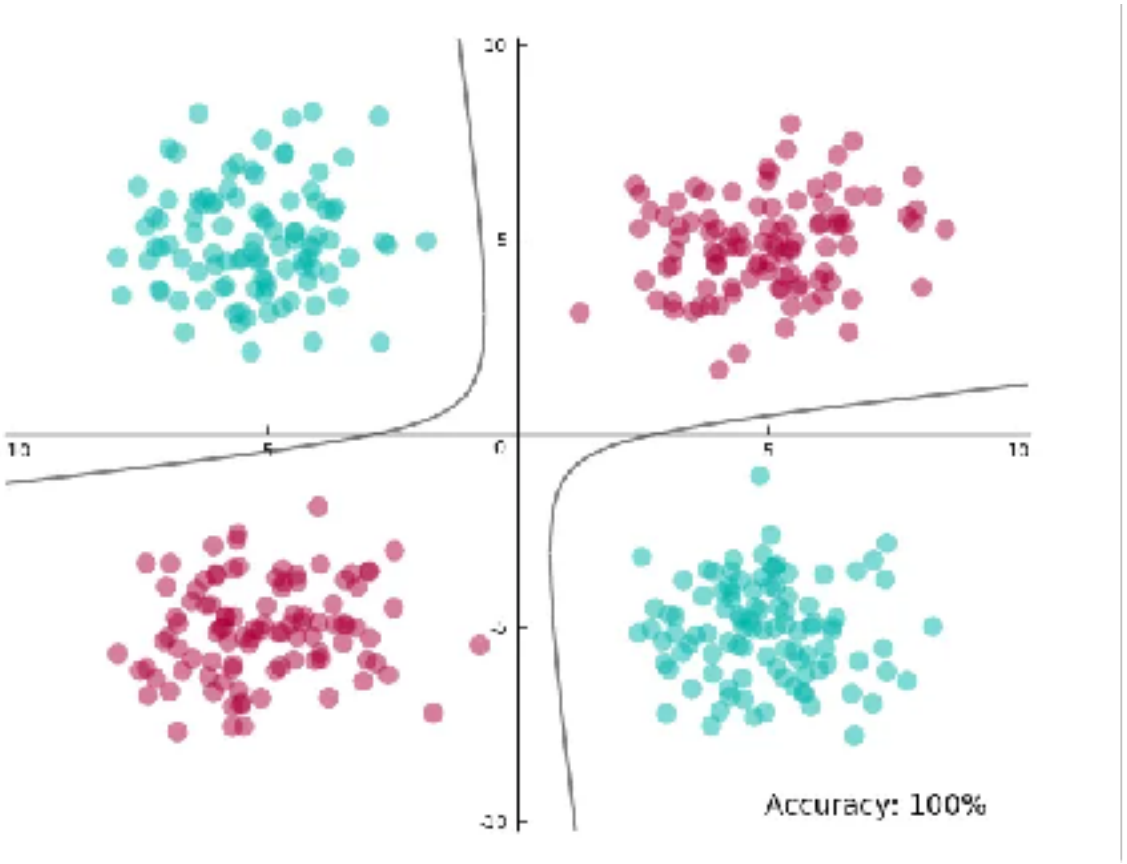


Week 5 is dedicated to SVMs

Support Vector Machines







Week 5 is dedicated to SVMs

Decision Trees

- Previously employed for regression
- What has to change for classification problems?

Classification Tree Regressor

Difference in Cost Function
Regression:

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{\text{node}} - y^{(i)})^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Classification:
Impurity in Target label

$$J(k,\ t_k) = \frac{n_L}{n_P}\text{Impurity}_L + \frac{n_R}{n_P}\text{Impurity}_R$$

$$\text{Gini} \quad = \quad 1 - \sum_{i=1}^{k} p_i^2$$

$$\text{Entropy} \quad = \quad \sum_{i=1}^{k} p_i \ \log(p_i)$$

Classification Tree Regressor

$$\Delta\text{Impurity} = \text{Impurity}_P - \left(\frac{n_L}{n_P}\text{Impurity}_L + \frac{n_R}{n_P}\text{Impurity}_R\right)$$

$$\text{Maximize}\,(\Delta\text{Impurity})$$

$$\text{Minimize}\left(\frac{n_L}{n_P}\text{Impurity}_L + \frac{n_R}{n_P}\text{Impurity}_R\right)$$

Classification versus Regression

Loss function is used to evaluate the performance of a model

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta(x_i) - y_i \right)^2$$

$$h_\theta(x_i) = \theta^T x_i$$

Classification versus Regression

Loss function is used to evaluate the performance of a model

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta(x_i) - y_i\right)^2$$

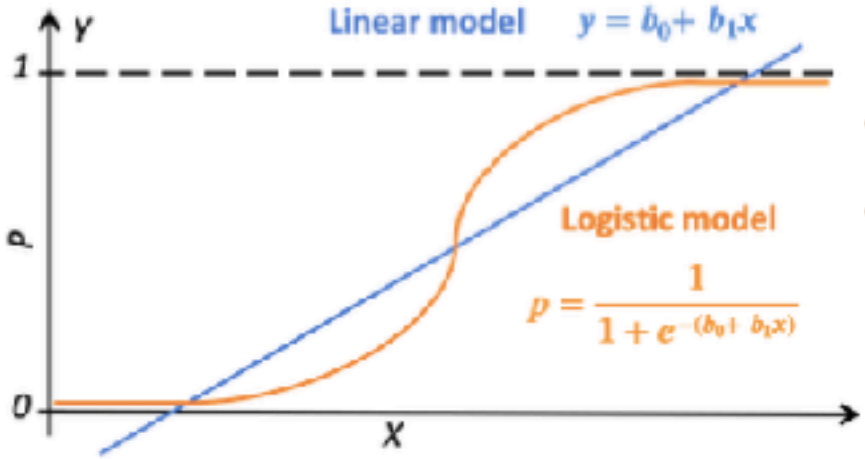
$$h_\theta(x_i) = \theta^T x_i$$

$$J(\theta) = -\frac{1}{n} \sum_{i}^{n} y_i \ log\left(h_\theta(x_i)\right) + (1 - y_i) \ log(1 - h_\theta(x_i))$$

$$h_\theta(x_i) = p(y_i = 1 | x_i) = p(x_i)$$

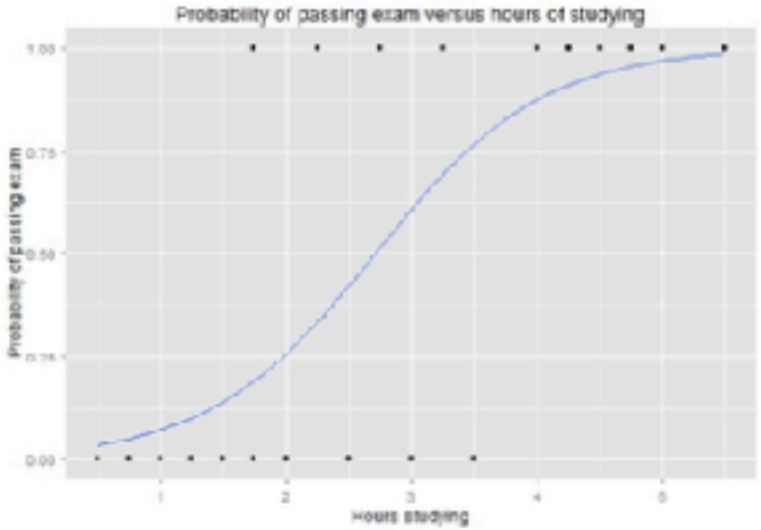
$$J(\theta) = -\frac{1}{n} \sum_{i}^{n} y_i \ log\left(p(x_i)\right) + (1 - y_i) \ log(1 - p(x_i))$$

Logistic Regression



- Logit model: $ln[\frac{p}{(1-p)}] = b_0 + b_1 X$

Models the probability of an event by assuming the log-odds of the event is a linear combination of variables



Transformation from log-odds to probability gives sigmoid function

- Estimated probability: $p = \frac{1}{1 + e^{-(b_0 + b_1 X)}}$
  - If $b_0 + b_1 X = 0$, then $p = 0.5$
  - As $b_0 + b_1 X$ gets large, $p -> 1$
  - As $b_0 + b_1 X$ gets small, $p -> 0$

Figure source: wikipedia

Logistic Regression

- Binary Output Variable
- Want to model: $h_\theta(x_i) = p(y_i = 1|x_i) = p(x_i)$
  - Let $p = \beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f = \vec{\beta} \cdot \vec{x}$
    - Linear function does not respect the bounds of p
    - Each increment of x would add or subtract from p

Logistic Regression

- Binary Output Variable
- Want to model: $h_\theta(x_i) = p(y_i = 1|x_i) = p(x_i)$
  - Let $p = \beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f = \vec{\beta} \cdot \vec{x}$
    - Linear function does not respect the bounds of p
    - Each increment of x would add or subtract from p

  - Let $log(p) = \beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f = \vec{\beta} \cdot \vec{x}$
    - Bounded on 0 to infinity
    - Each increment of x multiplies the probability

$$p = e^{\beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f} = e^{\beta_0 x_0} e^{\beta_1 x_1} \ldots e^{\beta_f x_f}$$

Logistic Regression

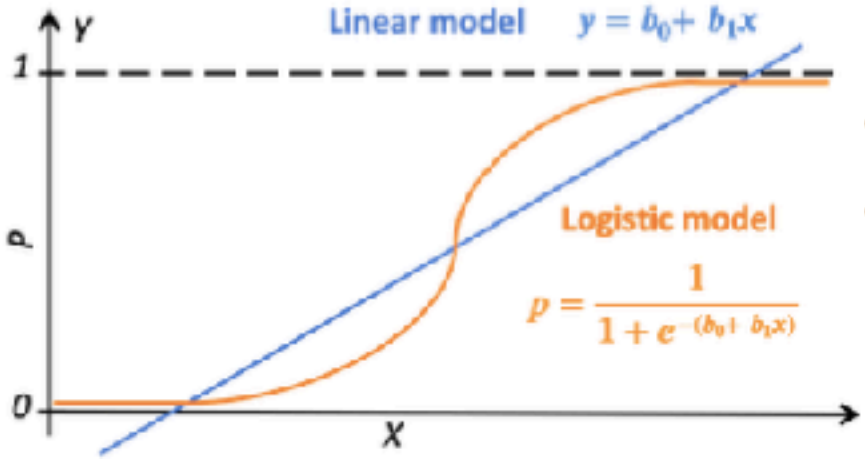
- Binary Output Variable
- Want to model: $h_\theta(x_i) = p(y_i = 1|x_i) = p(x_i)$
  - Let $p = \beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f = \vec{\beta} \cdot \vec{x}$
    - Linear function does not respect the bounds of p
    - Each increment of x would add or subtract from p
  - Let $log(p) = \beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f = \vec{\beta} \cdot \vec{x}$
    - Bounded on 0 to infinity
    - Each increment of x multiplies the probability

$$p = e^{\beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f} = e^{\beta_0 x_0} e^{\beta_1 x_1} \ldots e^{\beta_f x_f}$$

  - $log(Odds) = log\left(\frac{p}{1-p}\right) = \beta_0 x_0 + \beta_1 x_1 + \ldots + \beta_f x_f = \vec{\beta} \cdot \vec{x}$

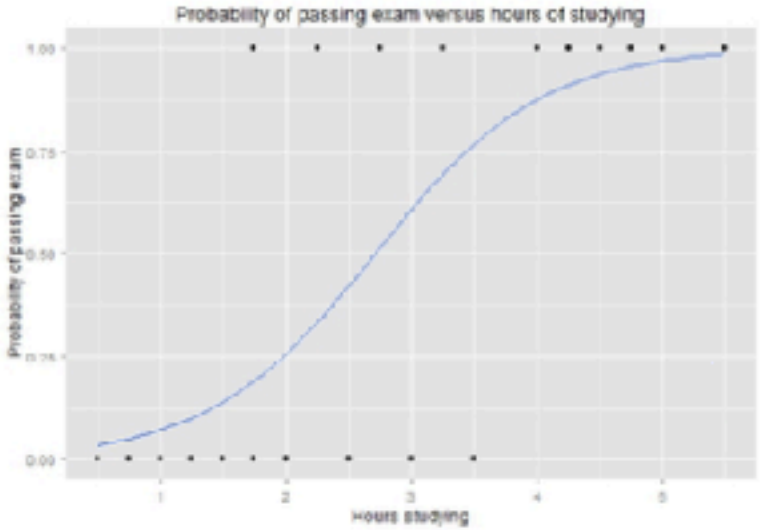
$$p = \frac{1}{1 + e^{\vec{\beta} \cdot \vec{x}}}$$

Logistic Regression



- Logit model: $ln[\frac{p}{(1-p)}] = b_0 + b_1 X$

Models the probability of an event by assuming the log-odds of the event is a linear combination of variables



Transformation from log-odds to probability gives sigmoid function

- Estimated probability: $p = \frac{1}{1 + e^{-(b_0 + b_1 X)}}$
  - If $b_0 + b_1 X = 0$, then $p = 0.5$
  - As $b_0 + b_1 X$ gets large, $p -> 1$
  - As $b_0 + b_1 X$ gets small, $p -> 0$

Figure source: wikipedia

Logistic Regression

- Unlike linear regression, a closed form solution does not exist
- Must use numerical approximation techniques to solve for parameters

$$L(\beta) = \prod_{s\ in\ y_i = 1} p(x_i) \ * \prod_{s\ in\ y_i = 0} (1 - p(x_i))$$

Optimize this likelihood function

$$L(\beta) = \prod_{s} \left(p(x_i)^{y_i} \ * \ (1 - p(x_i))^{1-y_i} \right)$$

$$l(\beta) = \sum_{i=1}^{n} y_i \, log(p(x_i)) + (1 - y_i) \, log(1 - p(x_i))$$

$$\frac{1}{1 + e^{-\beta x_i}}$$

$$l(\beta) = \sum_{i=1}^{n} y_i \beta x_i - log\left(1 + e^{\beta x_i}\right)$$

SciKit Documentation

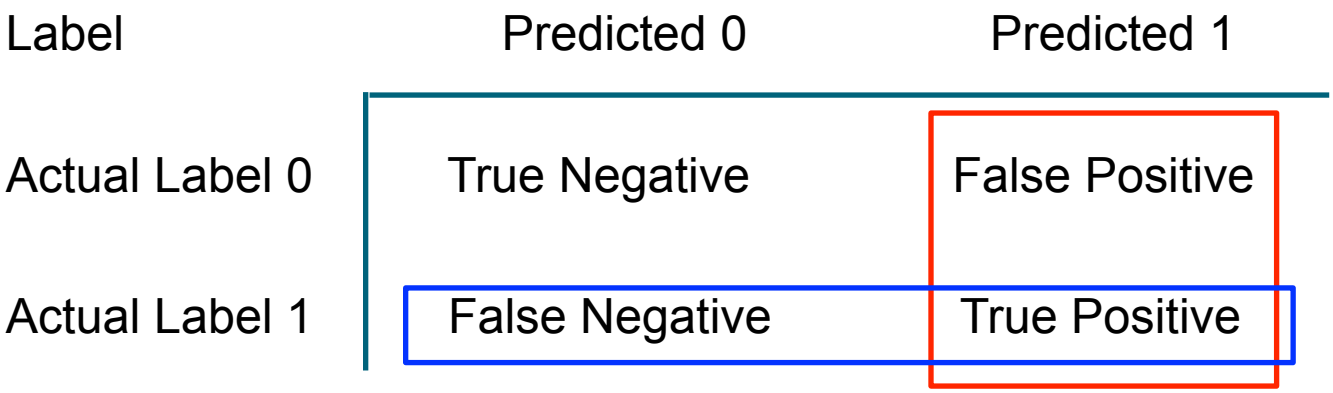
Performance Metrics

- Confusion Matrix
  - Recall
  - Precision
  - Accuracy
- ROC-AUC

Performance Metrics

Label	Predicted 0	Predicted 1
Actual Label 0	True Negative	False Positive
Actual Label 1	False Negative	True Positive

Performance Metrics



Precision = TP / (TP + FP)

Of those **predicted** as 1, what fraction are actually 1?

Recall = TP / (TP + FN)

Of those that are **actually** 1, what fraction are predicted 1?

Use these in collaboration.

Performance Metrics

Label	Predicted 0	Predicted 1
Actual Label 0	True Negative	False Positive
Actual Label 1	False Negative	True Positive

Threshold dependent. Typically, probability threshold is 0.5.

ROC-AUC

- Threshold independent

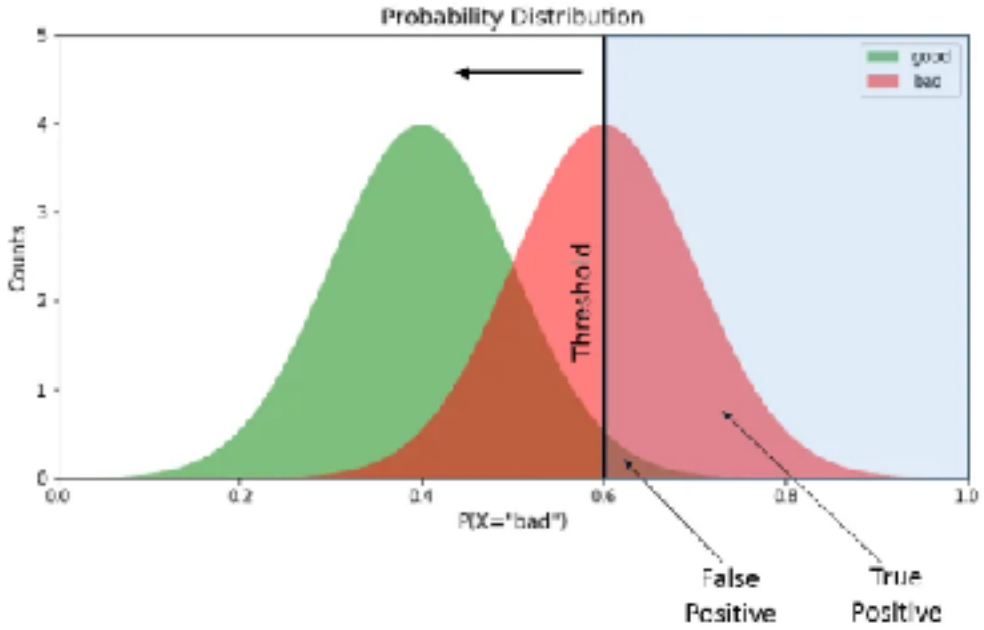
Label	Predicted 0	Predicted 1
Actual Label 0	True Negative	False Positive
Actual Label 1	False Negative	True Positive

False Positive Rate = FP / (TN + FP)

True Positive Rate = TP / (TP + FN)

Plot False Positive Rate versus True Positive Rate calculated at differing thresholds.

ROC-AUC



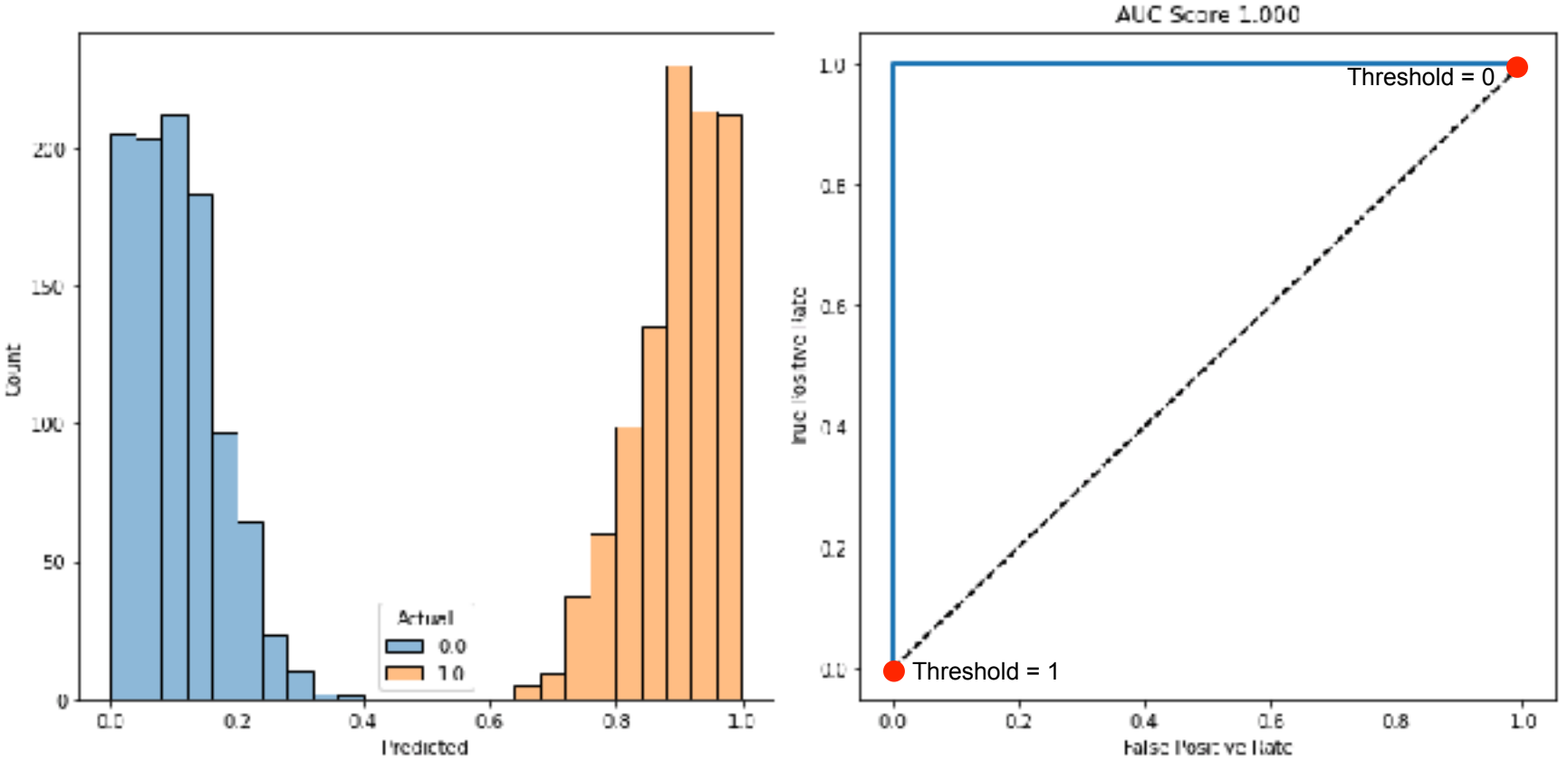
$$TPR = \frac{True\ Positive}{Total\ Positive}$$

$$FPR = \frac{False\ Positive}{Total\ Negative}$$

As threshold approaches 0, all instances are predicted as **in** class.
TP = *Total P* and *FP* = *Total N* and *FPR* = *TPR* = 1

As threshold approaches 1, all instances are predicted as **not in** class.
TP = 0 and *FP* = 0 and *FPR* = *TPR* = 0

ROC-AUC



Code Example

Appendix

- Hyperparameters
- Cross Validation
- Grid Search

Hyperparameters

- Configuration parameter used to control the learning process
  - depth of decision tree
  - number of estimators in an ensemble
  - learning rate

- Different from the parameters found during the model learning process
  - theta or beta values in regression
  - the features and split points in decision trees

Hyperparameters

```
from sklearn.tree import DecisionTreeRegressor

'Instantiate a decision tree regressor and examine the default hyperparameters'

DecisionTreeRegressor().get_params()
```

```
{'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

Using the default hyperparameter values for this decision tree (*max_depth* = None & *min_samples_leaf* = 1), the tree was grown until all leaves are pure when trained.

```
dec_tree_reg = DecisionTreeRegressor().fit(X= X_train, y= y_train)

dec_tree_reg.get_depth()
```

```
35
```

Hyperparameters

Updating the *min_samples_leaf* hyperparameter from 1 to 20 will not allow the tree to grow until the leaves are pure, reducing the depth of tree and overfitting to the training data and potentially increasing the generalizability.

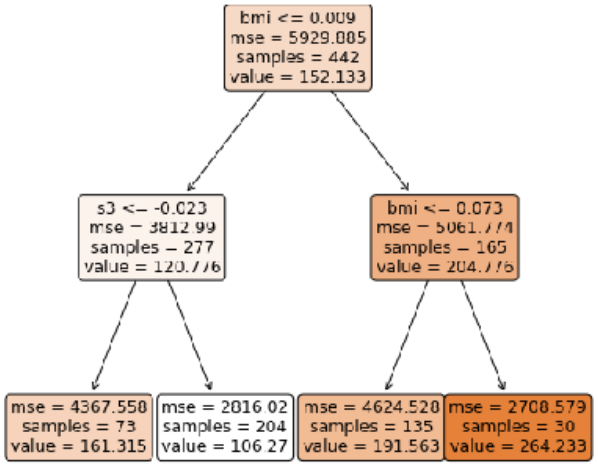
```
dec_tree_reg = DecisionTreeRegressor(min_samples_leaf= 20).fit(X= X_train, y= y_train)

dec_tree_reg.get_depth()
```

```
19
```

As seen in this simple example, the hyperparameters are preset by the user, represent the configuration of the model, and control the architecture of the decision tree regressor.

The image to the right shows the trained model parameters (features and split points) that are found during the learning phase based on a fixed set of hyperparameters. In this example, the depth was limited by defining *max_depth* = 2 when instantiating a DecisionTreeRegressor.

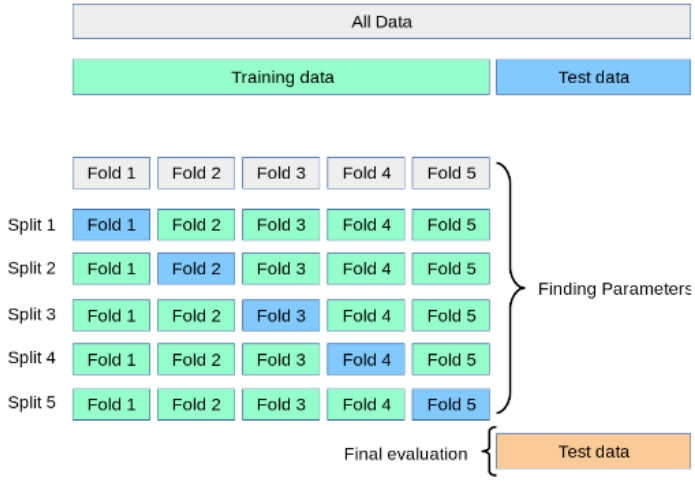


K-Fold Cross Validation

When evaluating different settings (hyperparameters) for estimators, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally. This way, knowledge about the test set can “leak” into the model and evaluation metrics no longer report on generalization performance. To solve this problem, yet another part of the dataset can be held out as a so-called “validation set”: training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.
However, by partitioning the available data into three sets instead of two, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.
A solution to this problem is a procedure called cross-validation. 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. The following procedure is followed for each of the k “folds”:

- A model is trained using k-1of 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 or RMSE).

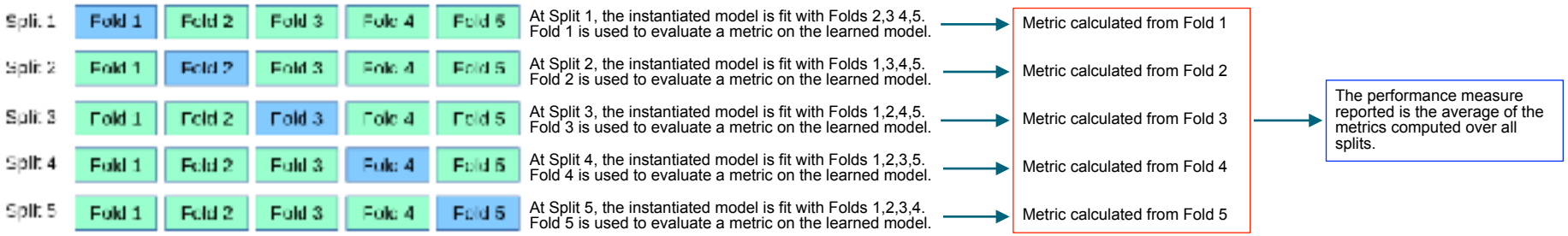
The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems such as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



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

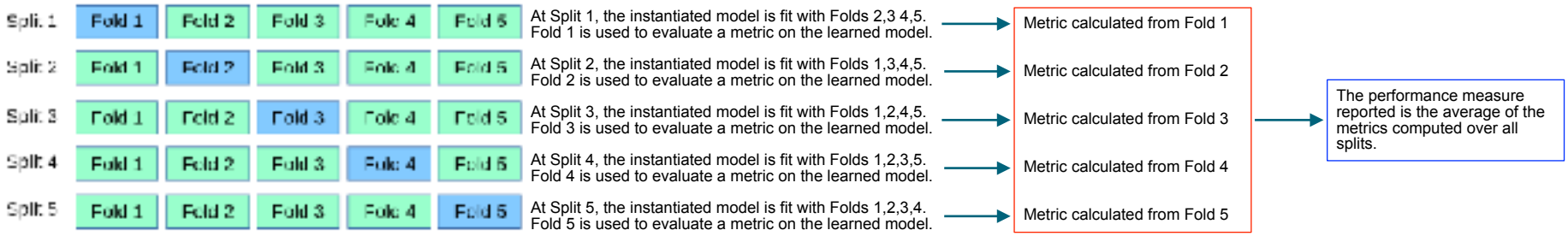
K-Fold Cross Validation

Training data is split into K equal partitions (folds).
At each split, the Kth fold is set aside as a proxy test set, and the remaining K-1 folds are used to fit an instantiated model.
Remember, the instantiated model is going to have a fixed hyperparameters. For example, the instantiated model could be a Random Forest with *n_estimators* = 100, *max_features* = 0.5, and *max_depth* = 7. These hyperparameters do not change over the K splits.
Therefore, the metrics calculated at each split are a result of the given hyperparameters and the differences in the data used to train the model at each split.



cross_val_score

With a given instantiated model with fixed hyperparameters, the Scikit function *cross_val_score* will return the evaluation scores calculated at each split (red outline).



Example usage of *cross_val_score.*
The instantiated model is a decision tree regressor with the default hyperparameters, K=5, and the evaluation metric calculated is the negated mean squared error.

```
dec_tree_reg = DecisionTreeRegressor()

cross_val_score(estimator= dec_tree_reg, X= X_train, y= y_train, cv= 5, scoring= 'neg_mean_squared_error')

array([-0.57528967, -0.53376643, -0.55292476, -0.52752564, -0.49486407])
```

Here, the instantiated model is a decision tree regressor with the *min_samples_leaf* updated to 20 from 1, K=5, and the evaluation metric calculated is the negated mean squared error. Notice the change to the calculated scores in each split.

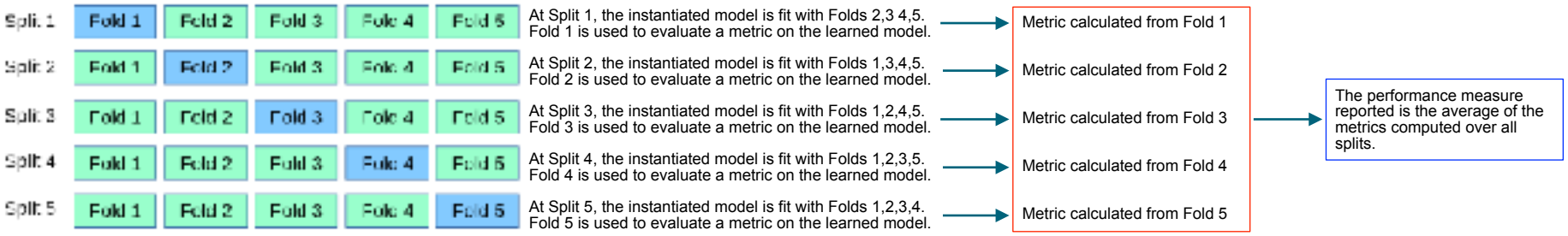
```
dec_tree_reg = DecisionTreeRegressor(min_samples_leaf= 20)

cross_val_score(estimator= dec_tree_reg, X= X_train, y= y_train, cv= 5, scoring= 'neg_mean_squared_error')

array([-0.40026224, -0.37392987, -0.39026294, -0.37972336, -0.35113034])
```

GridSeachCV

Conducts an exhaustive search to identify the hyperparameter values that construct an optimally performing estimator.
With a given algorithm and a grid of hyperparameter values, the Scikit function *GridSearchCV* will calculate the average evaluation metric (blue outline) for all possible combinations of the given hyperparameter values.



The combinatorics of the search space may be computationally demanding.
The following example shows a random forest regressor instantiated with a fixed random state.
The grid search will examine 10 values for number of trees in the ensemble, and 4 values for the number of features at each split, giving 40 possible combinations of hyperparameters.
With K=5 (cv=5 below), and 40 combinations of hyperparameters, a total of 200 random forest regressors are fit.
Each combination will be individually considered by K-Fold Cross Validation, giving 40 performance measures (blue outline). The hyperparameter combination with the optimum performance measure is chosen.

```
rand_forst = RandomForestRegressor(random_state= 98210)

param_grid = {'n_estimators': [13, 50, 73, 100, 113, 150, 175, 200, 225, 250],
              'max_features': [0.25, 0.5, 0.75, 1.0],
             }

grid_search = GridSearchCV(estimator= rand_forst, param_grid= param_grid, cv= 5, verbose= 5, scoring= 'neg_mean_squared_error')
```

GridSeachCV

Example continued.

```
rand_forst = RandomForestRegressor(random_state= 98210)

param_grid = {'n_estimators': [13, 50, 73, 100, 113, 150, 175, 200, 225, 250],
              'max_features': [0.25, 0.5, 0.75, 1.0],
             }

grid_search = GridSearchCV(estimator= rand_forst, param_grid= param_grid, cv= 5, verbose= 5, scoring= 'neg_mean_squared_error')
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
[CV 1/5] END max_features=0.25, n_estimators=25;, score=-0.286 total time=   1.0s
[CV 2/5] END max_features=0.25, n_estimators=25;, score=-0.262 total time=   0.9s
[CV 3/5] END max_features=0.25, n_estimators=25;, score=-0.256 total time=   0.8s
[CV 4/5] END max_features=0.25, n_estimators=25;, score=-0.271 total time=   0.8s
[CV 5/5] END max_features=0.25, n_estimators=25;, score=-0.247 total time=   0.8s
[CV 1/5] END max_features=0.25, n_estimators=50;, score=-0.271 total time=   2.2s
[CV 2/5] END max_features=0.25, n_estimators=50;, score=-0.246 total time=   1.7s
[CV 3/5] END max_features=0.25, n_estimators=50;, score=-0.256 total time=   1.6s
[CV 4/5] END max_features=0.25, n_estimators=50;, score=-0.268 total time=   1.7s
[CV 5/5] END max_features=0.25, n_estimators=50;, score=-0.236 total time=   1.6s
[CV 1/5] END max_features=0.25, n_estimators=75;, score=-0.264 total time=   2.3s
[CV 2/5] END max_features=0.25, n_estimators=75;, score=-0.245 total time=   2.5s
[CV 3/5] END max_features=0.25, n_estimators=75;, score=-0.245 total time=   2.5s
[CV 4/5] END max_features=0.25, n_estimators=75;, score=-0.255 total time=   2.9s
[CV 5/5] END max_features=0.25, n_estimators=75;, score=-0.232 total time=   2.5s
[CV 1/5] END max_features=0.25, n_estimators=100;, score=-0.262 total time=   3.1s
[CV 2/5] END max_features=0.25, n_estimators=100;, score=-0.244 total time=   3.1s
[CV 3/5] END max_features=0.25, n_estimators=100;, score=-0.242 total time=   3.1s
                                      ⋮
[CV 1/5] END max_features=1.0, n_estimators=225;, score=-0.278 total time=  22.1s
[CV 2/5] END max_features=1.0, n_estimators=225;, score=-0.266 total time=  22.8s
[CV 3/5] END max_features=1.0, n_estimators=225;, score=-0.256 total time=  23.7s
[CV 4/5] END max_features=1.0, n_estimators=225;, score=-0.267 total time=  22.0s
[CV 5/5] END max_features=1.0, n_estimators=225;, score=-0.236 total time=  22.7s
[CV 1/5] END max_features=1.0, n_estimators=250;, score=-0.278 total time=  25.0s
[CV 2/5] END max_features=1.0, n_estimators=250;, score=-0.266 total time=  25.5s
[CV 3/5] END max_features=1.0, n_estimators=250;, score=-0.256 total time=  25.7s
[CV 4/5] END max_features=1.0, n_estimators=250;, score=-0.267 total time=  26.0s
[CV 5/5] END max_features=1.0, n_estimators=250;, score=-0.236 total time=  26.2s
```

```
grid_search.best_params_
```

```
{'max_features': 0.25, 'n_estimators': 225}
```

The highlighted bands show the K=5 Cross Validation output for each hyperparameter combination (verify max_features and n_estimators are equal per highlighted band).

GridSeachCV

Examine the results dictionary for much more details of the grid search.

```
grid_search.cv_results_
```

Performance Metrics (average and standard deviations over all folds per combination, and the final rank of the average)

```
'mean_test_score': array([-0.26439592, -0.25275277, -0.2483603 , -0.24648186, -0.24537888,
       -0.24500791, -0.24482469, -0.24490698, -0.24479891, -0.24502825,
       -0.26554886, -0.25849765, -0.25492308, -0.25330753, -0.25147685,
       -0.25860344, -0.24972387, -0.24915631, -0.24905472, -0.2491219 ,
       -0.26588737, -0.25975725, -0.25738451, -0.25672386, -0.25584343,
       -0.2557367 , -0.25523411, -0.25515177, -0.25489779, -0.25471817,
       -0.26914938, -0.26410751, -0.26208797, -0.26047031, -0.25917955,
       -0.25858419, -0.25819904, -0.25777689, -0.25784078, -0.25794673]),
'std_test_score': array([0.01323074, 0.0112476 , 0.01089882, 0.01105341, 0.01158265,
       0.01155395, 0.01142925, 0.01163423, 0.01186142, 0.01189335,
       0.01363499, 0.01264912, 0.01356229, 0.0132586 , 0.01339602,
       0.01369723, 0.01253086, 0.01294698, 0.01304384, 0.01307601,
       0.01145602, 0.01260342, 0.01371576, 0.01344758, 0.01307803,
       0.01335125, 0.0132089 , 0.01316792, 0.01296817, 0.01270717,
       0.01283793, 0.01172526, 0.01149844, 0.01188628, 0.01160849,
       0.01174877, 0.01190788, 0.01202346, 0.01203521, 0.0122386 ]),
'rank_test_score': array([37, 15,  8,  7,  6,  4,  2,  3,  1,  5, 38, 30, 19, 16, 14, 13, 12,
       11,  9, 10, 39, 33, 25, 24, 23, 22, 21, 20, 18, 17, 40, 36, 35, 34,
       32, 31, 29, 26, 27, 28], dtype=int32)}
```

Hyperparameter combinations

```
'params': [{'max_features': 0.25, 'n_estimators': 25},
 {'max_features': 0.25, 'n_estimators': 50},
 {'max_features': 0.25, 'n_estimators': 75},
 {'max_features': 0.25, 'n_estimators': 100},
 {'max_features': 0.25, 'n_estimators': 125},
 {'max_features': 0.25, 'n_estimators': 150},
 {'max_features': 0.25, 'n_estimators': 175},
 {'max_features': 0.25, 'n_estimators': 200},
 {'max_features': 0.25, 'n_estimators': 225},
 {'max_features': 0.25, 'n_estimators': 250},
 {'max_features': 0.5, 'n_estimators': 25},
 {'max_features': 0.5, 'n_estimators': 50},
 {'max_features': 0.5, 'n_estimators': 75},
 {'max_features': 0.5, 'n_estimators': 100},
 {'max_features': 0.5, 'n_estimators': 125},
 {'max_features': 0.5, 'n_estimators': 150},
 {'max_features': 0.5, 'n_estimators': 175},
 {'max_features': 0.5, 'n_estimators': 200},
 {'max_features': 0.5, 'n_estimators': 225},
 {'max_features': 0.5, 'n_estimators': 250},
 {'max_features': 0.75, 'n_estimators': 25},
 {'max_features': 0.75, 'n_estimators': 50},
 {'max_features': 0.75, 'n_estimators': 75},
 {'max_features': 0.75, 'n_estimators': 100},
 {'max_features': 0.75, 'n_estimators': 125},
 {'max_features': 0.75, 'n_estimators': 150},
 {'max_features': 0.75, 'n_estimators': 175},
 {'max_features': 0.75, 'n_estimators': 200},
 {'max_features': 0.75, 'n_estimators': 225},
 {'max_features': 0.75, 'n_estimators': 250},
 {'max_features': 1.0, 'n_estimators': 25},
 {'max_features': 1.0, 'n_estimators': 50},
 {'max_features': 1.0, 'n_estimators': 75},
 {'max_features': 1.0, 'n_estimators': 100},
 {'max_features': 1.0, 'n_estimators': 125},
 {'max_features': 1.0, 'n_estimators': 150},
 {'max_features': 1.0, 'n_estimators': 175},
 {'max_features': 1.0, 'n_estimators': 200},
 {'max_features': 1.0, 'n_estimators': 225},
 {'max_features': 1.0, 'n_estimators': 250}],
```